インテル® アーキテクチャ対応 インテル® インテグレーテッド・パフォーマンス・プリミティブ（IPP）

リファレンス・マニュアル

第２巻：画像および動画処理

インテル® アーキテクチャ対応 インテル® インテグレーテッド・ パフォーマンス・プリミティブ（IPP）

リファレンス・マニュアル

第 2 巻 : 画像および動画処理

資料番号 : A70805-3304J

Web : http://www.intel.co.jp/jp/developer/（日本語）
: http://developer.intel.com（英語）

バージョン	バージョン情報	日付
-1001	インテル® インテグレーテッド・パフォーマンス・プリミティブ・リリース (インテル® IPP) 1.0 ベータの説明。	2000 年 7 月
-1002	インテル IPP 1.0 ベータ 2 の説明。アルファ合成、カラー・ツイスト、ガンマ補正、FFT/DFT/DCT 関数を追加。	2000 年 9 月
-1003	インテル IPP 1.0 最終リリースの説明。新しい機能 (ウェーブレット変換、コンピュータ・ビジョン関数、拡張されたジオメトリ変換) を追加。新しいデータ初期設定関数、算術演算関数、カラー変換関数も追加。	2001 年 2 月
-1101	インテル IPP リリース 1.1 ベータの説明。JPEG コーデック関数を追加。	2001 年 4 月
-2001	インテル IPP 2.0 ベータ・リリースについて説明。H.263+ および MPEG-4 デコーダ向けのビデオ処理関数と、JPEG コーデック向けのウェーブレット変換関数を追加。	2001 年 8 月
-2002	インテル IPP 2.0 ゴールド・リリースについて説明。ライブラリの共通の関数を追加。算術関数、変換関数、フィルタリング関数、統計関数、JPEG コーデック関数の新しい変種を追加。	2001 年 11 月
-3001	インテル IPP 3.0 プレベータ版について説明。ドメイン・ライブラリとの互換性レイヤをサポートする関数型を追加。JPEG200 コーデック関数を拡張。ビデオ・デコード関数を追加。	2002 年 4 月
-3002	インテル IPP 3.0 ベータ・リリースについて説明。	2002 年 6 月
-3003	インテル IPP 3.0 ベータの改訂について説明。ビデオ・エンコード関数を追加。	2002 年 9 月
-3004	インテル IPP 3.0 ゴールド・リリースについて説明。一連の MPEG-4 ビデオ処理関数を拡張。	2002 年 11 月

目 次

第 1 章　概要

第 2 章　インテル® インテグレーテッド・パフォーマンス・プリミティブの概念

第 3 章　サポート関数

第 4 章　画像データ交換関数と初期化関数

第 5 章　画像の算術演算と論理演算

第 6 章　画像のカラー変換

第 7 章　しきい値演算と比較演算

第 8 章　モルフォロジー演算

第 9 章　フィルタリング関数

第 10 章　画像の線形変換

第 11 章　画像の統計関数

第 12 章　画像のジオメトリ変換

第 13 章　ウェーブレット変換

第 14 章　コンピュータ・ビジョン

第 15 章　画像圧縮関数

第 16 章　H.263 ビデオ・デコーダ

第 17 章 MPEG-4 ビデオ・デコーダ

第 18 章　ビデオ符号化

概要 1

本書では、2 次元信号を操作するインテル® アーキテクチャに対応した画像および動画処理用インテル® インテグレーテッド・パフォーマンス・プリミティブ（インテル® IPP）の構造、動作、関数について説明する。本書は、インテル IPP リファレンス・マニュアル全 3 巻のうちの第 2 巻である（第 1 巻で信号処理用インテル IPP について、第 3 巻でスモール・マトリックスについて説明している）。インテル IPP ソフトウェア・パッケージは、多くの関数をサポートしている。これらの関数のパフォーマンスは、（特に MMX® テクノロジ、ストリーミング SIMD 拡張命令（SSE）、ストリーミング SIMD 拡張命令 2（SSE2）を使用している）インテル・アーキテクチャ上では大きく向上する。

インテル IPP 画像および動画処理ソフトウェアは、ピクセルの 2 次元配列に対して実行される、オーバーヘッドの小さい高性能の操作を集めたものである。

本書では、インテル IPP の概念と、画像および動画処理の領域で使用されるデータ・タイプの定義と動作モデルについて説明し、IPP 画像および動画処理関数について詳しく説明する。

本章では、インテル IPP ソフトウェアの概要と、本書の構成について説明する。

このソフトウェアについて

インテル IPP ソフトウェアは、MMX テクノロジとストリーミング SIMD 拡張命令のコアを構成する SIMD（Single-Instruction, Multiple Data）命令の並列処理をフルに利用する。これらの技術は、高い処理能力を必要とする信号、画像および動画処理アプリケーションのパフォーマンスを向上させる。

ハードウェアとソフトウェアの必要条件

インテル IPP ソフトウェアは、インテル・アーキテクチャ・プロセッサと Microsoft* Windows* 98、Windows 2000、Windows XP、Windows NT*、または Linux* を搭載したパーソナル・コンピュータ上で動作する。インテル IPP は、C または C++ で記述されたユーザ・アプリケーションまたはライブラリに統合して使用できる。

サポートしているプラットフォーム

インテル IPP ソフトウェアは、Windows および Linux プラットフォーム上で動作する。本書では、関数と変数の宣言に使用されるコードと構文を ANSI C 形式で記述している。ただし、プロセッサやオペレーティング・システムの種類に応じて、必要とされるインテル IPP のバージョンは多少異なる。

技術サポート

インテル IPP には、製品の特徴、ホワイトペーパー、技術記事など、製品に関する総合的な情報が適宜掲載される製品 Web サイトが用意されている。最新情報については、以下のサイトを参照のこと。
http://www.intel.co.jp/jp/developer/software/products/
または http://developer.intel.com/software/products/（英語）

インテルでは、基本操作のヒント、製品に関する確認済みの問題点、製品のエラッタ、ライセンス情報、ユーザ・フォーラムなどの大量のセルフヘルプ情報を利用できるサポート Web サイトも提供している（http://support.intel.co.jp/jp/support/ または http://support.intel.com/support/（英語）を参照のこと）。

ユーザ登録を行うと、インテル・プレミア・サポートによる 1 年間の技術サポートと製品アップデートのサービスが受けられる。インテル・プレミア・サポートは、以下のサービスを提供する、双方向型の問題管理 / コミュニケーション Web サイトである。

- 問題点の送信とその状態の検討
- 製品のアップデートのダウンロード（1 日 24 時間）

ユーザ登録、インテルへの問い合わせ、製品サポートについては、以下のサイトを参照のこと。http://premier.intel.com/（英語）

本書について

本書には、インテル IPP ソフトウェアで使用されている画像および動画処理の概念の基礎知識と、画像および動画処理用インテル IPP の各関数の詳しい説明が記載されている。IPP の関数は、機能によってグループ分けされている。関数の各グループについては、第 3 章 ～ 第 18 章で説明する。

本書の構成

本書は、次の各章で構成されている。

第 1 章	概要。インテル IPP ソフトウェアの概要、本書の構成、表記規則について説明する。
第 2 章	インテル® インテグレーテッド・パフォーマンス・プリミティブの概念。インテル IPP での画像処理の基礎になる概念と、サポートしているデータ形式および動作モードについて説明する。
第 3 章	サポート関数。インテル IPP ソフトウェアの動作をサポートするための関数 （メモリ割り当て関数、ステータス情報関数など）について説明する。
第 4 章	画像データ交換関数と初期化関数。画像の設定、コピー、変換、拡大 / 縮小、特定の画像の初期化に使用される画像処理プリミティブ関数について説明する。
第 5 章	画像の算術演算と論理演算。算術演算、論理演算、アルファ合成演算を使用してピクセル値を修正する、インテル IPP 画像処理関数について説明する。
第 6 章	画像のカラー変換。インテル IPP のカラー・スペース変換演算について説明する。
第 7 章	しきい値演算と比較演算。しきい値演算と画像の比較演算を実行する関数について説明する。
第 8 章	モルフォロジー演算。収縮および膨張モルフォロジー演算について説明する。
第 9 章	フィルタリング関数。リニア・フィルタを使用するインテル IPP フィルタリング関数とノンリニア・フィルタを使用するインテル IPP フィルタリング関数について説明する。
第 10 章	画像の線形変換。画像処理用インテル IPP でサポートしている、高速フーリエ変換（FFT）、離散フーリエ変換（DFT）、離散コサイン変換の各関数について説明する。
第 11 章	画像の統計関数。画像のノルムとモーメントなど、画像の統計的な特性を計算する IPP の関数について説明する。
第 12 章	画像のジオメトリ変換。サポートしている画像のジオメトリ変換について説明する。
第 13 章	ウェーブレット変換。サポートしている画像のウェーブレット分解および再構成について説明する。
第 14 章	コンピュータ・ビジョン。コンピュータ・ビジョン・アプリケーション専用の IPP 関数について説明する。
第 15 章	画像圧縮関数。JPEG および JPEG2000 規格に基づく静止画像の圧縮とコード化を実行するインテル IPP 関数について説明する。

第 16 章	H.263 ビデオ・デコーダ。汎用ビデオ処理および H.263+ デコーダをサポートするインテル IPP 関数について説明する。
第 17 章	MPEG-4 ビデオ・デコーダ。ISO/IEC 14496-2 MPEG- ビデオ・デコーダ向けに開発されたインテル IPP 関数について説明する。
第 18 章	ビデオ符号化。MPEG-1、MPEG-2、MPEG-4 規格に基づく動画データのコード化とデコードに使用されるインテル IPP 関数について説明する。

本書は、この他に、インテル IPP 関数による特殊な事例の処理について説明した特殊な事例の処理と、、、索引で構成されている。

関数の説明

第 3 章～第 18 章では、各関数を短い名前で（ippi プリフィックスと記述子なしで）示し、関数の目的について簡単に説明する。次に、関数の呼び出しシーケンス、引数の定義、関数の目的の詳しい説明を示す。各関数の説明には、以下の項目がある。

引数	関数のすべての引数について説明する。
説明	関数の定義を示し、その関数によって実行される処理について詳しく説明する。コードの例と関数が実行する式を示す場合もある。
戻り値	関数が実行された結果設定される、ステータス・コードの値について説明する。

本書の対象読者

本書は、画像および動画処理アプリケーション、画像処理ライブラリ、異種領域アプリケーションの開発者を対象とする。読者は、C の使用経験と、画像および動画処理の用語と原理に関する知識が必要である。

オンライン版

本書は、PDF（Portable Document Format）電子形式で提供される。本書のハードコピー版が必要な場合は、Adobe Acrobat*（本書のオンライン表示用ツール）の印刷機能を使用してファイルを印刷できる。

関連資料

画像処理の概念とアルゴリズムの詳細については、に示した資料を参照のこと。

表記規則

本書の表記規則には、以下のものがある。

- 本文とコードを区別するための字体の規則
- さまざまなアイテムの命名規則

字体の規則

本書では、以下の字体の規則を使用する。

`THIS TYPE STYLE`	本文中でインテル IPP の不変識別子を示す。例：`IPPI_INTER_LINEAR`
`This type style`	`IppiSize` のように、大文字と組み合わせて構造体の名前を示す。また、関数名、コード例、呼び出し文にも使用する。例：`ippiMomentInitAlloc()`
`This type style`	引数とパラメータの説明の中の変数。例：*`value`*, *`srcStep`*

命名規則

インテル IPP ソフトウェアは、さまざまなアイテムに対して次の命名規則を使用する。

- 不変識別子は大文字で示す。例：`IPPI_INTER_CUBIC`
- すべての画像および画像処理専用の構造体と列挙子には、`Ippi` プリフィックスが付く。これに対して、インテル IPP ソフトウェア全体に共通の構造体と列挙子には、`Ipp` プリフィックスが付く。例：`IppiPoint`, `IppDitherType`
- すべての画像および画像処理関数の名前には、`ippi` プリフィックスが付加される。コード例の中では、このプリフィックスにより、インテル IPP インターフェイス関数とアプリケーションの関数を区別できる。

注： 本書では、関数名の *`ippi`* プリフィックスは、コードの例には常に使用されるが、本文中では関数グループを参照する際に通常省略されることがある。

- 関数名の各部分の最初の文字は、アンダースコアなしの大文字である。例：`ippiGetSpatialMoment`

インテル IPP の関数名の構造の詳細は、第 2 章の関数の命名を参照のこと。

インテル® インテグレーテッド・パフォーマンス・プリミティブの概念 2

本章では、インテル® インテグレーテッド・パフォーマンス・プリミティブ（インテル® IPP）ソフトウェアの目的と構造について説明し、インテル IPP の画像および動画処理部分で使用されるいくつかの基本的な概念について解説する。また、サポートされるデータ形式と動作モード、関数の命名規則についても説明する。

基本的な機能

インテグレーテッド・パフォーマンス・プリミティブは、インテル® パフォーマンス・ライブラリの他の製品と同様に、特定の領域の操作を実行する高性能コードを集めたものである。インテル IPP が低レベルのステートレス・インターフェイスを提供する点が他と異なる。

インテル IPP は、インテル・パフォーマンス・ライブラリの開発と使用の経験に基づいて開発され、以下の主な特長を備えている。

- インテル IPP は、信号処理、画像および動画処理、スモール・マトリックスの操作など、複数の異なる領域のアプリケーションを作成するための基本的な低レベルの関数を提供する。
- インテル IPP の関数は、異なるアプリケーション領域を参照するプリミティブについても、共通の命名規則やよく似たプロトタイプの合成など、同じインターフェイス規則を適用する。
- インテル IPP の関数は、アプリケーション・プログラムのパフォーマンスの向上に最適な抽象レベルを使用する。

プログラムのパフォーマンスを向上させるために、インテル IPP の関数は、インテルアーキテクチャ・プロセッサの利点をすべて利用するように最適化されている。また、ほとんどのインテル IPP は、複雑なデータ構造体を使用しないため、全体的な実行オーバーヘッドが小さくなる。

インテル IPP ソフトウェアは、画像を抽象したものではなく、2 次元のデータ配列として操作する。したがって、インテル IPP は、他のほとんどの画像処理ライブラリより低レベルの画像処理関数を提供する。これには、以下のような主なメリットがある。

- 開発者が画像データおよびアーキテクチャを記述している固有の構造体に入力する必要がなくなる。
- 構造体のフィールドの解析とコード分岐が不要になり、関数のパフォーマンスが最大限に向上する。
- 既存のアプリケーションまたはアーキテクチャに簡単に統合できる、簡単でわかりやすい低レベルの関数の集合体が提供される。

インテル IPP は、異なるプラットフォームにおけるアプリケーションの開発に最適である。例えば、IA-32 プラットフォーム向けに開発した関数を、Itanium® ベースのプラットフォームとインテル® StrongARM* テクノロジまたは Intel XScale® テクノロジを搭載したシステムに簡単に移植できる。また、各インテル IPP 関数は、ANSI C で記述されたリファレンス・コードを持つ。このリファレンス・コードにより、使用されるアルゴリズムが明確に表現され、異なるオペレーティング・システム間の互換性が実現される。

関数の命名

IPP 関数の命名規則は、処理されるすべての領域で共通である。信号処理関数は関数名に ipps プリフィックスが付き、画像および動画処理関数は ippi プリフィックスが付き、スモール・マトリックス操作に固有の関数は ippm プリフィックスが付く。

IPP 関数の命名規則は、処理されるすべての領域で共通である。信号処理関数は関数名に ipps プリフィックスが付き、画像および動画処理関数は ippi プリフィックスが付き、スモール・マトリックス操作に固有の関数は ippm プリフィックスが付く。

インテル IPP の関数名の形式は、一般的に次の形式である。

```
ipp<data-domain><name>_<datatype>[_<descriptor>](<arguments>);
```

この形式の各要素について、以下の各項目で説明する。

データドメイン

data-domain は、関数が所属する機能のサブセットを表す 1 つの文字である。
インテル IPP の現在のバージョンは、以下のデータドメインをサポートしている。

s	信号 （予想されるデータ・タイプは 1 次元信号）
i	画像と動画 （予想されるデータ・タイプは 2 次元画像）
m	行列 （予想されるデータ・タイプは行列）

例えば、関数名が ippi で始まる場合、その関数は画像処理または動画処理に使用される関数である。

名前

name は、関数が実際に実行するコア動作を表す省略形である。例えば、Add、Sqrt などの後に、（場合によっては）関数固有の修飾子が付加される。

<*name*> = <*operation*>[_*modifier*]

修飾子が存在する場合、その修飾子は、特定の関数の多少異なるバージョンを示す。

データ・タイプ

datatype フィールドは、次の形式で関数によって使用されるデータ・タイプを示す。

<bit depth><bit interpretation> ,

ここで

bit depth = <1|8|16|32|64>

および

bit interpretation = <*u*|*s*|*f*>[*c*]

ここで、*u* は「符号なし整数」、*s* は「符号付き整数」、*f* は「浮動小数点」、*c* は「複素数」を示す。

インテル IPP の画像および動画処理関数は、以下のデータ・タイプをサポートしている。

- 8u　8 ビット、符号なしデータ
- 8s　8 ビット、符号付きデータ
- 16u　16 ビット、符号なしデータ
- 16s　16 ビット、符号付きデータ
- 16sc　16 ビット、short 型複素数データ
- 32u　32 ビット、符号なしデータ
- 32s　32 ビット、符号付きデータ
- 32sc　32 ビット、int 型複素数データ
- 32f　32 ビット、単精度実数浮動小数点データ

- 32fc　32 ビット、単精度複素数浮動小数点データ
- 64s　64 ビット、クワッドワード符号付きデータ
- 64f　64 ビット、倍精度実数浮動小数点データ

インテル IPP 内では、複素数データの形式は、次のように定義された構造体によって表現される。

```
typedef struct {
    Ipp16s re;
    Ipp16s im;
} Ipp16sc;
typedef struct {
    Ipp32s re;
    Ipp32s im;
} Ipp32sc;
typedef struct {
    Ipp32f re;
    Ipp32f im;
} Ipp32fc;
```

re,im は、それぞれ実数部と虚数部を示す。

複素数データ形式は、一部の画像処理用算術演算関数によって使用される。32fc 形式は、一部のフーリエ変換関数の入出力データの格納にも使用される。64 ビット形式（64s および 64f）は、一部の画像統計関数によって計算されたデータの格納に使用される。

1 つのデータ・タイプを操作する関数の場合、*datatype* フィールドには、上記の値のうち 1 つが入る。

関数によって操作されるソース・イメージとデスティネーション・イメージのデータ・タイプが異なる場合は、各データ・タイプの識別子が、次のようにソースからデスティネーションの順に関数名の中に示される。

<datatype> = <src1Depth>[src2Depth][dstDepth]

例えば、ソース・イメージの 8 ビット符号なしデータをデスティネーション・イメージの 32 ビット浮動小数点データに変換する関数の場合、*datatype* フィールドの値は、8u32f になる。

注： 関数の引数リスト内では、Ipp プリフィックスがデータ・タイプの中に表示される。例えば、8 ビット符号なしデータは、Ipp8s 型として示される。これらのインテル IPP 固有のデータ・タイプは、それぞれライブラリ・ヘッダ・ファイルの中で定義される。

記述子

descriptor フィールドでは、操作に関連付けられるデータがさらに詳しく記述される。このフィールドは、暗黙のパラメータを含むことも、必要な追加パラメータを示すこともある。

関数のコード分岐の数を最小限に抑え、不要な実行オーバーヘッドを減らすために、ほとんどの一般関数は個々のプリミティブ関数に分解される。関数のパラメータの一部は、記述子としてプリミティブ関数名の中に入る。

ただし、関数を置き換えたものの数が多すぎる場合、一部の関数は、内部操作を指定する引数を使用できる（例えば、ippiThreshold）。

画像および動画処理関数には、以下の記述子が使用される。

- A　データにアルファ・チャネル（常に最後のチャネル）が含まれる（C4 で処理されない）。
- Cn　データは n 個の離散インターリーブ・チャネルで構成される（1、2、3、4）。
- C　処理対象チャネル（COI）が操作に使用される。
- D2　画像は 2 次元画像である。
- I　操作はインプレース操作である。
- M　ソース・イメージとデスティネーション・イメージにマスク ROI を使用する。
- Pn　データは、n 個の離散プレーン（非インターリーブ）チャネルと各プレーンへのポインタで構成される。
- R　処理対象領域（ROI）を使用する。
- Sfs　飽和処理と固定スケーリング・モードを使用する。

関数名内の記述子の省略形は、常にアルファベット順に示される。

関数の中には、上記のすべての省略形を持たないものもある。例えば、コピー操作の関数には、インプレース・モードの記述子は不要である。

また、一部の操作では、いくつかのデータ記述子が暗黙的に前提とされる。画像処理関数は、デフォルトでは 2 次元画像を操作し、結果をスケーリングせずに飽和処理する。この場合、暗黙の省略形 D2 および Sns（飽和処理、スケーリングなし）は、関数名に含まれない。

引数

関数内の引数は、すべてのソースオペランド、すべてのデスティネーション・オペランド、それ以外のすべての操作固有の引数の順に並べられる。

ソースの引数は、"*Src*" または "*SrcN*" （入力画像が 2 つ以上ある場合）の名前である。デスティネーションの引数は、"*Dst*" の名前である。インプレース操作では、入力 / 出力引数に "*SrcDst*" の名前が含まれる。ポインタとして定義されるすべての引数は、小文字の *p* で始まる。例えば、*pSrc*、*pMean*、*pSpec* などである。

インテル® IPP の関数プロトタイプ

インテル IPP の関数名は、*name* フィールドの後に、*datatype* フィールドと *descriptor* フィールドが置かれる（本章の「関数の命名」を参照）。ほとんどのインテル IPP 画像処理関数は、多くの変種を持つ。それぞれの変種は、操作に関連付けられるデータ・タイプと追加パラメータが異なる。

関数の各変種は、独自のプロトタイプを持つ。プロトタイプは、関数の定義と、アプリケーション・プログラムからの関数の呼び出しに使用される。特定の関数に多くの変種がある場合、それらのプロトタイプは非常によく似ている。

本書の関数の説明の節では、よく似たプロトタイプをすべて示さなくてすむように、このようなプロトタイプのテンプレートを示し、続いて各関数に使用可能なデータ・タイプと記述子の表を示す。わかりやすいように、その場合、関数名のデータ・タイプ・フィールドと記述子フィールドは、次のように *mod* として示される。

<mod> = <datatype>_<descriptor>

例えば、非インプレース操作を実行する画像膨張関数のプロトタイプのテンプレートは、次のようになる。

```
IppStatus ippiDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R
8u_C3R
8u_AC4R
```

この表記は、ippiDilate 関数には非インプレース操作用の 3 つの変種があり、各変種は（Ipp8u 型の）8 ビット符号なしデータを処理するが、処理される画像内のチャネル数が異なる意味である。これらの変種のプロトタイプは次のようになる。

```
IppStatus ippiDilate_8u_C1R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDilate_8u_C3R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDilate_8u_AC4R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

したがって、直接示されていない特定の関数の変種の完全な名前と引数リストは、次の方法で得られる。

- 関数の動作モード（本書では**事例 1、事例 2**... として示される）を選び、サポートされるデータ・タイプと記述子を表で確認する。
- 選択したデータ・タイプと記述子をアンダースコアで区切って連結し、関数名の *mod* フィールドを設定する。
- それぞれのテンプレートを使用して、引数リスト内のすべての *datatype* フィールドを、選択したデータ・タイプで置き換える。引数リスト内の *datatype* の前に、Ipp プリフィックスが付いていることに注意する（詳細については、本章の「データ・タイプ」を参照）。

例：
3 チャネルのデスティネーション・イメージの各チャネルを符号付き 16 ビット値に設定する ippiSet 関数の変種のプロトタイプを得るには、**「事例 2：各カラー・チャネルを指定した値に設定する」**を選択して、*datatype*=16s, *descriptor*=C3R を使用する。
mod フィールドを 16s_C3R で置き換えると、次のような必要なプロトタイプが得られる。

```
ippiSet_16s_C3R(const Ipp16s value[3], Ipp16s* pDst, int dstStep,
    IppiSize roiSize);
```

整数結果のスケーリング

整数データを操作する画像処理関数の一部は、内部で計算された出力結果を、関数のパラメータの 1 つで指定される整数 *scaleFactor* でスケーリングする。これらの関数の名前には、Sfs 記述子が付く。

係数には、負の値、正の値、または 0 を使用できる。内部計算は、一般的に入力画像と出力画像に使用されるデータ・タイプより高い精度で実行されるため、スケーリングが適用される。

注： スケーリングが使用される場合でも、整数演算の結果は、常にデスティネーションのデータ・タイプの範囲に合わせて飽和処理される。

整数結果のスケーリングは、関数のリターンの前に、出力ピクセル値に $2^{-scaleFactor}$ を掛けて行われる。これによって、出力データの範囲または精度のいずれかを維持できる。通常は、正の係数を使用したスケーリングは、シフト操作によって実行される。得られた結果は、最も近い整数に合わせて切り捨てられる。
例えば、入力値 200 に対する 2 乗演算 ippiSqr の結果得られる Ipp16s の整数は、40000 ではなく 32767 になる。つまり、結果は飽和処理されるため、正確な値は復元できない。
係数 *scaleFactor* = 1 を使用して出力値をスケーリングすると、20000 の結果が得られる。この結果は飽和処理されないため、20000*2 として正確な値を復元できる。したがって、出力データの範囲が維持される。

次の例は、スケーリングによって精度を部分的に維持する方法を示している。
入力値 2 に対する整数平方根演算 ippiSqrt （スケーリングなし）の結果は、1.414 ではなく 1 になる。内部で計算された出力値を、係数 *scaleFactor* = -3 でスケーリングすると、11 の結果が得られる。これによって、より正確な値を $11*2^{-3}=1.375$ として復元できる。

エラー・レポート

インテル IPP 関数は、実行された操作のステータス・コードを返し、呼び出し元プログラムに対してエラーと警告を報告する。したがって、エラーに関連する処置の実行やエラーからの回復は、アプリケーション側の役割となる。エラー・ステータスの最後の値は格納されない。関数のリターン時にこの値をチェックするかどうかは、ユーザが決める必要がある。ステータス・コードは、IppStatus 型のグローバルな整数定数である。

画像および動画処理に関するステータス・コードと、それに対応するインテル IPP が報告するメッセージを、表 2-1 に示す。

表 2-1　ステータス・コードとメッセージ

ステータス・コード	メッセージ
ippStsNotSupportedModeErr	The requested mode is currently not supported 要求されたモードは、現在サポートされていない。
ippStsNotEvenStepErr	Step value is not pixel multiple ステップ値がピクセルの倍数になっていない。
ippStsJPEG2KBadPassNumber	Pass number exceeds allowed limits [0,nOfPasses-1]" パス番号が許容範囲 [0,nOfPasses-1]" を超えている。
ippStsJPEG2KDamagedCodeBlock	Codeblock for decoding is damaged デコード用のコードブックが一部壊れている。
ippStsH263CBPYCodeErr	Illegal Huffman code during CBPY stream processing CBPY ストリームの処理中に無効なハフマン・コードが検出された。
ippStsH263MCBPCInterCodeErr	Illegal Huffman code while MCBPC Inter stream processing MCBPC Inter ストリームの処理中に無効なハフマン・コードが検出された。
ippStsH263MCBPCIntraCodeErr	Illegal Huffman code while MCBPC Intra stream processing MCBPC Intra ストリームの処理中に無効なハフマン・コードが検出された。
ippStsHistoNofLevelsErr	Number of levels for histogram is less than 2 ヒストグラムのレベルの数が 2 より小さい。
ippStsLUTNofLevelsErr	Number of levels for LUT is less than 2 LUT のレベルの数が 2 より小さい。
ippStsMP4BitOffsetErr	Incorrect bit offset value ビット・オフセット値が不適当である。
ippStsMP4QPErr	Incorrect quantization parameter 量子化パラメータが不適当である。
ippStsMP4BlockIdxErr	Incorrect block index ブロック・インデックスが不適当である。
ippStsMP4BlockTypeErr	Incorrect block type ブロック・タイプが不適当である。
ippStsMP4MVCodeErrIllgal	Illegal Huffman code while MV stream processing MV ストリームの処理中に無効なハフマン・コードが検出された。
ippStsMP4VLCCodeErr	Illegal Huffman code while VLC stream processing VLC ストリームの処理中に無効なハフマン・コードが検出された。
ippStsMP4DCCodeErr	Illegal code while DC stream processing DC ストリームの処理中に無効なコードが検出された。
ippStsMP4FcodeErr	Incorrect fcode value fcode の値が不適当である。

続く

表 2-1　ステータス・コードとメッセージ（続き）

ステータス・コード	メッセージ
ippStsMP4AlignErr	Incorrect buffer alignment バッファのアライメントが不適当である。
ippStsMP4TempDiffErr	Incorrect temporal difference 時間的な差が不適当である。
ippStsMP4BlockSizeErr	Incorrect size of block or macroblock ブロックまたはマクロブロックのサイズが不適当である。
ippStsMP4ZeroBABErr	All BAB values are zero すべての BAB 値がゼロになっている。
ippStsMP4PredDirErr	Incorrect prediction direction 予測の方向が不適当である。
ippStsMP4BitsPerPixelErr	Incorrect number of bits per pixel ピクセル当たりのビット数が不適当である。
ippStsMP4VideoCompModeErr	Incorrect video component mode ビデオ・コンポーネント・モードが不適当である。
ippStsMP4LinearModeErr	Incorrect DC linear mode DC リニア・モードが不適当である。
ippStsH263PredModeErr	Prediction Mode value error Prediction Mode の値のエラー。
ippStsH263BlockStepErr	The step value is less than 8 ステップ値が 8 より小さい。
ippStsH263MBStepErr	The step value is less than 16 ステップ値が 16 より小さい。
ippStsH263FrameWidthErr	The frame width is less than 8 フレームの幅が 8 より小さい。
ippStsH263FrameHeightErr	The frame height is less than or equal to zero フレームの高さがゼロより小さいかまたはゼロである。
ippStsH263ExpandPelsErr	The expand pixels number is less than 8 拡張ピクセルの数が 8 より小さい。
ippStsH263PlaneStepErr	Step value is less than the plane width ステップ値がプレーンの幅より小さい。
ippStsH263QuantErr	Quantizer value is less than or equal to zero, or greater than 31 量子化器の値がゼロまたはゼロより小さいか、31 より大きい。
ippStsH263MVCodeErr	Illegal Huffman code while MV stream processing MV ストリームの処理中に無効なハフマン・コードが検出された。
ippStsH263VLCCodeErr	Illegal Huffman code while VLC stream processing VLC ストリームの処理中に無効なハフマン・コードが検出された。
ippStsH263DCCodeErr	Illegal code while DC stream processing DC ストリームの処理中に無効なコードが検出された。

続く

表 2-1　ステータス・コードとメッセージ（続き）

ステータス・コード	メッセージ
ippStsH263ZigzagLenErr	Zigzag compact length is more than 64 ジグザグ・コンパクト長が 64 より大きい。
ippStsJPEGHuffTableErr	JPEG Huffman table is destroyed JPEG ハフマン・テーブルが破壊されている。
ippStsJPEGDCTRangeErr	JPEG DCT coefficient is outs of the range JPEG DCT 係数が範囲外である。
ippStsJPEGOutOfBufErr	Attempt to access out of the buffer バッファの外側にアクセスしようとした。
ippStsChannelOrderErr	Wrong order of the destination channels デスティネーション・チャネルの順序が誤っている。
ippStsZeroMaskValueErr	All values of the mask are zero マスクのすべての値が 0 になっている。
ippStsRangeErr	Bad values of bounds: the lower bound is greater than the upper bound 上下限の値が無効である。下限が上限より大きい。
ippStsQuadErr	The quadrangle degenerates into triangle, line or point 四角形が、三角形、直線、または点に変換された。
ippStsRectErr	Size of the rectangular region is less than or equal to 1 矩形領域のサイズが、1 より小さいかまたは 1 である。
ippStsCoeffErr	Unallowable values of the transformation coefficients 変換係数の値が無効である。
ippStsNoiseValErr	Bad value of noise amplitude for dithering ディザリング用のノイズ振幅の値が無効である。
ippStsDitherLevelsErr	Number of dithering levels is out of range ディザリング・レベルの数が範囲外である。
ippStsNumChannelsErr	Bad or unsupported number of channels チャネル数が無効またはサポートされていない。
ippStsDataTypeErr	Bad or unsupported data type データ、タイプが無効またはサポートされていない。
ippStsCOIErr	COI is out of range COI が範囲外である。
ippStsOutOfRangeErr	Argument is out of range or point is outside the image 引数が範囲外であるか、点が画像の外側にある。
ippStsDivisorErr	Divisor is equal to zero, function is aborted 除数が 0 だったため、関数は中止された。
ippStsAlphaTypeErr	Illegal type of image compositing operation 画像合成操作のタイプが無効である。
ippStsGammaRangeErr	Gamma range bound is less than or equal to zero ガンマ範囲の限界が、0 より小さいかまたは 0 である。
ippStsGrayCoefSumErr	Sum of the conversion coefficients must be less than or equal to 1 変換係数の合計は、1 より小さいかまたは 1 でなければならない。

続く

表 2-1　ステータス・コードとメッセージ（続き）

ステータス・コード	メッセージ
ippStsChannelErr	Illegal channel number チャネル番号が無効である。
ippStsJaehneErr	Magnitude value is negative 絶対値が負になっている。
ippStsStepErr	Step value is less than or equal to zero ステップ値が、0 より小さいかまたは 0 である。
ippStsStrideErr	Stride value is less than the row length ストライド値が行の長さより小さい。
ippStsEpsValErr	Negative epsilon value error 負の ε 値エラー。
ippStsScaleRangeErr	Scale bounds are out of the range スケールの上下限が範囲外である。
ippStsThresholdErr	Invalid threshold bounds しきい値の上下限が無効である。
ippStsWtOffsetErr	Invalid offset value of wavelet filter ウェーブレット・フィルタのオフセット値が無効である。
ippStsAnchorErr	Anchor point is outside the mask アンカ・ポイントがマスクの外側にある。
ippStsMaskSizeErr	Invalid mask size マスクのサイズが無効である。
ippStsShiftErr	Shift value is less than zero シフト値が 0 より小さい。
ippStsSampleFactorErr	Sampling factor is less than or equal to zero サンプリング係数が、0 より小さいかまたは 0 である。
ippStsResizeFactorErr	Resize factor(s) is less or equal to zero サイズ変更係数が、0 より小さいかまたは 0 である。
ippStsDivByZeroErr	An attempt to divide by zero 0 で割ろうとした。
ippStsInterpolationErr	Invalid interpolation mode 補間モードが無効である。
ippStsMirrorFlipErr	Invalid flip mode フリップ・モードが無効である。
ippStsMoment00ZeroErr	Moment value M(0,0) is too small to continue calculations モーメント値 M （0,0）が小さすぎて計算を続行できない。
ippStsThreshNegLevelErr	Negative value of the level in the threshold operation しきい値演算のレベルの値が負になっている。
ippStsContextMatchErr	Context parameter doesn't match the operation コンテキスト・パラメータと操作が一致しない。
ippStsFftFlagErr	Invalid value of the FFT flag parameter FFT フラグ・パラメータの値が無効である。
ippStsFftOrderErr	Invalid value of the FFT order parameter FFT オーダ・パラメータの値が無効である。

続く

表 2-1　ステータス・コードとメッセージ（続き）

ステータス・コード	メッセージ
ippStsMemAllocErr	Not enough memory for the operation 操作に使用するメモリが足りない。
ippStsNullPtrErr	Null pointer error NULL ポインタ・エラー
ippStsSizeErr	Wrong value of data size データ・サイズの値が無効である。
ippStsNoErr	No error, it's OK エラーなし。
ippStsNoOperation	No operation has been executed 操作は何も実行されていない。
ippStsMisalignedBuf	Misaligned pointer in operation in which it must be aligned ポインタのアライメントが合っていなければならない操作で、ポインタのアライメントが合っていない。
ippStsSqrtNegArg	Negative value(s) of the argument in the function Sqrt 関数 Sqrt の引数の値が負になっている。
ippStsInvZero	INF result. Zero value was met by InvThresh with zero level INF 結果。0 レベルの InvThresh によって 0 の値が検出された。
ippStsEvenMedianMaskSize	Even size of the Median Filter mask was replaced by the odd number メディアン・フィルタ・マスクの偶数のサイズが、奇数で置き換えられた。
ippStsDivByZero	Zero value(s) of the divisor in the function Div 関数 Div の除数の値が 0 になっている。
ippStsLnZeroArg	Zero value(s) of the argument in the function Ln 関数 Ln の引数の値が 0 になっている。
ippStsLnNegArg	Negative value(s) of the argument in the function Ln 関数 Ln の引数の値が負になっている。
ippStsNanArg	Not a Number argument value warning NaN 引数値の警告
ippStsJPEGMarker	JPEG marker was met in the bitstream JPEG マーカをビットストリーム内で検出した。

Err で終わるステータス・コード（ippStsNoErr ステータスを除く）は、エラーを示す。これらのコードの整数値は負の値である。エラーが発生すると、関数の実行は中断される。
ステータス・コード ippStsNoErr はエラーがないことを示す。
これ以外のステータス・コードは、すべて警告を示す。警告状態が検出されると、関数は最後まで実行され、対応する警告ステータス・コードが返される。

例えば、関数 ippiDiv で、正の値を 0 で割ろうとした場合、関数の実行は中止されない。この演算の結果は、ソース・オペランドのデータ・タイプで表現可能な最大値に設定され、出力ステータス ippStsDivByZero によってユーザに警告が出される。詳細は、付録 A「特殊な事例の処理」を参照のこと。

構造体と列挙子

本節では、画像および動画処理用インテグレーテッド・パフォーマンス・プリミティブで使用される構造体と列挙子について説明する。

IppStatus 定数は、インテル IPP 関数によって返されたステータス・コードの値を列挙し、操作にエラーがなかったかどうかを示す。

画像および動画処理関数の有効なステータス・コードとそれに対応するエラー・メッセージの詳細については、本章の「エラー・レポート」の節を参照のこと。

点の幾何学的な位置を格納する構造体 IppiPoint は、次のように定義される。

```
typedef struct {
    int x;
    int y;
} IppiPoint;
```

x,y は、点の座標を示す。

矩形のサイズを格納する構造体 IppiSize は、次のように定義される。

```
typedef struct {
    int width;
    int height;
} IppiSize;
```

width と height は、それぞれ矩形の x 方向と y 方向の寸法を示す。

矩形の幾何学的な位置とサイズを格納する構造体 IppiRect は、次のように定義される。

```
typedef struct {
    int x;
    int y;
    int width;
    int height;
} IppiRect;
```

x,y は、x 方向に width、y 方向に height の寸法を持つ矩形の左上隅の座標を示す。

コンピュータ・ビジョン関数に使用される ippiConnectedComp 構造体は、接続されるコンポーネントを次のように定義する。

```
typedef struct {
    Ipp32s area;
    Ipp32f value;
    IppiRect rect;
} IppiConnectedComp;
```

area、value、rect は、接続されるコンポーネントのパラメータを示す。

IppiMaskSize 列挙子は、モルフォロジー関数とフィルタリング関数の隣接領域を次のように定義する。

```
typedef enum {
    ippMskSize1x3 = 13,
    ippMskSize1x5 = 15,
    ippMskSize3x1 = 31,
    ippMskSize3x3 = 33,
    ippMskSize5x1 = 51,
    ippMskSize5x5 = 55
} IppiMaskSize;
```

IppCmpOp 列挙子は、画像比較関数で使用される比較演算のタイプを次のように定義する。

```
typedef enum {
    ippCmpLess,
    ippCmpLessEq,
    ippCmpEq,
    ippCmpGreaterEq,
    ippCmpGreater
} IppCmpOp;
```

IppRoundMode 列挙子は、変換関数で使用される丸めモードを次のように定義する。

```
typedef enum {
    ippRndZero,
    ippRndNear
} IppRoundMode;
```

IppHintAlgorithm 列挙子は、一部の画像変換関数および統計係数で使用されるコードのタイプ（高速だが精度が低いコード、または精度は高いが低速なコード）を定義する。この列挙子の使い方の詳細は、表 10-2 または表 11-2 を参照のこと。

```
typedef enum {
    ippAlgHintNone,
    ippAlgHintFast,
    ippAlgHintAccurate
} IppHintAlgorithm;
```

ジオメトリ変換関数に使用される補間のタイプは、次のように定義される。

```
enum {
    IPPI_INTER_NN     = 1,
    IPPI_INTER_LINEAR = 2,
    IPPI_INTER_CUBIC  = 4,
    IPPI_INTER_SUPER  = 8,
    IPPI_SMOOTH_EDGE  = (1 << 31)
};
```

IppiAlphaType 列挙子は、アルファ合成関数で使用される合成操作のタイプを次のように定義する。

```
typedef enum {
    ippAlphaOver,
    ippAlphaIn,
    ippAlphaOut,
    ippAlphaATop,
    ippAlphaXor,
    ippAlphaPlus,
    ippAlphaOverPremul,
    ippAlphaInPremul,
    ippAlphaOutPremul,
    ippAlphaATopPremul,
    ippAlphaXorPremul,
    ippAlphaPlusPremul
} IppiAlphaType;
```

IppiDitherType 列挙子は、ippiReduceBits 関数で使用されるディザリングのタイプを次のように定義する。

```
typedef enum {
    ippDitherNone,
    ippDitherFS,
    ippDitherJJN,
    ippDitherStucki,
    ippDitherBayer
} IppiDitherType;
```

IppiShape 列挙子は、モルフォロジー関数で使用される構造要素の形状を次のように定義する。

```
typedef enum {
    ippiShapeRect = 0,
    ippiShapeCross = 1,
    ippiShapeEllipse = 2,
    ippiShapeCustom = 100
} IppiShape;
```

IppiAxis 列挙子は、ippiMirror 関数のフリップ軸または ippiImageRamp 関数の画像の輝度の傾斜の方向を次のように定義する。

```
typedef enum {
    ippAxsHorizontal,
    ippAxsVertical,
    ippAxsBoth
} IppiAxis;
```

IppiWTSubband 列挙子は、JPEG2000 コード化関数で使用される適切なウェーブレット変換サブバンドを次のように定義する。

```
typedef enum {
    ippWTSubbandLxLy,
    ippWTSubbandLxHy,
    ippWTSubbandHxLy,
    ippWTSubbandHxHy
} IppiWTSubband;
```

IppiMQTermination 列挙子は、JPEG2000 コード化関数の実行中の MQ コーダの終了方法を次のように定義する。

```
typedef enum {
    ippMQTermSimple,
    ippMQTermNearOptimal,
    ippMQTermPredictable
} IppiMQTermination;
```

IppiMQRateAppr 列挙子は、JPEG2000 コード化関数が実行するレートと歪みの推定手順のパディング近似モデルを次のように定義する。

```
typedef enum {
    ippMQRateApprGood
} IppiMQRateAppr;
```

JPEG2000 コード化関数に使用されるコード・フラグは、次のように定義される。

```
enum
{
  IPP_JPEG2K_VERTICALLY_CAUSAL_CONTEXT,
  IPP_JPEG2K_SELECTIVE_MQ_BYPASS,
  IPP_JPEG2K_TERMINATE_ON_EVERY_PASS,
  IPP_JPEG2K_RESETCTX_ON_EVERY_PASS,
  IPP_JPEG2K_USE_SEGMENTATION_SYMBOLS,
  IPP_JPEG2K_LOSSLESS_MODE,

  IPP_JPEG2K_DEC_CONCEAL_ERRORS,
  IPP_JPEG2K_DEC_DO_NOT_CLEAR_CB,
  IPP_JPEG2K_DEC_DO_NOT_RESET_LOW_BITS,
  IPP_JPEG2K_DEC_DO_NOT_CLEAR_SFBUFFER
  IPP_JPEG2K_DEC_CHECK_PRED_TERM
};
```

ライブラリ内の一部の構造体は、関数固有の （コンテキスト）情報の格納に使用される。例えば、IppiFFTSpec 構造体は、高速フーリエ変換の計算に必要な回転係数とビット反転指数を格納する。もう 1 つの例として、DFT 関数と DCT 関数によって使用される構造体がある。
これらのコンテキスト関連構造体は、パブリック・ヘッダ内では定義されない。また、ユーザはこれらの構造体のフィールドにアクセスできない。これは、関数コンテキストの解釈がプロセッサによって異なるためである。したがって、開発者は、コンテキスト関連関数を使用することはできるが、関数コンテキストを自動変数としては作成できない。

画像のデータ・タイプと範囲

インテル IPP 画像処理関数は、各ピクセルがチャネルの強さによって表現される、絶対カラー・イメージだけをサポートする。画像データの格納形式は、ピクセル指向またはプレーン指向である。ピクセル順序の画像では、各ピクセルのすべてのチャネル値がまとめられ、連続的に格納される。例えば、RGB 画像の場合、RGBRGBRGB のようになる。ピクセル順序の画像のチャネル数は、1、2、3、または 4 である。

プレーン順序の画像では、各チャネルのすべての画像データが連続的に格納され、その後に次のチャネルが続く。例えば、RRR...GGG...BBB のようになる。

プレーン画像を操作する関数の名前には、Pn 記述子が含まれている。この場合は、n 個のポインタ（各プレーンに 1 つずつ）を指定できる。

画像のデータ・タイプは、チャネル当たりのビット数単位のピクセル分解能（すなわち、色分解能）によって決まる。各チャネルの色分解能は、8、16、または 32 ビットであり、いずれかの値が関数名に含まれる （詳細については、本章の関数の命名を参照）。データは、符号付き （s）、符号なし（u）、または浮動小数点実数（f）形式である。一部の算術演算関数と FFT/DFT 関数では、複素数形式（sc または fc）のデータを使用できる。この場合、各チャネル値は、実数部と虚数部の 2 つの値で表現される。画像内のすべてのチャネルは、同じデータ・タイプでなければならない。

例えば、絶対カラー24 ビット RGB 画像では、ピクセル当たり連続する 3 バイト（24 ビット）で、ピクセル・モードの 3 チャネルの強さを表す。このデータ・タイプは、関数名の 8u_C3 記述子で示される。8u は各チャネルの 8 ビット符号なしデータを表し、C3 は 3 つのチャネルを表す。

画像データ内にアルファ（透過度）チャネルがある場合は、画像は必ず 4 つのチャネルを持ち、アルファ・チャネルが最後のチャネルになる。このデータ・タイプは、AC4 記述子で示される。アルファ・チャネルの有無によって、関数の動作が変わることがある。インテル IPP は、このような関数について、アルファ・チャネルありのバージョンとアルファ・チャネルなしのバージョンを用意している。アルファ・チャネルが指定されている場合、通常はそのチャネルに対して操作は行われない。

各データ・タイプで表現可能な値の範囲は、下限と上限で指定される。次の表に、各データ範囲と、範囲の上下限を示すインテル IPP の不変識別子を示す。

表 2-2　画像のデータ・タイプと範囲

データ・タイプ	識別子	上限値	識別子	下限値
8s	IPP_MIN_8S	-128	IPP_MAX_8S	127
8u		0	IPP_MAX_8U	255
16s	IPP_MIN_16S	-32768	IPP_MAX_16S	32767
16u		0	IPP_MAX_16U	65535
32s	IPP_MIN_32S	-2^{31}	IPP_MAX_32S	$2^{31}-1$
32u		0	IPP_MAX_32U	$2^{32}-1$
32f †	IPP_MINABS_32F	$1.175494351e^{-38}$	IPP_MAXABS_32F	$3.402823466e^{38}$

† 絶対値の範囲

主な動作モデル

ほとんどのインテル IPP 画像処理関数は、処理される画像のすべてのチャネル（アルファ・チャネルを除く）に対して、互いに独立した同じ操作を実行する。これは、各チャネルに対して同じ操作が実行され、計算された結果は他のチャネルの値に依存しないという意味である。ただし、ippiFilterMedianColor 関数とカラー変換関数は例外である。これらの関数は、3 つのチャネルをまとめて処理する。

インテル IPP 画像処理関数は、1 つのピクセルを操作して結果を計算する関数（点操作関数、例えば ippiAdd）と、ピクセルのグループ（隣接領域とも呼ばれる）を操作する関数（例えば ippiFilterBox）の 2 つの主な動作モデルに分けられる。

隣接操作

隣接操作の結果は、指定された入力ピクセルの近くに位置する、特定のピクセル・グループの値に基づいている。一連の隣接ピクセルは、通常は矩形マスク（またはカーネル）とアンカ・セルによって定義される。アンカ・セルは、入力ピクセルの位置を基準として矩形マスクのアライメントを指定する。

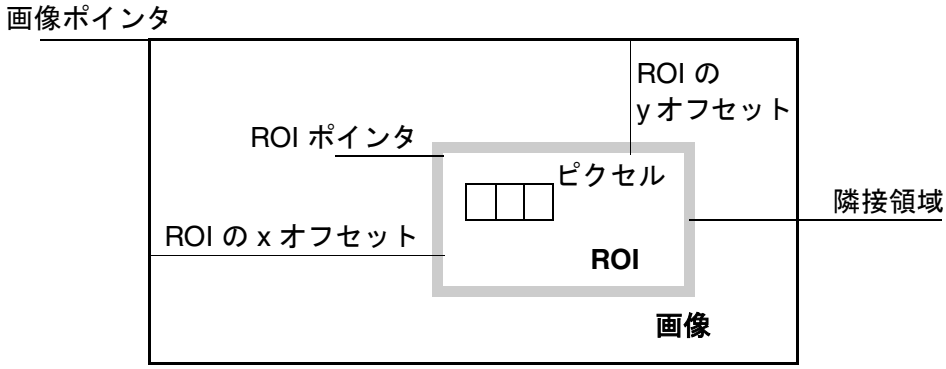
隣接領域を処理するインテル IPP 関数は、画像のすべての参照される点が利用可能である前提に基づいて動作する。隣接操作モードを使用する場合は、処理されるすべての隣接ピクセルが実際に画像内に存在するように、関数に渡される ROI パラメータの値が正しく設定されていることを、アプリケーション側で確認する必要がある。

インテル® IPP の処理対象領域

ほとんどインテルの IPP 画像処理関数は、画像全体だけでなく、画像内の領域も操作できる。画像の処理対象領域（ROI）は、画像の一部または画像全体に相当する矩形の領域である。

ROI をサポートするインテル IPP 関数の名前には、R 記述子が含まれている。図 2-1 に示すように、画像の ROI は、サイズと画像の原点からのオフセットによって定義される。画像の原点は左上隅に位置し、x 値は左から右に向かって増加し、y 値は上から下に向かって増加すると見なされる。

図 2-1　画像、ROI、オフセット



ソース・イメージとデスティネーション・イメージの両方に、処理対象領域を指定できる。このような場合、ソース ROI とデスティネーション ROI のサイズは同じと見なされるが、オフセットは異なることがある。ソース ROI のデータに対して画像処理が実行され、結果がデスティネーション ROI に書き込まれる。

関数呼び出しシーケンス内では、ROI は次の方法で指定される。

- IppiSize 型の *roiSize* 引数
- ソース ROI バッファとデスティネーション ROI バッファの始点を指す *pSrc* および *pDst* ポインタ
- *srcStep* および *dstStep* 引数。これらの引数の値は、それぞれソース・イメージ内およびデスティネーション・イメージ内の連続的な直線の始点間の距離（バイト単位）に等しい。

したがって、引数 *srcStep* および *dstStep* は、画像の ROI 内で新しい直線の処理を開始する画像バッファ内のステップをバイト単位で設定する。

次のコード例は、関数呼び出し内での *dstStep* パラメータの使用方法を示している。

例 2-1　バッファ・ステップ・パラメータの使用

```
IppStatus alignedLine( void ) {
   Ipp8u x[8*3] = {0};
   IppiSize imgSize = {5,3};
   /// The image is of size 5x3. Width 8 has been
   /// chosen by the user to align every line of the image
   return ippiSet_8u_C1R( 7, x, 8, imgSize);
}
```

得られる画像 *x* には、次のデータが含まれる。

```
07 07 07 07 07 00 00 00
07 07 07 07 07 00 00 00
07 07 07 07 07 00 00 00
```

ROI が存在する場合は、次のようになる。

- ソース・イメージとデスティネーション・イメージが異なるサイズでもかまわない。
- 直線のサイズを揃えるために、直線の終点にパディングが挿入されることがある。
- アプリケーションは、*pSrc*、*pDst*、*roiSize* 引数を正しく定義しなければならない。

pSrc および *pDst* 引数は、画像データへのシフトされたポインタである。例えば、ピクセル当たり 3 バイトの画像データ（8u_C3R）に対する ROI 操作の場合、*pSrc* はソース ROI バッファの始点を指し、次のように解釈される。

```
pSrc = pSrcImg + 3*(srcImgSize.width * srcRoiOffset.y +
srcRoiOffset.x)
```

ここで

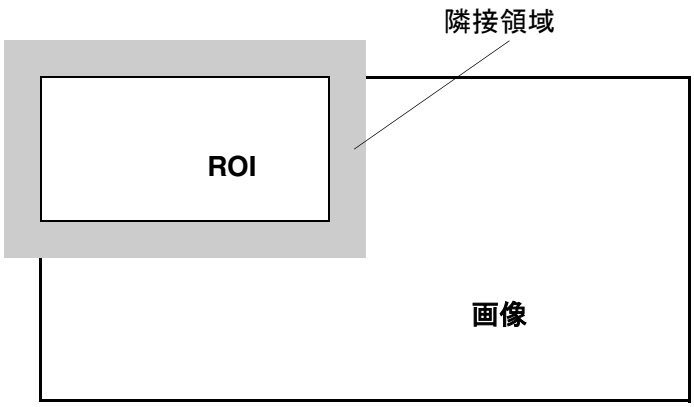
pSrcImg ソース・イメージ・バッファの始点を指す。

srcImgSize IppiSize 型の画像サイズ（ピクセル単位）である。

srcRoiOffset 図 2-1 に示すように、画像の始点を基準とする ROI のオフセットを指定する。

関数が隣接領域を持つ ROI を操作する場合は、*pSrc* および *roiSize* パラメータの値を正しく使用する必要がある。これらの関数は、隣接領域内の点が存在すること、したがって、*pSrc* と *pSrcImg* の値はほぼ常に異なることを前提とする。図 2-2 は、隣接ピクセルがソース・イメージの外側にある場合を示している。

図 2-2 隣接領域を持つ ROI の使用



次のコード例は、ROI を持つ画像を処理する方法を示している。

例 2-2　ROI の処理

```
IppStatus roi( void ) {
   Ipp8u x[8*3] = {0};
   IppiSize roiSize = {3,2};
   IppiPoint roiPoint = {2,1};
   /// place the pointer to the ROI start position
   return ippiSet_8u_C1R( 7,
x+8*roiPoint.y+roiPoint.x, 8, roiSize );
}
```

デスティネーション・イメージ *x* には、次のデータが含まれる。

```
00 00 00 00 00 00 00 00
00 00 07 07 07 00 00 00
00 00 07 07 07 00 00 00
```

サポート関数

3

本章では、次の目的で使用されるインテル® IPP のサポート関数について説明する。

- 現在使用しているインテル IPP ソフトウェアのバージョンに関する情報を返す。
- 返されるステータス・コードの簡単な説明を表示する。
- 他のインテル IPP 画像および動画処理関数の操作に必要なメモリの割り当てと解放を行う。

サポート関数の一覧を表 3-1 に示す。

表 3-1 インテル® IPP サポート関数

関数の基本名	操作
バージョン情報関数	
ippiGetLibVersion	アクティブなライブラリのバージョンに関する情報を返す。
ステータス情報関数	
CoreGetStatusString	ステータス・コードをメッセージに変換する。
メモリ割り当て関数	
ippiMalloc	32 バイト境界にアライメントされたメモリを割り当てる。
ippiFree	関数 `ippiMalloc` によって割り当てられたメモリを解放する。

バージョン情報関数

この関数は、現在使用している画像処理用インテル IPP ソフトウェアのバージョン番号などの情報を返す。

ippiGetLibVersion

現在使用している画像処理用 IPP ソフトウェアのバージョン情報を返す。

```
const IppLibraryVersion* ippiGetLibVersion(void);
```

説明

関数 `ippiGetLibVersion` は、`ippi.h` ファイルの中で宣言される。この関数は現在使用している画像処理用インテル IPP のバージョン情報を格納した静的データ構造体 `IppLibraryVersion` へのポインタを返す。このポインタは静的な変数を指すため、返されたポインタによって参照されるメモリを解放する必要はない。`IppLibraryVersion` 構造体の以下のフィールドが利用可能である。

major	現在のライブラリのメジャー・バージョン番号
minor	現在のライブラリのマイナー・バージョン番号
Name	現在のライブラリの名前
Version	ライブラリのバージョン文字列
BuildDate	このバージョンの実際の構築日

例えば、ライブラリのバージョンが "v1.2 Beta"、ライブラリ名が "ippim6"、構築日が "Jul 20 99" である場合、この構造体のフィールドは次のように設定される。

major = 1, *minor* = 2, *Name* = "ippim6",
Version = "v1.2 Beta", *BuildDate* = "Jul 20 99"

例 3-1 は、関数 ippiGetLibVersion の使用方法を示している。

例 3-1　ippiGetLibVersion 関数の使用

```
void libinfo(void) {
        const IppLibraryVersion* lib = ippiGetLibVersion();
        printf("%s %s %d.%d.%d.%d\n", lib->Name, lib->Version,
            lib->major, lib->minor, lib->majorBuild, lib->build);
}

Output:
    ippia6 v0.0 Alpha 0.0.5.5
```

注： 画像処理領域で使用される各サブライブラリは、アクティブなライブラリのバージョンに関する情報を取得する、ippiGetLibVersion によく似た独自の関数を持っている。
これらの関数は、ippGetLibVersion、ippjGetLibVersion、ippcvGetLibVersion、ippvcGetLibVersion、ippmpGetLibVersion である。これらの関数は、それぞれヘッダ・ファイル ippcore.h、ippj.h、ippcv.h、ippvc.h、ippmp.h で宣言され、ippiGetLibVersion 関数と同じインターフェイスを持つ。

ステータス情報関数

この関数を使用して、現在のインテル IPP ソフトウェアによって返されるステータス・コードの簡単な説明を表示できる。

GetStatusString

ステータス・コードをメッセージに変換する。

```
const char* ippCoreGetStatusString(IppStatus StsCode);
```

引数

StsCode	ステータスのタイプを示すコード（表 2-1 を参照）

説明

関数 ippCoreGetStatusString は、ipps.h ファイルの中で宣言される。この関数は、ステータス・コード *StsCode* に関連付けられた文字列へのポインタを返す。この関数を使用して、ユーザ向けのエラー・メッセージと警告メッセージを生成できる。返されるポインタは、内部の静的バッファへのポインタであるため、解放する必要はない。

注： 関数 ippCoreGetStatusString は、関数 ippGetStatusString に代わるものである。関数 ippGetStatusString は、以前のバージョンとの互換性を保証するために、インテル IPP バージョン 3 でサポートされている。

次のコード例は、関数 ippCoreGetStatusString の使用方法を示している。NULL ポインタを使用してインテル IPP 関数（この場合は ippiSet_8u_C1R,）を呼び出すと、エラー・コード -8 が返される。

ステータス情報関数は、このコードを、対応するメッセージ "Null Pointer Error" に変換する。

例 3-2　ステータス情報関数の使用

```
void statusInfo( void ) {
   IppiSize roi = {0};
   IppStatus st = ippiSet_8u_C1R( 3, 0, 0, roi );
   printf( " %d : %s\n", st, ippCoreGetStatusString( st ));
}
Output:
    -8, Null Pointer Error
```

メモリ割り当て関数

本節では、必要なタイプのデータに対してアライメントされたメモリ・ブロックを割り当てるインテル IPP 関数と、以前に割り当てられたメモリを解放するインテル IPP 関数について説明する。割り当てられるメモリのサイズは、割り当てられる要素の数 len によって指定される。

注： これらの関数によって割り当てられたメモリを解放する関数は、ippsFree ( ) のみである。

ippiMalloc

32 バイト境界にアライメントされたメモリを割り当てる。

```
Ipp<datatype>* ippiMalloc_<mod>(int widthPixels, int heightPixels,
    int* pStepBytes);
```

サポートされる *mod* は、次のとおりである。

8u_C1	16u_C1	16s_C1	32s_C1	32f_C1	32sc_C1	32fc_C1
8u_C2	16u_C2	16s_C2	32s_C2	32f_C2	32sc_C2	32fc_C2
8u_C3	16u_C3	16s_C3	32s_C3	32f_C3	32sc_C3	32fc_C3
8u_C4	16u_C4	16s_C4	32s_C4	32f_C4	32sc_C4	32fc_C4
8u_AC4	16u_AC4	16s_AC4	32s_AC4	32f_AC4	32sc_AC4	32fc_AC4
8u_P3	16u_P3	16s_P3	32s_P3	32f_P3	32sc_P3	32fc_P3

引数

widthPixels	画像の幅 （ピクセル単位）
heightPixels	画像の高さ （ピクセル単位）
pStepBytes	画像内のステップ （バイト単位）へのポインタ

説明

関数 ippiMalloc は、ippi.h ファイルの中で宣言される。この関数は、さまざまなデータ・タイプの要素に対して、32 バイト境界にアライメントされたメモリ・ブロックを割り当てる。
画像の各行は、*pStepBytes* パラメータに従って、ゼロのパディングによってアライメントされる。このパラメータは、関数 ippiMalloc によって計算され、将来の使用のために返される。

関数 ippiMalloc は 1 つの連続したメモリ・ブロックを割り当てるが、プレーン画像を処理する関数は、各プレーンに対する別々のポインタの配列 （IppType* plane[3]）を入力として要求する。以下のコード例は、プレーン画像を処理するインテル IPP 関数で割り当てられたメモリ・ブロックを使用するために、このような配列を構築し、ポインタを適切な値に設定する方法を示している。16s データ・タイプの例を示す。

例 3-3　画像プレーンへのポインタの値の設定

```
int stepBytes;
Ipp16s*  plane[3];

Ipp16s* img =  ippiMalloc_16s_P3( widthPixels, heightPixels,
&stepBytes )
plane[0] = img;
plane[1] = (Ipp16s*)((Ipp8u*)img + heightPixels * stepBytes);
plane[2] = (Ipp16s*)((Ipp8u*)img + 2 * heightPixels *
stepBytes);
```

戻り値

関数 ippiMalloc の戻り値は、アライメントされたメモリ・ブロックへのポインタである。システム内に利用可能なメモリがない場合は、NULL の値が返される。
割り当てられたメモリ・ブロックを解放するには、関数 ippiFree を使用しなければならない。

ippiFree

関数 *ippiMalloc* によって割り当てられたメモリを解放する。

```
void ippiFree(void* ptr);
```

引数

ptr	解放されるメモリ・ブロックへのポインタ。このブロックは、関数 ippiMalloc によってすでに割り当てられていなければならない。

説明

関数 ippiFree は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiMalloc によって割り当てられた、アライメントされたメモリ・ブロックを解放する。

注： 関数 ippiFree を使用して、malloc や calloc などの標準関数によって割り当てられたメモリを解放することはできない。また、ippiMalloc によって割り当てられたメモリを、free を使用して解放することはできない。

画像データ交換関数と初期化関数

4

本章では、画像データの操作、交換、初期化を実行するインテル® IPP 画像処理関数について説明する。

表 4-1 に、本章で詳しく説明する関数の一覧を示す。

表 4-1　画像データ交換関数と初期化関数

関数の基本名	操作
Convert	画像内のピクセル値の色分解能を変換する。
Scale	バッファのピクセル値をスケーリングし、色分解能を変換する。
Set	ピクセルの配列の値を設定する。
Copy	2 つのバッファの間でピクセル値をコピーする。
CopyConstBorder	2 つのバッファの間でピクセル値をコピーし、一定の値を持つ境界ピクセルを追加する。
CopyReplicateBorder	2 つのバッファの間でピクセル値をコピーし、複製された境界ピクセルを追加する。
SwapChannels	画像のチャネルの順序を変更する。
AddRandUniform_Direct	一様な分布を持つランダム・サンプルを生成し、画像データに追加する。
AddRandGauss_Direct	ガウス分布を持つランダム・サンプルを生成し、画像データに追加する。
ImageJaehne	Jaehne テスト画像を作成する。
ImageRamp	輝度の傾斜を持つテスト画像を作成する。
SampleLine	ラスタ・ライン上のすべてのピクセルをバッファに読み込む。
ZigzagFwd8x8	通常の順序をジグザグ・シーケンスに変換する。
ZigzagInv8x8	ジグザグ・シーケンスを通常の順序に変換する。

Convert

画像のピクセル値の色分解能を変換する。

事例 1：高い色分解能への変換

```
IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int
      srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize
      roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u16u_C1R	8u16s_C1R	8u32s_C1R	8s32s_C1R
8u16u_C3R	8u16s_C3R	8u32s_C3R	8s32s_C3R
8u16u_C4R	8u16s_C4R	8u32s_C4R	8s32s_C4R
8u16u_AC4R	8u16s_AC4R	8u32s_AC4R	8s32s_AC4R

8u32f_C1R	8s32f_C1R	16u32f_C1R	16s32f_C1R
8u32f_C3R	8s32f_C3R	16u32f_C3R	16s32f_C3R
8u32f_C4R	8s32f_C4R	16u32f_C4R	16s32f_C4R
8u32f_AC4R	8s32f_AC4R	16u32f_AC4R	16s32f_AC4R

事例 2：低い色分解能への変換
整数から整数型

```
IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int
      srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize
      roiSize);
```

サポートされる *mod* の値は、次のとおりである。

16u8u_C1R	16s8u_C1R	32s8u_C1R	32s8s_C1R
16u8u_C3R	16s8u_C3R	32s8u_C3R	32s8s_C3R
16u8u_C4R	16s8u_C4R	32s8u_C4R	32s8s_C4R
16u8u_AC4R	16s8u_AC4R	32s8u_AC4R	32s8s_AC4R

浮動小数点数から整数型

```
IppStatus ippiConvert_<mod>(const Ipp32f* pSrc, int srcStep,
      Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,
      IppRoundMode roundMode);
```

サポートされる *mod* の値は、次のとおりである。

32f8u_C1R	32f8s_C1R	32f16u_C1R	32f16s_C1R
32f8u_C3R	32f8s_C3R	32f16u_C3R	32f16s_C3R
32f8u_C4R	32f8s_C4R	32f16u_C4R	32f16s_C4R
32f8u_AC4R	32f8s_AC4R	32f16u_AC4R	32f16s_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ （バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ （バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ （ピクセル単位）
roundMode	丸めモード。*ippRndZero* または *ippRndNear*
ippRndZero	浮動小数点値をゼロの方向に切り捨てる。
ippRndNear	浮動小数点値を最も近い整数に丸める。

説明

関数 ippiConvert は、ippi.h ファイルの中で宣言される。この関数は、ソース・バッファ *pSrc* 内のピクセル値の色分解能を変換し、結果をデスティネーション・バッファ *pDst* に書き込む。スケーリングは行わない。浮動小数点型から整数型への変換時には、*roundMode* で定義された丸めが実行され、結果はデスティネーションのデータ・タイプの範囲に合わせて飽和処理される。

次の例は、スケーリングなしのデータ変換を示している。

例 4-1　スケーリングなしのデータ変換

```
IppStatus convert( void ) {
   IppiSize roi = {5,4};
   Ipp32f x[5*4];
   Ipp8u y[5*4];
   ippiSet_32f_C1R( -1.0f, x, 5, roi );
   x[1] = 300; x[2] = 150;
   return ippiConvert_32f8u_C1R( x, 5, y, 5, roi,
ippRndNear );
}
```

デスティネーション・イメージ *y* には、次のデータが含まれる。

```
00 FF 96 00 00
00 00 00 00 00
00 00 00 00 00
00 00 00 00 00
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

Scale

バッファのピクセル値をスケーリングし、
色分解能を変換する。

事例 1：スケーリングと、高い色分解能の整数データへの変換

```
IppStatus ippiScale_<mod>(const Ipp<srcDatatype>* pSrc, int
      srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize
      roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u16u_C1R	8u16s_C1R	8u32s_C1R
8u16u_C3R	8u16s_C3R	8u32s_C3R
8u16u_C4R	8u16s_C4R	8u32s_C4R
8u16u_AC4R	8u16s_AC4R	8u32s_AC4R

事例 2：スケーリングと、浮動小数点データへの変換

```
IppStatus ippiScale_<mod>(const Ipp8u* pSrc, int srcStep,
      Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f vMin,
      Ipp32f vMax);
```

サポートされる *mod* の値は、次のとおりである。

8u32f_C1R
8u32f_C3R
8u32f_C4R
8u32f_AC4R

事例 3：整数データのスケーリングと、低い色分解能への変換

```
IppStatus ippiScale_<mod>(const Ipp<srcDatatype>* pSrc, int
      srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize
      roiSize, IppHintAlgorithm hint);
```

サポートされる *mod* の値は、次のとおりである。

16u8u_C1R	16s8u_C1R	32s8u_C1R
16u8u_C3R	16s8u_C3R	32s8u_C3R
16u8u_C4R	16s8u_C4R	32s8u_C4R
16u8u_AC4R	16s8u_AC4R	32s8u_AC4R

事例 4：浮動小数点データのスケーリングと、整数データ・タイプへの変換

```
IppStatus ippiScale_<mod>(const Ipp32f* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f vMin,
      Ipp32f vMax);
```

サポートされる *mod* の値は、次のとおりである。

```
32f8u_C1R
32f8u_C3R
32f8u_C4R
32f8u_AC4R
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のストライドまたはステップ （バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のストライドまたはステップ （バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ （ピクセル単位）
vMin, *vMax*	入力データの最小値と最大値
hint	導入する関数のアルゴリズムを選択するオプション

説明

関数 ippiScale は、ippi.h ファイルの中で宣言される。この関数は、リニア・マッピングを使用して、ソース・バッファのピクセル値をデスティネーションのデータ・タイプに変換する。計算アルゴリズムは、*hint* 引数で指定する （表 11-2 を参照）。整数データ・タイプ同士の変換では、入力データ・タイプの全範囲 [*src_Min..src_Max*] が、出力データ・タイプの範囲 [*dst_Min..dst_Max*] にマッピングされる （データ・タイプと範囲の詳細は、第 2 章を参照）。ソース・ピクセル *p* をデスティネーション・ピクセル *p’* にマッピングする、次のスケーリング公式が使用される。

$$p' = dst\_Min + k*(p - src\_Min)$$

ここで、$k = (dst\_Max - dst\_Min)/(src\_Max - src\_Min)$

浮動小数点データ・タイプとの間の変換では、ユーザ定義の浮動小数点データ範囲 [*vMin..vMax*] が、ソースまたはデスティネーションのデータ・タイプの範囲にマッピングされる。

Ipp32f 型からの変換で、入力される浮動小数点値の一部が、指定した入力データ範囲 [*vMin..vMax*] から外れる場合は、それに対応する出力値は飽和処理される。画像内にある実際の浮動小数点データの範囲を求めるには、関数 ippiMinMax を使用する。

次の例は、スケーリングを使用してデータ範囲を維持する方法を示している。

例 4-2　スケーリングを使用するデータ変換

```
IppStatus scale( void ) {
   IppiSize roi = {5,4};
   Ipp32f x[5*4];
   Ipp8u y[5*4];
   ippiSet_32f_C1R( -1.0f, x, 5, roi );
   x[1] = 300; x[2] = 150;
   return ippiScale_32f8u_C1R( x, 5, y, 5, roi, -1, 300 );
}
```

デスティネーション・イメージ *y* には、次のデータが含まれる。

```
00 FF 80 00 00
00 00 00 00 00
00 00 00 00 00
00 00 00 00 00
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

ippStsScaleRangeErr	入力データの上下限が不適当である（すなわち、*vMax* が *vMin* より小さいか *vMin* に等しい）場合のエラー状態を示す。

Set

ピクセルの配列の値を設定する。

事例 1：1 チャネル・データの値を設定する。

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>*
     pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R

事例 2：各カラー・チャネルを指定された値に設定する。

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[3],
     Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

事例 3：カラー・チャネルとアルファ・チャネルを指定された値に設定する。

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[4],
     Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4R	16s_C4R	32f_C4R

事例 4：マスクされた 1 チャネル・データの値を設定する。

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>*
     pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask,
     int maskStep);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1MR	16s_C1MR	32f_C1MR

事例 5：マスクされたマルチチャネル・データのカラー・チャネルを指定された値に設定する。

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[3],
     Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const
     Ipp8u* pMask, int maskStep);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3MR	16s_C3MR	32f_C3MR
8u_AC4MR	16s_AC4MR	32f_AC4MR

事例 6：マスクされたマルチチャネル・データのすべてのチャネルを指定された値に設定する。

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[4],
     Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const
     Ipp8u* pMask, int maskStep);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4MR	16s_C4MR	32f_C4MR

事例 7：マルチチャネル・データのうち、選択されたチャネルの値を設定する。

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>*
     pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3CR	16s_C3CR	32f_C3CR
8u_C4CR	16s_C4CR	32f_C4CR

引数

value	デスティネーション・バッファ内の各ピクセルをこの定数値に設定する。
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のストライドまたはステップ （バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ （ピクセル単位）

pMask	マスク・イメージ・バッファへのポインタ
maskStep	マスク・イメージ・バッファ内のストライドまたはステップ （バイト単位）

説明

関数 ippiSet は、ippi.h ファイルの中で宣言される。この関数は、デスティネーション・バッファ *pDst* 内のピクセル値を定数 *value* に設定する。
矩形 ROI 内のすべてのピクセル値を設定することも、指定されたマスク *pMask* によって選択されたピクセル値だけを設定することも可能である。マスク操作を行う場合、関数は空間的に対応するマスク配列の値が 0 でない場合にのみピクセル値をデスティネーション・バッファに設定する。
処理対象チャネルが選択されている場合、すなわちマルチチャネル・イメージのうち 1 チャネルだけが設定される場合は （**事例 7** を参照）、*pDst* ポインタはそのチャネル内の ROI バッファの始点を指す。
ソース・イメージ・データにアルファ・チャネルがある場合は、選択した ippiSet 関数のタイプに基づいて、アルファ成分は無視されるか、特定の値に設定される。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pMask* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*dstStep* または *maskStep* の値が 0 または負の場合のエラー状態を示す。

Copy

2 つのバッファの間でピクセル値をコピーする。

事例 1：すべてのカラー・チャネルのすべてのピクセルのコピー

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int
      srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
      roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R
8u_C3AC4R	16s_C3AC4R	32f_C3AC4R
8u_AC4C3R	16s_AC4C3R	32f_AC4C3R

事例 2：マスクされたピクセルのみの操作

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int
     srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
     roiSize, const Ipp8u* pMask, int maskStep);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1MR	16s_C1MR	32f_C1MR
8u_C3MR	16s_C3MR	32f_C3MR
8u_C4MR	16s_C4MR	32f_C4MR
8u_AC4MR	16s_AC4MR	32f_AC4MR

事例 3：マルチチャネル画像内の選択したチャネルのコピー

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int
     srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
     roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3CR	16s_C3CR	32f_C3CR
8u_C4CR	16s_C4CR	32f_C4CR

事例 4：1 チャネル画像への選択したチャネルのコピー

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int
     srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
     roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3C1R	16s_C3C1R	32f_C3C1R
8u_C4C1R	16s_C4C1R	32f_C4C1R

事例 5：マルチチャネル画像への 1 チャネル画像へのコピー

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int
srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1C3R	16s_C1C3R	32f_C1C3R
8u_C1C4R	16s_C1C4R	32f_C1C4R

事例 6：別々のプレーンへのカラー・イメージの分割

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int
     srcStep, Ipp<datatype>* const pDst[3], int dstStep,
     IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3P3R	16s_C3P3R	32f_C3P3R

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int
     srcStep, Ipp<datatype>* const pDst[4], int dstStep,
     IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4P4R	16s_C4P4R	32f_C4P4R

事例 7：別々のプレーンからのカラー・イメージの合成

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* const pSrc[3],
     int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
     roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3C3R	16sP3C3R	32f_P3C3R

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* const pSrc[4],
     int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
     roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4C4R	16s_P4C4R	32f_P4C4R

引数

pSrc	ソース・バッファへのポインタ。**事例 7** の場合は、ソース・イメージのカラー・プレーンへのポインタを格納する配列
srcStep	ソース・イメージ・バッファ内のストライドまたはステップ（バイト単位）

pDst	デスティネーション・バッファへのポインタ。**事例 6** の場合は、得られたカラー・プレーンへのポインタを格納する配列
dstStep	デスティネーション・イメージ・バッファ内のストライドまたはステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
pMask	マスク・イメージ・バッファへのポインタ
maskStep	マスク・イメージ・バッファ内のストライドまたはステップ（バイト単位）

説明

関数 ippiCopy は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ・バッファ *pSrc* からデスティネーション・バッファ *pDst* へデータをコピーする。ピクセル・バッファ間で画像データをコピーする。マスク *pMask* によって選択されたピクセルのコピー操作をサポートしている。

マスク操作の場合、この関数は、（以下のコード例に示すように）空間的に対応するマスク配列の値がゼロでない場合にのみ、ピクセル値をデスティネーション・バッファに書き込む。

処理対象チャネル記述子（C）が付いた関数の変種は、ソース・マルチチャネル画像の指定した 1 チャネルだけを他のマルチチャネル画像にコピーする（**事例 3** を参照）。これらの関数では、*pSrc* ポインタと *pDst* ポインタは、それぞれソース・イメージとデスティネーション・イメージの該当チャネル内の ROI バッファの始点を指す。

関数の変種には、マルチチャネル画像の指定した 1 チャネル *pSrc* から 1 チャネル画像 *pDst* にデータをコピーするものや（**事例 4** を参照）、1 チャネル画像 *pSrc* からマルチチャネル画像の指定した 1 チャネル *pDst* にデータをコピーするものがある（**事例 5** を参照）。

関数 ippiCopy を使用して、カラー・イメージを別々のプレーンに分割することができる（**事例 6** を参照）。

次のコード例は、マスクされたデータをコピーする方法を示している。

例 4-3　マスクされたデータのコピー

```
IppStatus  copyWithMask( void ) {
   Ipp8u mask[3*3], x[5*4], y[5*4]={0};
   IppiSize imgroi={5,4}, mskroi={3,3};
   ippiSet_8u_C1R( 3, x, 5, imgroi );
   /// set mask with a hole in upper left corner
   ippiSet_8u_C1R( 1, mask, 3, mskroi );
   mask[0] = 0;
   /// copy roi with mask
   return ippiCopy_8u_C1MR( x, 5, y, 5, mskroi, mask, 3 );
}
```

デスティネーション・イメージ *y* には、次のデータが含まれる。

```
00 03 03 00 00
03 03 03 00 00
03 03 03 00 00
00 00 00 00 00
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタが NULL の場合のエラー状態を示す。ただし、**事例 4** の第 2 のモードを除く。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	次の場合のエラー状態を示す。 *srcStep*、*dstStep*、または *maskStep* の値が 0 または負の場合。 **事例 4** と**事例 5** については、srcStep または *dstStep* . *roiSize.width* * <pixelSize> より小さい場合。

CopyConstBorder

2 つのバッファの間でピクセル値をコピーし、一定の値を持つ境界ピクセルを追加する。

事例 1：1 チャネル・データの操作

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst,
      int dstStep, IppiSize dstRoiSize, int topBorderWidth, int
      leftBorderWidth, Ipp<datatype> value);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32s_C1R

事例 2：マルチチャネル・データの操作

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst,
      int dstStep, IppiSize dstRoiSize, int topBorderWidth, int
      leftBorderWidth, const Ipp<datatype> value[3]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16s_C3R	32s_C3R
8u_AC4R	16s_AC4R	32s_AC4R

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst,
      int dstStep, IppiSize dstRoiSize, int topBorderWidth, int
      leftBorderWidth, const Ipp<datatype> value[4]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4R	16s_C4R	32s_C4R

引数

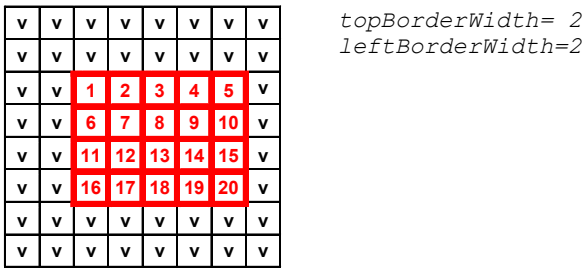
pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ

dstStep	デスティネーション・イメージ・バッファ内のステップ （バイト単位）
srcRoiSize	ソース ROI のサイズ （ピクセル単位）
dstRoiSize	デスティネーション ROI のサイズ （ピクセル単位）
topBorderWidth	上の境界の幅 （ピクセル単位）
leftBorderWidth	左の境界の幅 （ピクセル単位）
value	境界ピクセルに割り当てられる一定の値 （マルチチャネル画像の場合は一定のベクトル）

説明

関数 ippiCopyConstBorder は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ *pSrc* をデスティネーション・イメージ *pDst* にコピーし、コピーされた領域の外側に境界を作成する。境界のピクセル値は、*value* 引数によって渡される、指定された一定の値に設定される。パラメータ *topBorderWidth* および *leftBorderWidth* は、デスティネーション・イメージの ROI 内のソース ROI の最初のピクセルの位置を指定する（図 4-1 を参照）。赤でマークされた正方形は、ソース・イメージからコピーされたピクセルに対応する。
デスティネーション ROI の高さ（幅）は、ソース ROI の高さ（幅）と *topBorderWidth*（*leftBorderWidth*）パラメータの和と同じかそれ以上でなければならない。

図 4-1　一定の値を持つピクセルの境界の作成



戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。

ippStsSizeErr	エラー状態を示す。*roiSize* のフィールドの値がゼロまたは負であるか、 または *topBorderWidth* または *leftBorderWidth* がゼロより小さいか、 または *dstRoiSize.width* < *srcRoiSize.width* + *leftBorderWidth*、 または *dstRoiSize.height* < *srcRoiSize.height* + *topBorderWidth* になっている。
ippStsStepErr	*srcStep* または *dstStep* の値が0または負の場合のエラー状態を示す。

CopyReplicateBorder

2 つのバッファの間でピクセル値をコピーし、複製された境界ピクセルを追加する。

```
IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>*
      pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>*
      pDst, int dstStep, IppiSize dstRoiSize, int
      topBorderWidth, int leftBorderWidth);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32s_C1R
8u_C3R	16s_C3R	32s_C3R
8u_C4R	16s_C4R	32s_C4R
8u_AC4R	16s_AC4R	32s_AC4R

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
srcRoiSize	ソース ROI のサイズ（ピクセル単位）
dstRoiSize	デスティネーション ROI のサイズ（ピクセル単位）

topBorderWidth	上の境界の幅（ピクセル単位）
leftBorderWidth	左の境界の幅（ピクセル単位）

説明

関数 ippiCopyReplicateBorder は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ *pSrc* をデスティネーション・イメージ *pDst* にコピーし、図 4-1 に示す方式に従って、デスティネーション・イメージ内のコピーされた領域の外側のピクセル（「境界」）をソース・イメージのピクセル値で充填する。赤でマークされた正方形は、ソース・イメージからコピーされたピクセルに対応する。パラメータ *topBorderWidth* および *leftBorderWidth* は、デスティネーション・イメージの ROI 内のソース ROI の最初のピクセルの位置を指定する。
デスティネーション ROI の高さ（幅）は、ソース ROI の高さ（幅）と *topBorderWidth*（*leftBorderWidth*）パラメータの和と同じかそれ以上でなければならない。

図 4-2　複製された境界の作成

1	1	1	2	3	4	5	5
1	1	1	2	3	4	5	5
1	1	1	2	3	4	5	5
6	6	6	7	8	9	10	10
11	11	11	12	13	14	15	15
16	16	16	17	18	19	20	20
16	16	16	17	18	19	20	20
16	16	16	17	18	19	20	20

topBorderWidth= 2
leftBorderWidth=2

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	エラー状態を示す。*roiSize* のフィールドの値がゼロまたは負であるか、 または *topBorderWidth* または *leftBorderWidth* がゼロより小さいか、 または *dstRoiSize.width* < *srcRoiSize.width* + *leftBorderWidth* になっているか、

	または *dstRoiSize.height* < *srcRoiSize.height* + *topBorderWidth* になっている。
ippStsStepErr	*srcStep* または *dstStep* の値が0または負の場合のエラー状態を示す。

SwapChannels

画像のチャネルの順序を変更する。

事例 1：非インプレース操作

```
IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc,
     int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
     roiSize, const int dstOrder[3]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16u_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	32s_AC4R	32f_AC4R

事例 2：インプレース操作

```
ippiSwapChannels_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep,
     IppiSize roiSize, const int dstOrder[3]);
```

引数

pSrc	ソース・バッファのポインタ
pDst	デスティネーション・バッファのポインタ
pSrcDst	ソースおよびデスティネーション・バッファへのポインタ（インプレース操作の場合）
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ （バイト単位）

roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
dstOrder	デスティネーション・イメージ内のチャネルの順序

説明

関数 ippiSwapChannels は、ippi.h ファイルの中で宣言される。この関数は、ソース・バッファの各ピクセルのチャネルの順序を変更し、入れ替えられた値をデスティネーション・バッファに格納する。デスティネーション・イメージ内の最初のチャネルは、*dstOrder* の最初の構成要素によって決まる。この値は [0..2] の範囲内で、ソース・イメージの対応するチャネル番号を示す。他のチャネルも同様の方法で指定される。例えば、ソース・イメージ内のチャネルの順序が *A*、*B*、*C* で、*dstOrder*[0]=2, *dstOrder*[1]=0, *dstOrder*[2]=1, である場合、デスティネーション・イメージ内のチャネルの順序は *C*、*A*、*B* になる。*dstOrde*r の構成要素のうちいくつかまたはすべてに、同じ値を指定できる。

この関数は、アルファ・チャネルの操作を実行しない。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsSizeErr	roiSize フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsChannelOrderErr	*dstOrder* が [0..2] の範囲外の場合のエラー状態を示す。

AddRandUniform_Direct

一様な分布を持つランダム・サンプルを生成し、画像データに追加する。

```
IppStatus ippiAddRandUniform_Direct_<mod>(Ipp<datatype>*
     pSrcDst, int srcDstStep, IppiSize roiSize, Ipp<datatype>
     low, Ipp<datatype> high, unsigned int* pSeed);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16s_AC4IR	32f_AC4IR

引数

pSrcDst	ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）
low	一様に分布する値の範囲の下限
high	一様に分布する値の範囲の上限
pSeed	疑似乱数ジェネレータの初期シード値

説明

関数 ippiAddRandUniform_Direct は、ippi.h ファイルの中で宣言される。この関数は、範囲 [*low, high*] にわたって一様な分布を持つサンプルを生成し、*pSrcDst* で指定されるソース・イメージに追加する。

得られたピクセル値のうち画像のデータ範囲を超える部分は、それぞれのデータ範囲の上下限に合わせて飽和処理される。一様な分布を持つ純粋ノイズ画像を生成するには、0 のデータで構成されるソース・イメージを入力として使用し、ippiAddRandUniform_Direct を呼び出す。

次のコード例は、ランダム・サンプルの生成を示している。

例 4-4　ランダム・サンプルの生成

```
IppStatus randUniform( void ) {
   unsigned int seed = 123456;
   Ipp8u img[2048], mn, mx;
   IppiSize roi = { 2048, 1 };
   Ipp64f mean;
   IppStatus st;
   ippiSet_8u_C1R( 0, img, 8*4, roi );
   st = ippiAddRandUniform_Direct_8u_C1IR( img, 2048, roi, 0,
   255, &seed );
   ippiMean_8u_C1R( img, 2048, roi, &mean );
   ippiMinMax_8u_C1R( img, 2048, roi, &mn, &mx );
   printf( "[%d..%d], mean=%.3f\n", mn, mx, mean );
   return st;
}
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrcDst* または *pSeed* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcDstStep* の値が 0 または負の場合のエラー状態を示す。

AddRandGauss_Direct

ガウス分布を持つランダム・サンプルを生成し、画像データに追加する。

```
IppStatus ippiAddRandGauss_Direct_<mod>(Ipp<datatype>*
      pSrcDst, int srcDstStep, IppiSize roiSize, Ipp<datatype>
      mean, Ipp<datatype> stdev, unsigned int* pSeed);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16s_AC4IR	32f_AC4IR

引数

pSrcDst	ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）
mean	ガウス分布の平均
stdev	ガウス分布の標準偏差
pSeed	疑似乱数ジェネレータの初期シード値

説明

関数 ippiAddRandGauss_Direct は、ippi.h ファイルの中で宣言される。この関数は、平均値 *mean* と標準偏差 *stdev* を持つガウス分布サンプルを生成し、*pSrcDst* で指定されるソース・イメージに追加する。

得られたピクセル値のうち画像のデータ範囲を超える部分は、それぞれのデータ範囲の上下限に合わせて飽和処理される。ガウス分布を持つ純粋ノイズ画像を生成するには、0 のデータで構成されるソース・イメージを入力として使用し、ippiAddRandGauss_Direct を呼び出す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrcDst* または *pSeed* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcDstStep* の値が 0 または負の場合のエラー状態を示す。

ImageJaehne

Jaehne テスト画像を作成する。

```
IppStatus ippiImageJaehne_<mod>(Ipp<datatype>* pDst, int
      DstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	8s_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	8s_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	8s_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	8s_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

引数

pDst	デスティネーション・バッファへのポインタ
DstStep	デスティネーション・イメージ・バッファ内のステップ （バイト単位）
roiSize	デスティネーション・イメージの ROI のサイズ （ピクセル単位）

説明

関数 ippiImageJaehne は、ippi.h ファイルの中で宣言される。この関数は、B.Jaehne （「Jaehne95」を参照）によってデジタル画像処理に導入された、1 チャネルまたは 3 チャネルのテスト画像を作成する。

デスティネーション・イメージのピクセル値は、次の公式に従って計算される。

```
Dst(x,y) = A*sin(0.5*IPP_PI* (x22 + y22) / roiSize.height),
```

ここで、

x,y は、次の範囲内で変化するピクセル座標である。
0 ≤ x ≤ roiSize.width-1, 0 ≤ y ≤ roiSize.height-1;

IPPI_PI は、π の値を表すライブラリ定数である。

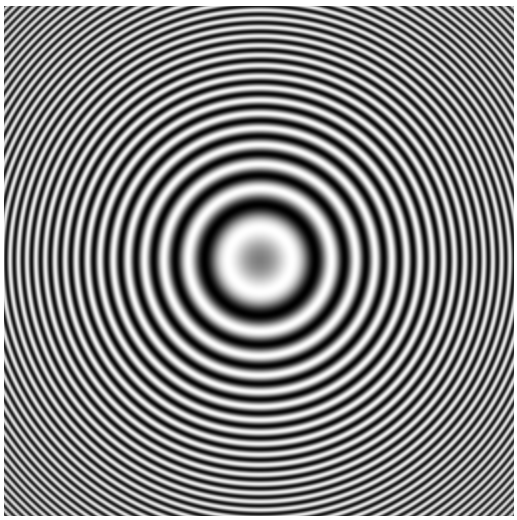
```
x2 = (x-roiSize.width+1)/2.0 ,
y2 = (y-roiSize.height+1)/2.0;
```

A は、作成される画像のタイプによって異なる定数値である。

32f 浮動小数点データの場合、作成される画像内のピクセル値は、0 （この値を含む）と 1 （この値を含まない）の間の範囲内で変化する。

図 4-3 は、ippiImageJaehne 関数によって生成されるテスト画像の例を示している。

図 4-3　生成される Jaehne テスト画像の例



これらのテスト画像は、[Jaehne95] で提案されているように、フィルタリング関数を適用した結果を視覚化して解釈する場合に効果的である。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負である場合と、*DstStep* の値が 0 より小さいか 0 である場合のエラー状態を示す。

ImageRamp

輝度の傾斜を持つテスト画像を作成する。

```
IppStatus ippiImageRamp_<mod>(Ipp<datatype>* pDst, int
      DstStep, IppiSize roiSize, float offset, float slope,
      IppiAxis axis);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	8s_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	8s_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	8s_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	8s_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

引数

pDst	デスティネーション・バッファへのポインタ
DstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	デスティネーション・イメージの ROI のサイズ（ピクセル単位）
offset	オフセット値
slope	傾斜係数
axis	画像の輝度の傾斜の方向を指定する。値は次のいずれかである。

ippAxsHorizontal	X 方向の傾斜
ippAxsVertical	Y 方向の傾斜
ippAxsBoth	X 方向と Y 方向の傾斜

説明

関数 ippiImageRamp は、ippi.h ファイルの中で宣言される。この関数は、各種の画像処理関数を適用した結果を検討する際にテスト画像として使用できる、1 チャネルまたは 3 チャネルの画像を生成する。
デスティネーション・イメージのピクセル値は、次のいずれかの公式に従って計算される。

```
Dst(x,y) = offset + slope * x      , if axis = ippAxsHorizontal;
Dst(x,y) = offset + slope * y      , if axis = ippAxsVertical;
Dst(x,y) = offset + slope * x * y , if axis = ippAxsBoth;
```

ここで、

x,y は、次の範囲内で変化するピクセル座標である。
0 ≤ x ≤ roiSize.width-1, 0 ≤ y ≤ roiSize.height-1

ただし、リニア変換係数 *offset* および *slope* の値は、すべての関数タイプで浮動小数点値になる。計算されたピクセル値のうち画像のデータ範囲を超える部分は、それぞれのデータ範囲の上下限に合わせて飽和処理される。
次のコード例は、ippiImageRamp 関数の使用方法を示している。

例 4-5 ippiImageRamp 関数によるテスト画像の作成

```
IppStatus ramp( void ) {
   Ipp8u dst[8*4];
   IppiSize roiSize = { 8, 4 };
   return ippiImageRamp_8u_C1R( dst, 8, roiSize, 0.0f,
256.0f/7, ippAxsHorizontal );
}
 The destination image contains the following data:
 00 25 49 6E 92 B7 DB FF
 00 25 49 6E 92 B7 DB FF
 00 25 49 6E 92 B7 DB FF
 00 25 49 6E 92 B7 DB FF
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合と、*DstStep* の値が 0 より小さいか 0 である場合のエラー状態を示す。

SampleLine

ラスタ・ラインをバッファに格納する。

```
IppStatus ippiSampleLine_<mod>(const Ipp<DataType>* pSrc, int
srcStep,
    IppiSize roiSize, Ipp<DataType>* pDst, IppiPoint pt1,
IppiPoint pt2);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R

引数

pSrc	ラスタ・イメージへのポインタ
srcStep	ラスタ・イメージ内のステップ （バイト単位）
roiSize	画像の ROI のサイズ
pDst	デスティネーション・バッファへのポインタ。バッファは、少なくとも max (|*pt2.x* – *pt1.x*| + 1, |*pt2.y* – *pt1.y*| + 1) 点で結果を格納する。
pt1	ラインの始点
pt2	ラインの終点

説明

関数 SampleLine は、ippi.h ファイルの中で宣言される。この関数は、8 点接続 Bresenham アルゴリズムを使用して、ラスタ・ラインに属する複数の点で反復され、得られたピクセルをデスティネーション・バッファに格納する。

戻り値

ippStsNullPtrErr	指定されたポインタが NULL の場合のエラーを示す。
ippStsSizeErr	*roiSize.width*、または *roiSize.height* の値が負の場合のエラーを示す。
ippStsStepErr	*srcStep* が *roiSize.width * <pixelSize>* より小さい場合のエラーを示す。
ippStsNotEvenStepErr	浮動小数点画像のステップが 4 で割り切れない場合のエラーを示す。
ippStsOutOfRangeErr	ラインの終点が画像の外側にある場合のエラーを示す。

ZigzagFwd8x8

通常の順序をジグザグ順序に変換する。

```
IppStatus ippiZigzagFwd8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

引数

pSrc	ソース・データへのポインタ
pDst	デスティネーション・データへのポインタ

説明

関数 ippiZigzagFwd8x8 は、ippi.h ファイルで宣言される。この関数は、8 × 8 ブロック内のデータを通常の順序 （左から右、上から下）からジグザグ・シーケンスに変更する。ジグザグ・シーケンスを図 4-4 に示す。

図 4-4　ジグザグ・シーケンス

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。

ZigzagInv8x8

ジグザグ順序を通常の順序に変換する。

```
IppStatus ippiZigzagInv8x8_16s_C1(const Ipp16s* pSrc,  Ipp16s* pDst);
```

引数

pSrc	ソース・データへのポインタ
pDst	デスティネーション・データへのポインタ

説明

関数 ippiZigzagInv8x8 は、ippi.h ファイルの中で宣言される。この関数は、8 × 8 ブロック内のデータをジグザグ順序から通常の順序（左から右、上から下）に変更する。ジグザグ・シーケンスを図 4-4 に示す。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。

画像の算術演算と論理演算 5

本章では、算術演算または論理演算を使用してイメージ・バッファのピクセル値を修正するインテル® IPP 画像処理関数について説明する。また、透過度（アルファ・ブレンディング）に基づいて画像の合成を実行する関数についても説明する。

表 5-1 に、本章で詳しく説明する関数の一覧を示す。

表 5-1　算術演算と論理演算

関数の基本名	操作
算術演算関数	
Add	2 つのイメージ・バッファのピクセル値の和を求める。
AddC	イメージ・バッファのピクセル値に定数を加算する。
AddSquare	ソース・イメージのピクセル値を 2 乗し、アキュムレータの画像の浮動小数点ピクセル値に加算する。
AddProduct	2 つのソース・イメージのピクセル値の積を求め、アキュムレータの画像の浮動小数点ピクセル値に加算する。
AddWeighted	ソース・イメージのピクセル値に係数 α を掛けて、アキュムレータの画像の浮動小数点ピクセル値に係数（1-α）を掛けた値に加算する。
Mul	2 つのイメージ・バッファのピクセル値の積を求める。
MulC	イメージ・バッファのピクセル値に定数を掛ける。
MulScale	2 つのイメージ・バッファのピクセル値の積を求め、その積をスケーリングする。
MulCScale	イメージ・バッファのピクセル値に定数を掛けて、その積をスケーリングする。
Sub	2 つのイメージ・バッファのピクセル値の差を求める。
SubC	イメージ・バッファのピクセル値から定数を引く。
Div	2 つのイメージ・バッファのピクセル値の間で除算を行う。
DivC	イメージ・バッファのピクセル値を定数で割る。
Abs	ソース・イメージのピクセルの絶対値を計算し、デスティネーション・イメージに格納する。
AbsDiff	2 つの画像の差の絶対値を求める。
AbsDiffC	画像とスカラ値の差の絶対値を求める。
Sqr	画像のピクセル値を 2 乗し、デスティネーション・イメージに書き込む。
Sqrt	ソース・イメージのピクセル値の平方根を計算し、デスティネーション・イメージに書き込む。

続く

表 5-1　算術演算と論理演算　（続き）

関数の基本名	操作
Ln	ソース・イメージのピクセル値の自然対数関数を計算し、その結果をデスティネーション・イメージに書き込む。
Exp	ソース・イメージのピクセル値の指数関数を計算し、その結果をデスティネーション・イメージに書き込む。
Complement	負の数値を補数から直接コードに変換する。
論理演算関数	
And	ビット単位の AND 演算を使用して、2 つのイメージ・バッファの対応するピクセルを合成する。
AndC	各ピクセルと定数の間で、ビット単位の AND 演算を実行する。
Or	ビット単位の OR 演算を使用して、2 つのイメージ・バッファの対応するピクセルを合成する。
OrC	各ピクセルと定数の間で、ビット単位の OR 演算を実行する。
Xor	ビット単位の XOR 演算を使用して、2 つのイメージ・バッファの対応するピクセルを合成する。
XorC	各ピクセルと定数の間で、ビット単位の XOR 演算を実行する。
Not	各ピクセルに対して、ビット単位の NOT 演算を実行する。
RShiftC	ピクセル値の各ビットを右に論理シフトまたは算術シフトすることによって、2 を定数で累乗した値でピクセル値を割る。
LShiftC	ピクセル値の各ビットを左にシフトする。
アルファ・ブレンディング関数	
AlphaComp	アルファ （透過度）値を使用して、2 つのイメージ・バッファを合成する。
AlphaCompC	一定のアルファ値を使用して、2 つのイメージ・バッファを合成する。
AlphaPremul	イメージ・バッファのピクセル値とアルファ値の積を事前に求める。
AlphaPremulC	一定のアルファ値を使用して、イメージ・バッファのピクセル値を事前に乗算する。

名前にサフィックス C が含まれている関数は、定数を使用する演算を実行する。整数データを操作する算術演算関数は、内部で計算された結果に対して固定スケーリングを実行する。オーバーフローが発生した場合は、結果の値はデスティネーションのデータ・タイプの範囲に合わせて飽和処理される。

注： ほとんどの算術演算関数および論理演算関数は、1 チャネル、3 チャネル、または 4 チャネルのピクセル値を含むデータをサポートしている。アルファ・チャネルがある場合（AC4）、アルファ・チャネルは処理されないことに注意する。

算術演算

本節で説明する関数は、ピクセル値の算術演算を実行する。算術演算には、2 つの画像のピクセル値の加算、乗算、減算、除算と、1 つの画像と定数の間の加算、乗算、減算、除算がある。イメージ・バッファ内のピクセルの絶対値、2 乗、平方根、指数関数、自然対数関数の計算を行う関数も用意されている。
算術演算関数は、ソース・バッファ内の各ピクセルの演算を実行し、その結果をデスティネーション・バッファに書き込む。一部の関数は、複素数データを使用した画像の処理にも対応している。

Add

2 つのイメージ・バッファのピクセル値の
和を求める。

事例 1：整数データまたは複素数データに対する非インプレース操作

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
      const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
      pDst,
      int dstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs    16s_C1RSfs    16sc_C1RSfs    32sc_C1RSfs
8u_C3RSfs    16s_C3RSfs    16sc_C3RSfs    32sc_C3RSfs
8u_AC4RSfs   16s_AC4RSfs   16sc_AC4RSfs   32sc_AC4RSfs
8u_C4RSfs    16s_C4RSfs
```

事例 2：浮動小数点データまたは複素数データに対する非インプレース操作

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
     const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
     pDst, int
     dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R          32fc_C1R
32f_C3R          32fc_C3R
32f_AC4R         32fc_AC4R
32f_C4R
```

事例 3：整数データまたは複素数データに対するインプレース操作

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
     int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IRSfs     16s_C1IRSfs    16sc_C1IRSfs   32sc_C1IRSfs
8u_C3IRSfs     16s_C3IRSfs    16sc_C3IRSfs   32sc_C3IRSfs
8u_AC4IRSfs    16s_AC4IRSfs   16sc_AC4IRSfs  32sc_AC4IRSfs
8u_C4IRsfs     16s_C4IRSfs
```

事例 4：浮動小数点データまたは複素数データに対するインプレース操作

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1IR       32fc_C1IR
32f_C3IR       32fc_C3IR
32f_AC4IR      32fc_AC4IR
32f_C4IR
```

事例 5：浮動小数点アキュムレータの画像を使用するインプレース操作

```
IppStatus ippiAdd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
     Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u32f_C1IR     8s32f_C1IR
```

事例 6：浮動小数点アキュムレータの画像を使用する、マスクを使用したインプレース操作

```
IppStatus ippiAdd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
      const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int
      srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

`8u32f_C1IMR` `8s32f_C1IMR` `32f_C1IMR`

引数

pSrc, pSrc1, pSrc2	ソース・バッファへのポインタ
srcStep, src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
dstSrcStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ （バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
scaleFactor	整数結果のスケーリングに使用される係数

説明

関数 `ippiAdd` は、`ippi.h` ファイルの中で宣言される。**事例 5** および**事例 6** での関数の種類の場合は例外で、`ippcv.h` ファイルの中で宣言される。この関数は、2 つのソース・イメージ・バッファの対応するピクセル値の和を求め、その結果をデスティネーション・バッファに格納する。整数データの演算の場合、得られた値は *scaleFactor* によってスケーリングされる。複素数データの場合は、ピクセル値の実数部と虚数部の両方が処理される。

一部の関数タイプは、`8u`、`8s`、または `32f` 型のソース・イメージのピクセル値を、インプレースで浮動小数点アキュムレータの画像に加算する。マスク演算の場合、各ピクセル値は、それに対応するマスク値が 0 でない場合にのみ加算される。マスク値が 0 の場合は、アキュムレータの画像のピクセル値は変更されない。

AC4 記述子を持つ関数は、アルファ・チャネルは処理されないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	次の場合のエラー状態を示す。 指定されたバッファのステップ値のいずれかが 0 または負の場合。 アキュムレータの画像を使用する関数で、*srcStep*、*maskStep*、または *srcDstStep* が *roiSize.width* * <pixelSize> より小さいか、あるいは浮動小数点画像のステップ数が 4 で割り切れない場合。

AddC

イメージ・バッファのピクセル値に定数を加算する。

事例 1：1 チャネル整数データまたは複素数データに対する非インプレース操作

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs      16s_C1RSfs      16sc_C1RSfs      32sc_C1RSfs
```

事例 2：マルチチャネル整数データまたは複素数データに対する非インプレース操作

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int
      srcStep, const
      Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3RSfs      16s_C3RSfs     16sc_C3RSfs    32sc_C3RSfs
8u_AC4RSfs     16s_AC4RSfs    16sc_AC4RSfs   32sc_AC4RSfs
```

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int
     srcStep, const
     Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
     IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4RSfs      16s_C4RSfs
```

事例 3：1 チャネル浮動小数点データまたは複素数データに対する非インプレース操作

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
     IppiSize
     roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R        32fc_C1R
```

事例 4：マルチチャネル浮動小数点データまたは複素数データに対する非インプレース操作

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int
     srcStep, const
     Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
     IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R        32fc_C3R
32f_AC4R       32fc_AC4R
```

```
IppStatus ippiAddC_32f_C4R(const Ipp32f* pSrc, int srcStep, const
     Ipp32f value[4], Ipp32f* pDst, int dstStep, IppiSize
     roiSize);
```

事例5:1 チャネル整数データまたは複素数データに対するインプレース操作

```
IppStatus ippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>*
     pSrcDst,
     int srcDstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IRSfs     16s_C1IRSfs    16sc_C1IRSfs   32sc_C1IRSfs
```

事例 6：マルチチャネル整数データまたは複素数データに対するインプレース操作

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3],
      Ipp<datatype>*
      pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3IRSfs	16s_C3IRSfs	16sc_C3IRSfs	32sc_C3IRSfs
8u_AC4IRSfs	16s_AC4IRSfs	16sc_AC4IRSfs	32sc_AC4IRSfs

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[4],
      Ipp<datatype>*
      pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4IRSfs	16s_C4IRSfs

事例 7：1 チャネル浮動小数点データまたは複素数データに対するインプレース操作

```
IppStatus ippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>*
      pSrcDst,
      int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1IR	32fc_C1IR

事例 8：マルチチャネル浮動小数点データまたは複素数データに対するインプレース操作

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3],
      Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR

```
IppStatus ippiAddC_32f_C4IR(const Ipp32f value[4],
      Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
value	画像のピクセル値に加算される定数値（マルチチャネル・イメージの場合は定数ベクトル）

pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
scaleFactor	整数結果のスケーリングに使用される係数

説明

関数 ippiAddC は、ippi.h ファイルの中で宣言される。この関数は、画像のピクセル値に *value* を加算することによって、画像の輝度を変更する。1 チャネル・イメージの場合、正の *value* を指定すると、画像が明るくなる（輝度が上がる）。負の *value* を指定すると、画像が暗くなる（輝度が下がる）。マルチチャネル・イメージの場合は、ピクセルの各チャネルの値に定数ベクトル *value* の成分を加算する。複素数データの場合は、ピクセル値の実数部と虚数部の両方が処理される。整数データの演算の場合、得られた値は *scaleFactor* によってスケーリングされる。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

AddSquare

ソース・イメージのピクセル値を 2 乗し、アキュムレータの画像の浮動小数点ピクセル値に加算する。

事例 1：インプレース操作

```
IppStatus ippiAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int
      srcStep,
      Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u32f_C1IR      8s32f_C1IR      32f_C1IR
```

事例 2：マスクを使用したインプレース操作

```
IppStatus ippiAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int
      srcStep,
      const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int
      srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u32f_C1IMR     8s32f_C1IMR     32f_C1IMR
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ （バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ （バイト単位）
pSrcDst	デスティネーション（アキュムレータ）イメージへのポインタ
srcDstStep	アキュムレータの画像内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）

説明

関数 ippiAddSquare は、ippcv.h ファイルの中で宣言される。この関数は、次のように、ソース・イメージ *pSrc* のピクセル値を 2 乗し、アキュムレータの画像 *pSrcDst* の浮動小数点ピクセル値に加算する。

pSrcDst(x,y) = pSrcDst(x,y) + pSrc(x,y)2

マスク演算の場合、2 乗されたピクセル値は、それに対応するマスク値が 0 でない場合にのみ加算される。マスク値が 0 の場合は、アキュムレータの画像のピクセル値は変更されない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタが NULL の場合のエラーを示す。
ippStsSizeErr	*roiSize.width* または *roiSize.height* の値が負の場合のエラーを示す。
ippStsStepErr	*srcStep*、*maskStep*、または *srcDstStep* が *roiSize.width * <pixelSize>* より小さいか、あるいは浮動小数点画像のステップ数が 4 で割り切れない場合のエラーを示す。

AddProduct

2 つのソース・イメージのピクセル値の積を求め、アキュムレータの画像の浮動小数点ピクセル値に加算する。

事例 1：インプレース操作

```
IppStatus ippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1, int
      src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step,
      Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u32f_C1IR　　8s32f_C1IR　　32f_C1IR

事例 2：マスクを使用したインプレース操作

```
IppStatus ippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1,
      int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step,
      const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst,
      int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u32f_C1IMR　　8s32f_C1IMR　　32f_C1IMR

引数

pSrc1, pSrc2	ソース・イメージへのポインタ
src1Step, src2Step	ソース・イメージ内のステップ（バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
pSrcDst	デスティネーション（アキュムレータ）イメージへのポインタ
srcDstStep	アキュムレータの画像内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）

説明

関数 ippiAddProduct は、ippcv.h ファイルの中で宣言される。この関数は、次のように、2 つのソース・イメージ *pSrc1* および *pSrc2* のピクセル値の積を求め、アキュムレータの画像 *pSrcDst* の浮動小数点ピクセル値に加算する。

*pSrcDst(x,y) = pSrcDst(x,y) + pSrc1(x,y) * pSrc2(x,y)*

マスク演算の場合、ピクセル値の積は、それに対応するマスク値が 0 でない場合にのみ加算される。マスク値が 0 の場合は、アキュムレータの画像のピクセル値は変更されない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタが NULL の場合のエラーを示す。
ippStsSizeErr	*roiSize.width* または *roiSize.height* の値が負の場合のエラーを示す。
ippStsStepErr	*src1Step*、*src2Step*、*maskStep*、または *srcDstStep* が *roiSize.width * <pixelSize>* より小さいか、あるいは浮動小数点画像のステップ数が 4 で割り切れない場合のエラーを示す。

AddWeighted

ソース・イメージのピクセル値に係数 α を掛けて、アキュムレータの画像の浮動小数点ピクセル値に係数（1-α）を掛けた値に加算する。

事例 1：インプレース操作

```
IppStatus ippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc,
      int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize
      roiSize,
      Ipp32f alpha);
```

サポートされる *mod* の値は、次のとおりである。

```
8u32f_C1IR      8s32f_C1IR      32f_C1IR
```

事例 2：マスクを使用したインプレース操作

```
IppStatus ippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc,
      int srcStep, const Ipp8u* pMask, int maskStep, Ipp32f*
      pSrcDst, int
      srcDstStep, IppiSize roiSize, Ipp32f alpha);
```

サポートされる *mod* の値は、次のとおりである。

```
8u32f_C1IMR     8s32f_C1IMR     32f_C1IMR
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ （バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
pSrcDst	デスティネーション（アキュムレータ）イメージへのポインタ
srcDstStep	アキュムレータの画像内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）
alpha	ソース・イメージの重み α

説明

関数 ippiAddWeighted は、ippcv.h ファイルの中で宣言される。この関数は、次のように、ソース・イメージ *pSrc* のピクセル値に重み係数 α を掛けて、アキュムレータの画像 *pSrcDst* の浮動小数点ピクセル値に係数（1-α）を掛けた値に加算する。

$$pSrcDst(x,y) = pSrcDst(x,y) * (1-\alpha) + pSrc(x,y) * \alpha$$

マスク演算の場合、重み付けされたピクセル値は、それに対応するマスク値が 0 でない場合にのみ加算される。マスク値が 0 の場合は、アキュムレータの画像のピクセル値は変更されない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタが NULL の場合のエラーを示す。
ippStsSizeErr	*roiSize.width* または *roiSize.height* の値が負の場合のエラーを示す。
ippStsStepErr	*srcStep*、*maskStep*、または *srcDstStep* が *roiSize.width * <pixelSize>* より小さいか、あるいは浮動小数点画像のステップ数が 4 で割り切れない場合のエラーを示す。

Mul

2 つのイメージ・バッファのピクセル値の
積を求める。

事例 1：整数データまたは複素数データに対する非インプレース操作

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
      const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
      pDst,
      int dstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1RSfs	16s_C1RSfs	16sc_C1RSfs	32sc_C1RSfs
8u_C3RSfs	16s_C3RSfs	16sc_C3RSfs	32sc_C3RSfs
8u_AC4RSfs	16s_AC4RSfs	16sc_AC4RSfs	32sc_AC4RSfs
8u_C4RSfs	16s_C4RSfs		

事例 2：浮動小数点データまたは複素数データに対する非インプレース操作

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
     const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
     pDst, int
     dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R    32fc_C1R
32f_C3R    32fc_C3R
32f_AC4R   32fc_AC4R
32f_C3R
```

事例 3：整数データまたは複素数データに対するインプレース操作

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
     int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IRSfs     16s_C1IRSfs    16sc_C1IRSfs   32sc_C1IRSfs
8u_C3IRSfs     16s_C3IRSfs    16sc_C3IRSfs   32sc_C3IRSfs
8u_AC4IRSfs    16s_AC4IRSfs   16sc_AC4IRSfs  32sc_AC4IRSfs
8u_C4IRSfs     16s_C4IRSfs
```

事例 4：浮動小数点データまたは複素数データに対するインプレース操作

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1IR       32fc_C1IR
32f_C3IR       32fc_C3IR
32f_AC4IR      32fc_AC4IR
32f_C4IR
```

引数

pSrc, pSrc1, pSrc2	ソース・バッファへのポインタ
srcStep, src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
dstSrcStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
scaleFactor	整数結果のスケーリングに使用される係数

説明

関数 ippiMul は、ippi.h ファイルの中で宣言される。この関数は、2 つのソース・イメージ・バッファの対応するピクセル値の積を求め、その結果をデスティネーション・バッファに格納する。整数データの演算の場合、得られた値は *scaleFactor* によってスケーリングされる。複素数データの場合は、ピクセル値の実数部と虚数部の両方が処理される。
AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

MulC

イメージ・バッファのピクセル値に定数を
掛ける。

事例 1：1 チャネル整数データまたは複素数データに対する非インプレース操作

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs      16s_C1RSfs     16sc_C1RSfs    32sc_C1RSfs
```

事例 2：マルチチャネル整数データまたは複素数データに対する非インプレース操作

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp<datatype> value[3], Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3RSfs      16s_C3RSfs     16sc_C3RSfs    32sc_C3RSfs
8u_AC4RSfs     16s_AC4RSfs    16sc_AC4RSfs   32sc_AC4RSfs
```

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp<datatype> value[4], Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4RSfs      16s_C4RSfs
```

事例 3：1 チャネル浮動小数点データまたは複素数データに対する非インプレース操作

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
      IppiSize
      roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R        32fc_C1R
```

事例 4：マルチチャネル浮動小数点データまたは複素数データに対する非インプレース操作

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int
     srcStep, const
     Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
     IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R        32fc_C3R
32f_AC4R       32fc_AC4R
```

```
IppStatus ippiMulC_32f_C4R(const Ipp32f* pSrc, int srcStep, const
     Ipp32f value[4], Ipp32f* pDst, int dstStep, IppiSize
     roiSize);
```

事例 5:1 チャネル整数データまたは複素数データに対するインプレース操作

```
IppStatus ippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>*
     pSrcDst,
     int srcDstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IRSfs     16s_C1IRSfs    16sc_C1IRSfs   32sc_C1IRSfs
```

事例 6：マルチチャネル整数データまたは複素数データに対するインプレース操作

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3],
     Ipp<datatype>*
     pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3IRSfs     16s_C3IRSfs    16sc_C3IRSfs   32sc_C3IRSfs
8u_AC4IRSfs    16s_AC4IRSfs   16sc_AC4IRSfs  32sc_AC4IRSfs
```

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[4],
     Ipp<datatype>*
     pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4IRSfs     16s_C4IRSfs
```

事例 7：1 チャネル浮動小数点データまたは複素数データに対するインプレース操作

```
IppStatus ippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>*
      pSrcDst,
      int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1IR        32fc_C1IR
```

事例 8：マルチチャネル浮動小数点データまたは複素数データに対するインプレース操作

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3],
      Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3IR        32fc_C3IR
32f_AC4IR       32fc_AC4IR
```

```
IppStatus ippiMulC_32f_C4IR(const Ipp32f value[4],
      Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
value	画像のピクセル値に加算される定数値（マルチチャネル・イメージの場合は定数ベクトル）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
scaleFactor	整数結果のスケーリングに使用される係数

説明

関数 ippiMulC は、ippi.h ファイルの中で宣言される。この関数は、画像のピクセル値に定数 *value* を掛ける。マルチチャネル・イメージの場合は、ピクセルの各チャネルの値に定数ベクトル *value* の成分を掛ける。複素数データの場合は、ピクセル値の実数部と虚数部の両方が処理される。
整数データの演算の場合、得られた値は *scaleFactor* によってスケーリングされる。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

MulScale

2 つのイメージ・バッファのピクセル値の
積を求め、その積をスケーリングする。

事例 1：非インプレース操作

```
IppStatus ippiMulScale_<mod>(const Ipp<datatype>* pSrc1, int
      src1Step,
      const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
      pDst,
      int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R
8u_C3R	16u_C3R
8u_C4R	16u_C4R
8u_AC4R	16u_AC4R

事例 2：インプレース操作

```
IppStatus ippiMulScale_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR	16u_C1IR
8u_C3IR	16u_C3IR
8u_C4IR	16u_C4IR
8u_AC4IR	16u_AC4IR

引数

pSrc, pSrc1, pSrc2	ソース・バッファへのポインタ
srcStep, src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiMulScale は、ippi.h ファイルの中で宣言される。この関数は、次の公式を使用して、2 つの入力バッファの対応するピクセル値の積を求め、その積をスケーリングする。

```
dst_pixel = src1_pixel * src2_pixel / max_val
```

ここで、src1_pixel と src2_pixel はソース・バッファのピクセル値、dst_pixel は結果のピクセル値、max_val はピクセルのデータ範囲の最大値である（詳細は、第 2 章の表 2-2 を参照）。
この関数は、8 ビットおよび 16 ビットの符号なしデータ・タイプのみをサポートしている。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

MulCScale

イメージ・バッファのピクセル値に定数を掛けて、その積をスケーリングする。

事例 1：1 チャネル・データに対する非インプレース操作

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
      IppiSize
      roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1R          16u_C1R
```

事例 2：マルチチャネル・データに対する非インプレース操作

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      const Ipp<datatype> value[3],  Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C3R          16u_C3R
      8u_AC4R         16u_AC4R
```

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      const Ipp<datatype> value[4],  Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C4R          16u_C4R
```

事例 3：1 チャネル・データに対するインプレース操作

```
IppStatus ippiMulCScale_<mod>(Ipp<datatype> value, const
      Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1IR         16u_C1IR
```

事例 4：マルチチャネル・データに対するインプレース操作

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype> value[3], const
      Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C3IR         16u_C3IR
      8u_AC4IR        16u_AC4IR
```

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype> value[4], const
      Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C4IR         16u_C4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
value	ソース・バッファ内の各ピクセル値に掛けられる定数値（3 チャネルまたは 4 チャネル・イメージの場合は定数ベクトル）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ

srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiMulCScale は、ippi.h ファイルの中で宣言される。この関数は、次の公式を使用して、入力バッファ内のピクセル値に定数 *value* を掛けて、その積をスケーリングする。

```
dst_pixel = src_pixel * value / max_val
```

ここで、src_pixel はソース・バッファ内のピクセル値、dst_pixel は結果のピクセル値、max_val はピクセルのデータ範囲の最大値である（詳細は、第 2 章の 表 2-2 を参照）。
この関数は、8 ビットおよび 16 ビットの符号なしデータ・タイプのみをサポートしている。この関数を使用して、0 ～ 1 の範囲内の数値でピクセル値を乗算できる。
AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

Sub

2 つのバッファのピクセル値の差を求める。

事例 1：整数データまたは複素数データに対する非インプレース操作

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
      const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
      pDst,
      int dstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs    16s_C1RSfs    16sc_C1RSfs    32sc_C1RSfs
8u_C3RSfs    16s_C3RSfs    16sc_C3RSfs    32sc_C3RSfs
8u_AC4RSfs   16s_AC4RSfs   16sc_AC4RSfs   32sc_AC4RSfs
8u_C4RSfs    16s_C4RSfs
```

事例 2：浮動小数点データまたは複素数データに対する非インプレース操作

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
      const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
      pDst,
      int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R    32fc_C1R
32f_C3R    32fc_C3R
32f_AC4R   32fc_AC4R
32f_C4R
```

事例 3：整数データまたは複素数データに対するインプレース操作

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
      int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IRSfs    16s_C1IRSfs    16sc_C1IRSfs    32sc_C1IRSfs
8u_C3IRSfs    16s_C3IRSfs    16sc_C3IRSfs    32sc_C3IRSfs
8u_AC4IRSfs   16s_AC4IRSfs   16sc_AC4IRSfs   32sc_AC4IRSfs
8u_C4IRSfs    16s_C4IRSfs
```

事例 4：浮動小数点データまたは複素数データに対するインプレース操作

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1IR	32fc_C1IR
32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR
32f_C4IR	

引数

pSrc, pSrc1, pSrc2	ソース・バッファへのポインタ
srcStep, src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
dstSrcStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
scaleFactor	整数結果のスケーリングに使用される係数

説明

関数 ippiSub は、ippi.h ファイルの中で宣言される。この関数は、ソース・バッファ *pSrc1* のピクセル値をバッファ *pSrc2* の対応するピクセル値から引き、その結果をデスティネーション・バッファ *pDst* に格納する。インプレース操作の場合は、*pSrc* 内の値を *pSrcDst* 内の値から引き、その結果を *pSrcDst* に格納する。複素数データの場合は、ピクセル値の実数部と虚数部の両方が処理される。
整数データの演算の場合、得られた値は *scaleFactor* によってスケーリングされる。アルファ・AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

SubC

イメージ・バッファのピクセル値から定数を引く。

事例 1：1 チャネル整数データまたは複素数データに対する非インプレース操作

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1RSfs	16s_C1RSfs	16sc_C1RSfs	32sc_C1RSfs

事例 2：マルチチャネル整数データまたは複素数データに対する非インプレース操作

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp<datatype> value[3], Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3RSfs	16s_C3RSfs	16sc_C3RSfs	32sc_C3RSfs
8u_AC4RSfs	16s_AC4RSfs	16sc_AC4RSfs	32sc_AC4RSfs

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp<datatype> value[4], Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4RSfs	16s_C4RSfs

事例 3：1 チャネル浮動小数点データまたは複素数データに対する非インプレース操作

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
      IppiSize
      roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
      32f_C1R         32fc_C1R
```

事例 4：マルチチャネル浮動小数点データまたは複素数データに対する非インプレース操作

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int
      srcStep, const
      Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
      32f_C3R         32fc_C3R
      32f_AC4R        32fc_AC4R
```

```
IppStatus ippiSubC_32f_C4R(const Ipp32f* pSrc, int srcStep, const
      Ipp32f value[4], Ipp32f* pDst, int dstStep, IppiSize
      roiSize);
```

事例5:1チャネル整数データまたは複素数データに対するインプレース操作

```
IppStatus ippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>*
      pSrcDst,
      int srcDstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1IRSfs      16s_C1IRSfs     16sc_C1IRSfs    32sc_C1IRSfs
```

事例 6：マルチチャネル整数データまたは複素数データに対するインプレース操作

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3],
      Ipp<datatype>*
      pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C3IRSfs      16s_C3IRSfs     16sc_C3IRSfs    32sc_C3IRSfs
      8u_AC4IRSfs     16s_AC4IRSfs    16sc_AC4IRSfs   32sc_AC4IRSfs
```

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[4],
      Ipp<datatype>*
      pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4IRSfs      16s_C4IRSfs
```

事例 7：1 チャネル浮動小数点データまたは複素数データに対するインプレース操作

```
IppStatus ippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>*
      pSrcDst,
      int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1IR        32fc_C1IR
```

事例 8：マルチチャネル浮動小数点データまたは複素数データに対するインプレース操作

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3],
      Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3IR        32fc_C3IR
32f_AC4IR       32fc_AC4IR
```

```
IppStatus ippiSubC_32f_C4IR(const Ipp32f> value[4],
      Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
value	ソース・バッファ内の各ピクセル値から引かれる定数値（マルチチャネル・イメージの場合は定数ベクトル）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）

roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
scaleFactor	整数結果のスケーリングに使用される係数

説明

関数 ippiSubC は、ippi.h ファイルの中で宣言される。この関数は、イメージ・バッファのピクセル値から定数 *value* を引くことによって、画像の輝度を変更する。マルチチャネル・イメージの場合は、ピクセルの各チャネルの値から定数ベクトル *value* の成分を引く。複素数データの場合は、ピクセル値の実数部と虚数部の両方が処理される。
整数データの演算の場合、得られた値は *scaleFactor* によってスケーリングされる。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

Div

イメージ・バッファのピクセル値を他のバッファのピクセル値で割る。

事例 1：整数データまたは複素数データに対する非インプレース操作

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
      const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
      pDst,
      int dstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1RSfs	16s_C1RSfs	16sc_C1RSfs	32sc_C1RSfs
8u_C3RSfs	16s_C3RSfs	16sc_C3RSfs	32sc_C3RSfs
		16sc_AC4RSfs	32sc_AC4RSfs

事例 2：浮動小数点データまたは複素数データに対する非インプレース操作

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
     const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
     pDst,
     int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1R	32fc_C1R
32f_C3R	32fc_C3R
32f_AC4R	32fc_AC4R
32f_C4R	

事例 3：整数データまたは複素数データに対するインプレース操作

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
     int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IRSfs	16s_C1IRSfs	16sc_C1IRSfs	32sc_C1IRSfs
8u_C3IRSfs	16s_C3IRSfs	16sc_C3IRSfs	32sc_C3IRSfs
		16sc_AC4IRSfs	32sc_AC4IRSfs

事例 4：浮動小数点データまたは複素数データに対するインプレース操作

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1IR	32fc_C1IR
32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR
32f_C4IR	

引数

pSrc, pSrc1, pSrc2	ソース・バッファへのポインタ
srcStep, src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
scaleFactor	整数結果のスケーリングに使用される係数

説明

関数 ippiDiv は、ippi.h ファイルの中で宣言される。この関数は、ソース・バッファ *pSrc2* のピクセル値をバッファ *pSrc1* の対応するピクセル値で割り、その結果をデスティネーション・バッファ *pDst* に格納する。インプレース操作の場合は、*pSrcDst* 内の値を *pSrc* 内の値で割り、その結果を *pSrcDst* に格納する。複素数データの場合は、ピクセル値の実数部と虚数部の両方が処理される。整数データの演算の場合、得られた値は *scaleFactor* によってスケーリングされる。
0 の除数を検出すると、関数の実行は中止される。

被除数の値に基づいて、0 による除算の結果は、次の表のように設定される。

表 5-2　0 による除算の結果の値

データ・タイプ	割り当てられる結果の値		
	0 の被除数	正の被除数	負の被除数
8u	0	IPP_MAX_8U	IPP_MIN_8U
16s, 16sc	0	IPP_MAX_16S	IPP_MIN_16S
32sc	0	IPP_MAX_32S	IPP_MIN_32S
32f, 32fc	QNAN	+INF	-INF

使用される定数の定義については、第 2 章の「画像のデータ・タイプと範囲」を参照のこと。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

次のコード例は、ippiDiv 関数の処理を示している。

例 5-1　ピクセル値の除算

```
IppStatus div32f( void ) {
   Ipp32f a[4*3], b[4*3];
   IppiSize roi = {2,2};
   int i;
   for( i=0; i<4*3; ++i ) a[i] = b[i] = (float)i;
   return ippiDiv_32f_C1IR( a, 4*4, b, 4*4, roi );
}
```

デスティネーション・イメージ *b* には、以下のデータが含まれる。

```
-1.#IND  +1.000  +2.000  +3.000
+1.000   +1.000  +6.000  +7.000
+8.000   +9.000  +10.00  +11.00
```

関数を呼び出したときのコンソール出力は、次のようになる。

```
-- warning in div32f: (6) Zero value(s) of divisor in the
function Div
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。
ippStsDivByZero	除数の値が 0 の場合の警告を示す。関数の実行は続けられる。

DivC

イメージ・バッファのピクセル値を定数で
割る。

事例 1：1 チャネル整数データまたは複素数データに対する非インプレース操作

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs      16s_C1RSfs      16sc_C1RSfs     32sc_C1RSfs
```

事例 2：マルチチャネル整数データまたは複素数データに対する非インプレース操作

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp<datatype> value[3], Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3RSfs      16s_C3RSfs      16sc_C3RSfs     32sc_C3RSfs
8u_AC4RSfs     16s_AC4RSfs     16sc_AC4RSfs    32sc_AC4RSfs
```

事例 3：1 チャネル浮動小数点データまたは複素数データに対する非インプレース操作

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
      IppiSize
      roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R        32fc_C1R
```

事例 4：マルチチャネル浮動小数点データまたは複素数データに対する非インプレース操作

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int
      srcStep, const
      Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

32f_C3R	32fc_C3R
32f_AC4R	32fc_AC4R

事例 5:1 チャネル整数データまたは複素数データに対するインプレース操作

```
IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>*
     pSrcDst,
     int srcDstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IRSfs	16s_C1IRSfs	16sc_C1IRSfs	32sc_C1IRSfs

事例 6：マルチチャネル整数データまたは複素数データに対するインプレース操作

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3],
     Ipp<datatype>*
     pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3IRSfs	16s_C3IRSfs	16sc_C3IRSfs	32sc_C3IRSfs
8u_AC4IRSfs	16s_AC4IRSfs	16sc_AC4IRSfs	32sc_AC4IRSfs

事例 7：1 チャネル浮動小数点データまたは複素数データに対するインプレース操作

```
IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>*
     pSrcDst,
     int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1IR	32fc_C1IR

事例 8：マルチチャネル浮動小数点データまたは複素数データに対するインプレース操作

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3],
     Ipp<datatype>*
     pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
value	ソース・バッファ内の各ピクセル値を割る定数値（3 チャネルまたは 4 チャネル・イメージの場合は定数ベクトル）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
scaleFactor	整数結果のスケーリングに使用される係数

説明

関数 ippiDivC は、ippi.h ファイルの中で宣言される。この関数は、イメージ・バッファのピクセル値を定数 *value* で割ることによって、画像の輝度を変更する。マルチチャネル・イメージの場合は、ピクセルの各チャネルの値を定数ベクトル *value*[3] の成分で割る。複素数データの場合は、ピクセル値の実数部と虚数部の両方が処理される。整数データの演算の場合、得られた値は *scaleFactor* によってスケーリングされる。

除数が 0 の場合は、関数の実行は中止され、ippStsDivByZeroErr エラー・ステータスが設定される。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。

ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsDivByZeroErr	除数の値が 0 の場合のエラー状態を示す。

Abs

ソース・イメージのピクセルの絶対値を計算し、デスティネーション・イメージに格納する。

事例 1：非インプレース操作

```
IppStatus ippiAbs_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

16s_C1R	32f_C1R
16s_C3R	32f_C3R
16s_C4R	32f_C4R
16s_AC4R	32f_AC4R

事例 2：インプレース操作

```
IppStatus ippiAbs_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
     IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

16s_C1IR	32f_C1IR
16s_C3IR	32f_C3IR
16s_C4IR	32f_C4IR
16s_AC4IR	32f_AC4IR

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ

dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiAbs は、ippi.h ファイルの中で宣言される。この関数は、ソース・バッファの各ピクセル内にある各チャネルの絶対値を計算し、その結果をデスティネーション・バッファに格納する。この関数は、符号付きデータのみを処理する。AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。

AbsDiff

2 つの画像の差の絶対値を計算する。

```
IppStatus ippiAbsDiff_8u_C1R(const Ipp8u* pSrc1, int src1Step,
      const Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep,
      IppiSize roiSize);
IppStatus ippiAbsDiff_32f_C1R(const Ipp32f* pSrc1, int src1Step,
      const Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep,
      IppiSize roiSize);
```

引数

pSrc1	第 1 のソース・イメージへのポインタ

src1Step	第 1 のソース・イメージ内のステップ（バイト単位）
pSrc2	第 2 のソース・イメージへのポインタ
src2Step	第 2 のソース・イメージ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）

説明

関数 ippiAbsDiff は、ippi.h ファイルの中で宣言される。この関数は、次の公式によって、2 つの画像の差の絶対値をピクセル単位で計算する。

$$pDst(x,y) = \mathrm{abs}(pSrc1(x,y) - pSrc2(x,y))$$

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタが NULL の場合のエラーを示す。
ippStsSizeErr	*roiSize.width* または *roiSize.height* の値が負の場合のエラーを示す。
ippStsStepErr	*src1Step*、*src2Step*、または *dstStep* が *roiSize.width * <pixelSize>* より小さいか、あるいは浮動小数点画像のステップ数が 4 で割り切れない場合のエラーを示す。

AbsDiffC

画像とスカラ値の差の絶対値を計算する。

```
IppStatus ippiAbsDiffC_8u_C1R(const Ipp8u* pSrc, int srcStep,
     Ipp8u* pDst, int dstStep, IppiSize roiSize, int value);
IppStatus ippiAbsDiffC_32f_C1R(const Ipp32f* pSrc, int srcStep,
     Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f value);
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）

pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）
value	ソース・イメージの各要素の減分に使用されるスカラ値

説明

関数 ippiAbsDiffC は、ippi.cv ファイルの中で宣言される。この関数は、次の公式によって、ソース・イメージ *pSrc* とスカラ値 *value* の差の絶対値をピクセル単位で計算する。

$$pDst(x, y) = \text{abs}(pSrc(x, y) - value)$$

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタが NULL の場合のエラーを示す。
ippStsSizeErr	*roiSize.width* または *roiSize.height* の値が負の場合のエラーを示す。
ippStsStepErr	*srcStep* または *dstStep* が *roiSize.width* * *<pixelSize>* より小さいか、あるいは浮動小数点画像のステップ数が 4 で割り切れない場合のエラーを示す。

Sqr

画像のピクセル値を 2 乗し、デスティネーション・イメージに書き込む。

事例 1：整数データに対する非インプレース操作

```
IppStatus ippiSqr_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int
      scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs

事例 2：浮動小数点データに対する非インプレース操作

```
IppStatus ippiSqr_<mod>(const Ipp32f* pSrc, int srcStep,
      Ipp32f* pDst,
      int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

事例 3：整数データに対するインプレース操作

```
IppStatus ippiSqr_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4RISfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs

事例 4：浮動小数点データに対するインプレース操作

```
IppStatus ippiSqr_<mod>(Ipp32f* pSrcDst, int srcDstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ

dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
scaleFactor	整数結果のスケーリングに使用される係数

説明

関数 ippiSqr は、ippi.h ファイルの中で宣言される。この関数は、ソース・バッファのピクセル値を 2 乗し、デスティネーション・バッファに書き込む。整数データを処理する関数タイプは、内部で計算された値に対して *scaleFactor* で定義される固定スケーリングを適用し、その結果を飽和処理してからデスティネーション・バッファに書き込む。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。

Sqrt

ソース・イメージのピクセル値の平方根を計算し、デスティネーション・イメージに書き込む。

事例 1：整数データに対する非インプレース操作

```
IppStatus ippiSqrt_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int
      scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs       16u_C1RSfs      16s_C1RSfs
8u_C3RSfs       16u_C3RSfs      16s_C3RSfs
8u_AC4RSfs      16u_AC4RSfs     16s_AC4RSfs
```

事例 2：浮動小数点データに対する非インプレース操作

```
IppStatus ippiSqrt_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f*
      pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R
32f_C3R
32f_AC4R
```

事例 3：整数データに対するインプレース操作

```
IppStatus ippiSqrt_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IRSfs      16u_C1IRSfs     16s_C1IRSfs
8u_C3IRSfs      16u_C3IRSfs     16s_C3IRSfs
8u_AC4IRSfs     16u_AC4IRSfs    16s_AC4IRSfs
```

事例 4：浮動小数点データに対するインプレース操作

```
IppStatus ippiSqrt_<mod>(Ipp32f* pSrcDst, int srcDstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1IR
32f_C3IR
```

32f_C4IR
32f_AC4IR

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
scaleFactor	整数結果のスケーリングに使用される係数

説明

関数 ippiSqrt は、ippi.h ファイルの中で宣言される。この関数は、ソース・バッファのピクセル値の平方根を計算し、デスティネーション・バッファに書き込む。整数データを処理する関数タイプは、内部で計算された値に対して *scaleFactor* で定義される固定スケーリングを適用し、その結果を飽和処理してからデスティネーション・バッファに書き込む。負のソース・ピクセル値が検出された場合は、警告が発行され、関数の実行は続けられる。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsSqrtNegArg	ソース・ピクセルの値が負であるという警告を示す。

Ln

ソース・イメージ内のピクセル値の自然対数関数を計算し、その結果をデスティネーション・イメージに書き込む。

事例 1：整数データに対する非インプレース操作

```
IppStatus ippiLn_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int
      scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs       16s_C1RSfs
8u_C3RSfs       16s_C3RSfs
```

事例 2：浮動小数点データに対する非インプレース操作

```
IppStatus ippiLn_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f*
      pDst,
      int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R
32f_C3R
```

事例 3：整数データに対するインプレース操作

```
IppStatus ippiLn_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IRSfs      16s_C1IRSfs
8u_C3IRSfs      16s_C3IRSfs
```

事例 4：浮動小数点データに対するインプレース操作

```
IppStatus ippiLn_<mod>(Ipp32f* pSrcDst, int srcDstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1IR
32f_C3IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
scaleFactor	整数結果のスケーリングに使用される係数

説明

関数 ippiLn は、ippi.h ファイルの中で宣言される。この関数は、ソース・バッファのピクセル値の自然対数関数を計算し、得られた値をデスティネーション・バッファに書き込む。整数データを処理する関数タイプは、内部で計算された値に対して *scaleFactor* で定義される固定スケーリングを適用し、その結果を飽和処理してからデスティネーション・バッファに書き込む。

0 または負のソース・ピクセル値が検出された場合は、それに対応する警告が発行され、関数の実行は続けられる。
浮動小数点入力データを処理する場合、ソース値が 0 または負の場合の関数の実行結果は、`Ln(0)= - inf`、`Ln(x<0) = NaN` になる。

複数の入力値が 0 または負の場合は、次のコード例に示すように、関数によって返されるステータス・コードは、最初に検出された値に対応するものになる。

例 5-2　対数関数の使用

```
IppStatus ln( void ) {
   Ipp32f img[8*8];
   IppiSize roi = { 8, 8 };
   IppStatus st;
   ippiSet_32f_C1R( (float)IPP_E, img, 8*4, roi );
   img[0] = -0;
   img[1] = -1;
   st = ippiLn_32f_C1IR( img, 8*4, roi );
   printf( "%f %f %f\n", img[0], img[1], img[2] );
   return st;
}
出力値 :
-1.#INF00 -1.#IND00 1.000000
ステータス値とメッセージ :
(7) Zero value(s) of argument in the Ln function
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsLnZeroArg	ソース・ピクセルの値が 0 であるという警告を示す。
ippStsLnNegArg	ソース・ピクセルの値が負であるという警告を示す。

Exp

ソース・イメージ内のピクセル値の指数関数を計算し、その結果をデスティネーション・イメージに書き込む。

事例 1：整数データに対する非インプレース操作

```
IppStatus ippiExp_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int
      scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs       16s_C1RSfs
8u_C3RSfs       16s_C3RSfs
```

事例 2：浮動小数点データに対する非インプレース操作

```
IppStatus ippiExp_<mod>(const Ipp32f* pSrc, int srcStep,
      Ipp32f* pDst,
      int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R
32f_C3R
```

事例 3：整数データに対するインプレース操作

```
IppStatus ippiExp_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
      IppiSize roiSize, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IRSfs      16s_C1IRSfs
8u_C3IRSfs      16s_C3IRSfs
```

事例 4：浮動小数点データに対するインプレース操作

```
IppStatus ippiExp_<mod>(Ipp32f* pSrcDst, int srcDstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1IR
32f_C3IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
scaleFactor	整数結果のスケーリングに使用される係数

説明

関数 ippiExp は、ippi.h ファイルの中で宣言される。この関数は、e をソース・バッファのピクセル値で累乗した値を計算し、得られた値をデスティネーション・バッファに書き込む。整数データを処理する関数タイプは、内部で計算された値に対して *scaleFactor* で定義される固定スケーリングを適用し、その結果を飽和処理してからデスティネーション・バッファに書き込む。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

Complement

負の数値を補数から直接コードに変換する。

```
IppStatus ippiComplement_32s_C1IR(Ipp32s* pSrcDst, int
      srcDstStep,
      IppiSize roiSize);
```

引数

pSrcDst	ソース・バッファとデスティネーション・バッファへのポインタ
srcDstStep	ソース・イメージ・バッファとデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）.

説明

関数 ippiComplement_32s_C1IR は、ippi.h ファイルの中で宣言される。この関数は、負の整数を補数から直接コードに変換する。最上位ビットの符号は維持される。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcDstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsStrideErr	*srcDstStep* が画像の幅より小さい場合のエラー状態を示す。

論理演算

本節で説明する関数は、ピクセル値のビット単位の論理演算を実行する。論理演算には、AND （論理積）、NOT（論理否定）、OR（論理和）、XOR（排他的論理和）、ビット・シフトがある。

And

2 つのソース・バッファの対応するピクセルの間で、ビット単位の AND 演算を実行する。

事例 1：非インプレース操作

```
IppStatus ippiAnd_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
      const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
      pDst,
      int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

事例 2：インプレース操作

```
IppStatus ippiAnd_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

引数

pSrc, pSrc1, pSrc2	ソース・バッファへのポインタ
srcStep, src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiAnd は、ippi.h ファイルの中で宣言される。この関数は、2 つの入力バッファの対応するピクセル値の間でビット単位の AND 演算を実行し、その結果をデスティネーション・バッファに書き込む。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

AndC

各ピクセルと定数の間で、ビット単位の
AND 演算を実行する。

事例 1：1 チャネル・データに対する非インプレース操作

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
      IppiSize
      roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R	32s_C1R

事例 2：マルチチャネル・データに対する非インプレース操作

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp<datatype> value[3], Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16u_C3R	32s_C3R
8u_AC4R	16u_AC4R	32s_AC4R

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp<datatype> value[4], Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4R	16u_C4R	32s_C4R

事例 3：1 チャネル・データに対するインプレース操作

```
IppStatus ippiAndC_<mod>(Ipp<datatype> value, const Ipp<datatype>*
      pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR	16u_C1IR	32s_C1IR

事例 4：マルチチャネル・データに対するインプレース操作

```
IppStatus ippiAndC_<mod>(const Ipp<datatype> value[3], const
      Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3IR	16u_C3IR	32s_C3IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

```
IppStatus ippiAndC_<mod>(const Ipp<datatype> value[4], const
     Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_AC4IR	16u_AC4IR	32s_AC4IR

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
value	ソース・バッファの各ピクセルとのビット単位の AND 演算に使用される定数値 （マルチチャネル・イメージの場合は定数ベクトル）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiAndC は、ippi.h ファイルの中で宣言される。この関数は、ソース・バッファの各ピクセル値と定数 *value* の間で、ビット単位の AND 演算を実行する。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。

ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

Not

ソース・バッファの各ピクセルに対して、
ビット単位の NOT 演算を実行する。

事例 1：非インプレース操作

```
IppStatus ippiNot_<mod>(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R
8u_C3R
8u_C4R
8u_AC4R
```

事例 2：インプレース操作

```
IppStatus ippiNot_<mod>(Ipp8u* pSrcDst, int srcDstStep, IppiSize
      roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IR
8u_C3IR
8u_C4IR
8u_AC4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）

pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiNot は、ippi.h ファイルの中で宣言される。この関数は、ソース・バッファの各ピクセル値に対してビット単位の NOT 演算を実行する。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

Or

2 つのソース・バッファのピクセル値の間で、ビット単位の OR 演算を実行する。

事例 1：非インプレース操作

```
IppStatus ippiOr_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
      const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
      pDst,
      int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1R      16u_C1R      32s_C1R
```

8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

事例 2：インプレース操作

```
IppStatus ippiOr_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

引数

pSrc, pSrc1, pSrc2	ソース・バッファへのポインタ
srcStep, src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiOr は、ippi.h ファイルの中で宣言される。この関数は、2 つの入力バッファの対応するピクセル値の間でビット単位の OR 演算を実行し、その結果をデスティネーション・バッファに書き込む。
AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。

ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

OrC

バッファの各ピクセルと定数の間で、ビット単位の OR 演算を実行する。

事例 1：1 チャネル・データに対する非インプレース操作

```
IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R	32s_C1R

事例 2：マルチチャネル・データに対する非インプレース操作

```
IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp<datatype> value[3], Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16u_C3R	32s_C3R
8u_AC4R	16u_AC4R	32s_AC4R

```
IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp<datatype> value[4], Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4R	16u_C4R	32s_C4R

事例 3：1 チャネル・データに対するインプレース操作

```
IppStatus ippiOrC_<mod>(Ipp<datatype> value, const Ipp<datatype>*
      pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR	16u_C1IR	32s_C1IR

事例 4：マルチチャネル・データに対するインプレース操作

```
IppStatus ippiOrC_<mod>(const Ipp<datatype> value[3], const
     Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3IR	16u_C3IR	32s_C3IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

```
IppStatus ippiOrC_<mod>(const Ipp<datatype> value[4], const
     Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4IR	16u_C4IR	32s_C4IR

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
value	ソース・バッファの各ピクセルとのビット単位の OR 演算に使用される定数値（マルチチャネル・イメージの場合は定数ベクトル）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiOrC は、ippi.h ファイルの中で宣言される。この関数は、ソース・バッファの各ピクセル値と定数 *value* の間で、ビット単位の OR 演算を実行する。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

Xor

2 つのソース・バッファのピクセルの間で、
ビット単位の XOR 演算を実行する。

事例 1：非インプレース操作

```
IppStatus ippiXor_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
      const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
      pDst,
      int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

事例 2：インプレース操作

```
IppStatus ippiXor_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

引数

pSrc, pSrc1, pSrc2	ソース・バッファへのポインタ
srcStep, src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiXor は、ippi.h ファイルの中で宣言される。この関数は、2 つの入力バッファの対応するピクセル値の間でビット単位の XOR 演算を実行し、その結果をデスティネーション・バッファに書き込む。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

XorC

バッファの各ピクセルと定数の間で、ビット単位の XOR 演算を実行する。

事例 1：1 チャネル・データに対する非インプレース操作

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R	32s_C1R

事例 2：マルチチャネル・データに対する非インプレース操作

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp<datatype> value[3], Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16u_C3R	32s_C3R
8u_AC4R	16u_AC4R	32s_AC4R

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp<datatype> value[4], Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4R	16u_C4R	32s_C4R

事例 3：1 チャネル・データに対するインプレース操作

```
IppStatus ippiXorC_<mod>(Ipp<datatype> value, const Ipp<datatype>*
      pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR	16u_C1IR	32s_C1IR

事例 4：マルチチャネル・データに対するインプレース操作

```
IppStatus ippiXorC_<mod>(const Ipp<datatype> value[3], const
      Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3IR	16u_C3IR	32s_C3IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

```
IppStatus ippiXorC_<mod>(const Ipp<datatype> value[4], const
     Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4IR	16u_C4IR	32s_C4IR

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
value	ソース・バッファの各ピクセルとのビット単位の XOR 演算に使用される定数値 （マルチチャネル・イメージの場合は定数ベクトル）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiXorC は、ippi.h ファイルの中で宣言される。この関数は、ソース・バッファの各ピクセル値と定数 *value* の間で、ビット単位の XOR 演算を実行する。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。

ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

LShiftC

ピクセル値の各ビットを左にシフトする。

事例 1：1 チャネル・データに対する非インプレース操作

```
IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp32u value, Ipp<datatype>* pDst, int dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16u_C1R     32s_C1R
```

事例 2：マルチチャネル・データに対する非インプレース操作

```
IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      const Ipp32u value[3], Ipp<datatype>* pDst, int dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16u_C3R     32s_C3R
8u_AC4R     16u_AC4R    32s_AC4R
```

```
IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      const Ipp32u value[4], Ipp<datatype>* pDst, int dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16u_C4R     32s_C4R
```

事例 3：1 チャネル・データに対するインプレース操作

```
IppStatus ippiLShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IR     16u_C1IR     32s_C1IR
```

事例 4：マルチチャネル・データに対するインプレース操作

```
IppStatus ippiLShiftC_<mod>(const Ipp32u value[3],
     Ipp<datatype>*
     pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3IR     16u_C3IR     32s_C3IR
8u_AC4IR    16u_AC4IR    32s_AC4IR
```

```
IppStatus ippiLShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>*
     pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4IR     16u_C4IR     32s_C4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
value	シフトするビット数（マルチチャネル・イメージの場合は定数ベクトル）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiLShiftC は、ippi.h ファイルの中で宣言される。この関数は、各ピクセル値のビットを *value* ビット左にシフトすることによって、ソース・バッファ内のピクセルの輝度を変更する。マルチチャネル・データの場合は、各カラー・チャネルに独自のシフト値を指定できる。ビットのシフト後に空いた位置は、0 で埋められ

る。左シフト演算によって得られた値は、飽和処理されない。結果を飽和処理する必要がある場合は、シフト関数ではなく乗算関数を使用する。
AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

次のコード例は、左シフト関数の使用方法を示している。

例 5-3　ビットの左シフト

```
IppStatus lshift( void ) {
   Ipp8u img[8*8] = { 1, 0x7F, 0xFE };
   IppiSize roi = { 8, 8 };
   IppStatus st = ippiLShiftC_8u_C1IR( 1, img, 8, roi );
   printf( "%02x %02x %02x\n", img[0], img[1], img[2] );
   return st;
}
出力値 :
02 fe fc
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

RShiftC

ピクセル値の各ビットを右にシフトする。

事例 1：1 チャネル・データに対する非インプレース操作

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp32u value, Ipp<datatype>* pDst, int dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R　8s_C1R　16u_C1R　16s_C1R　32s_C1R

事例 2：マルチチャネル・データに対する非インプレース操作

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int
     srcStep,
     const Ipp32u value[3], Ipp<datatype>* pDst, int dstStep,
     IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R　8s_C3R　16u_C3R　16s_C3R　32s_C3R
8u_AC4R　8s_AC4R　16u_AC4R　16s_AC4R　32s_AC4R

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int
     srcStep,
     const Ipp32u value[4], Ipp<datatype>* pDst, int dstStep,
     IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4R　8s_C4R　16u_C4R　16s_C4R　32s_C4R

事例 3：1 チャネル・データに対するインプレース操作

```
IppStatus ippiRShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst,
     int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR　8s_C1IR　16u_C1IR　16s_C1IR　32s_C1IR

事例 4：マルチチャネル・データに対するインプレース操作

```
IppStatus ippiRShiftC_<mod>(const Ipp32u value[3], Ipp<datatype>*
     pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3IR　8s_C3IR　16u_C3IR　16s_C3IR　32s_C3IR
8u_AC4IR　8s_AC4IR　16u_AC4IR　16s_AC4IR　32s_AC4IR

```
IppStatus ippiRShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>*
     pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4IR　8s_C4IR　16u_C4IR　16s_C4IR　32s_C4IR

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
value	シフトするビット数（マルチチャネル・イメージの場合は定数ベクトル）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRShiftC は、ippi.h ファイルの中で宣言される。この関数は、各ピクセル値のビットを *value* ビット右にシフトすることによって、ソース・バッファ内のピクセルの輝度を下げる。ビットのシフト後に空いた位置は、符号ビットで埋められる。マルチチャネル・データの場合は、各カラー・チャネルに独自のシフト値を指定できる。この操作は、2 を定数で累乗した値でピクセル値を割る操作と同等である。

AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

アルファ合成

インテル IPP には、画像の透過度（アルファ）チャネルまたは指定されたアルファ値を使用して、2 つのイメージ・バッファを合成する関数が用意されている。

アルファ合成関数は、RGB または RGBA 形式の 8 ビットまたは 16 ビット・データを含むイメージ・バッファを操作する。すべての合成処理では、前景イメージ・バッファ *pSrc1* のピクセルを背景イメージ・バッファ *pSrc2* のピクセルの上に重ねることによって、デスティネーション・バッファ *pDst* 内の結果のピクセルが作成される。表 5-4 は、IPP でサポートされる、アルファ値を使用した画像合成のタイプを示している。

表 5-3　画像合成処理のタイプ

タイプ	カラー成分		
	出力ピクセル	アルファ値	画像処理用語による説明
OVER	α_A*A+(1-α_A)*α_B*B	α_A+(1-α_A)*α_B	A は B をさえぎる。
IN	α_A*A*α_B	α_A*α_B	B の中の A 。A は B のマットとして機能する。A は、B が見える個所でのみ見える。
OUT	α_A*A*(1-α_B)	α_A *(1-α_B)	B の外の A。NOT-B が A のマットとして機能する。A は、B が見えない個所でのみ見える。
ATOP	α_A*A*α_B+(1-α_A)*α_B*B	α_A*α_B+(1-α_A)*α_B	（A IN B）と（B OUT A）の組み合わせ。B は、A の背景とマットの両方になる。
XOR	α_A*A*(1-α_B)+(1-α_A)*α_B*B	α_A*(1-α_B)+(1-α_A)*α_B	（A OUT B）と （B OUT A）の組み合わせ。A と B は、互いに相手が見えない個所でのみ見える。
PLUS	α_A *A + α_B*B	α_A + α_B	優先順位なしのブレンディング

上の公式では、わかりやすいように、入力イメージ・バッファを A と B で示す。添字付きのギリシャ文字 α は、0 ～ 1 の範囲で正規化（スケーリング）されたアルファ値を示す。この値と整数のアルファ値 *alpha* の関係は、次の式で示される。

$$\alpha = alpha \ / \ max\_val$$

ここで、*max_val* は、8 ビット符号なしピクセル・データの場合は 255、16 ビット符号なしピクセル・データの場合は 65535 である。

4 チャネルの RGBA バッファのみを操作する ippiAlphaComp 関数の場合は、αA と αB は、それぞれ入力イメージ・バッファ A と B の正規化されたアルファ値である。
ippiAlphaCompC 関数の場合は、αA と αB は、パラメータとして関数に渡される、正規化された一定のアルファ値である。
ただし、結果のカラー・チャネル値を計算する公式の中では、A と B は、それぞれの入力イメージ・バッファのピクセル・カラー成分を示す。

一部のアルファ合成処理では、関数 ippiAlphaPremul, ippiAlphaPremulC を使用して、カラー・チャネル値とアルファ値の積を事前に求めることによって、計算の回数を大きく減らせる。この方法は、合成処理に必要な乗算の回数を減らせるため、画像を繰り返し合成する場合には特に効果的である。

ippiAlphaComp 関数と ippiAlphaCompC 関数で実行される合成処理のタイプは、*alphaType* パラメータによって指定される。このパラメータには、次の表に示した値を使用できる。

表 5-4　*alphaType* パラメータに使用できる値

処理のタイプ	パラメータの値	
OVER	ippAlphaOver	ippAlphaOverPremul
IN	ippAlphaIn	ippAlphaInPremul
OUT	ippAlphaOut	ippAlphaOutPremul
ATOP	ippAlphaATop	ippAlphaATopPremul
XOR	ippAlphaXor	ippAlphaXorPremul
PLUS	ippAlphaPlus	ippAlphaPlusPremul

AlphaComp

2 つの画像のアルファ（透過度）値を使用して、2 つのイメージを合成する。

```
IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* pSrc1,
     int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
     Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
     IppiAlphaType alphaType);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_AC1R     8u_AC4R
16u_AC1R    16u_AC4R
```

```
IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* const pSrc1[4],
      int src1Step, const Ipp<datatype>* const pSrc2[4], int
      src2Step,
      Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize,
      IppiAlphaType alphaType);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_AP4R      16u_AP4R
```

引数

pSrc1, pSrc2	（ピクセル順序データの場合）ソース・バッファへのポインタ。 （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ。 （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
alphaType	実行する合成のタイプ。各タイプの値と説明は、表 5-4 を参照のこと。

説明

関数 ippiAlphaComp は、ippi.h ファイルの中で宣言される。この関数は、2 つの画像のアルファ値を使用して、RGBA イメージ・バッファの画像合成処理を実行する。合成は、前景イメージ pSrc1 のピクセル（r_A, g_A, b_A, α_A）を、背景イメージ *pDst2* のピクセル（r_B, g_B, b_B, α_B）の上に重ねて、結果のイメージ *pDst* 内にピクセル（r_C, g_C, b_C, α_C）を生成することによって行われる。
アルファ値は、[0..1] の範囲に合わせて正規化されているものとする。
合成処理のタイプは、*alphaType* パラメータで指定される。
表 5-4 を使用して、必要な合成のタイプに基づいて有効な *alphaType* の値を選択できる。
例えば、OVER 演算の場合（表 5-3 を参照）、結果のピクセル・カラー成分は、次の式で計算される。

$$r_C = \alpha_A * r_A + (1 - \alpha_A) * \alpha_B * r_B$$

$g_C = \alpha_A * g_A + (1 - \alpha_A) * \alpha_B * g_B$

$b_C = \alpha_A * b_A + (1 - \alpha_A) * \alpha_B * b_B$

結果の（正規化された）アルファ値は、次の式で計算される。

$\alpha_C = \alpha_A + (1 - \alpha_A) * \alpha_B$

この関数は、符号なしピクセル・データにのみ使用できる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*src1Step*、*src2Step*、または *dstStep* の値が 0 または負の場合のエラー状態を示す。

AlphaCompC

一定のアルファ値を使用して、2 つのイメージを合成する。

```
IppStatus ippiAlphaCompC_<mod>(const Ipp<datatype>* pSrc1, int
      src1Step, Ipp<datatype> alpha1, const Ipp<datatype>*
      pSrc2, int
      src2Step, Ipp<datatype> alpha2, Ipp<datatype>* pDst,
      int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R
8u_C3R	16u_C3R
8u_C4R	16u_C4R
8u_AC4R	16u_AC4R

```
IppStatus ippiAlphaCompC_<mod>(const Ipp<datatype>* const
      pSrc1[4],
      int src1Step, Ipp<datatype> alpha1, const Ipp<datatype>*
      const
```

```
pSrc2[4], int src2Step, Ipp<datatype> alpha2, Ipp<datatype>*
const pDst[4], int dstStep, IppiSize roiSize, IppiAlphaType
alphaType);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_AP4R      16u_AP4R
```

引数

pSrc1, pSrc2	（ピクセル順序データの場合）ソース・バッファへのポインタ。 （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ。 （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
alpha1, alpha2	合成処理に使用される一定のアルファ値
alphaType	実行する合成のタイプ。使用できるタイプ値については、表 5-4 を参照のこと。

説明

関数 ippiAlphaCompC は、ippi.h ファイルの中で宣言される。この関数は、一定のアルファ値 *alpha1* および *alpha2* を使用して、1 チャネル・イメージ・バッファ、3 チャネル RGB/4 チャネル RGBA イメージ・バッファ、プレーン・イメージの画像合成処理を実行する。これらのアルファ値は、パラメータとして関数に渡される。
合成は、前景イメージ・バッファ *pSrc1* のピクセルを、背景イメージ・バッファ *pDst2* のピクセルの上に重ねて、結果のイメージ・バッファ *pDst* 内にピクセルを生成することによって行われる。
アルファ値は、[0..1] の範囲に合わせて正規化される。
合成処理のタイプは、*alphaType* 引数で指定される。表 5-4 を使用して、必要な合成のタイプに基づいて有効な *alphaType* の値を選択できる。
例えば、OVER 演算の場合（表 5-3 を参照）、結果のピクセル・カラー成分は、次の式で計算される。

$r_C = \alpha_1 * r_A + (1 - \alpha_1) * \alpha_2 * r_B$

$g_C = \alpha_1 * g_A + (1 - \alpha_1) * \alpha_2 * g_B$

$b_C = \alpha_1 * b_A + (1 - \alpha_1) * \alpha_2 * b_B$

ここで、α_1, α_2 は、正規化されたアルファ値 *alpha1*, *alpha2* である。

この関数は、符号なしピクセル・データにのみ使用できる。

次のコード例は、アルファ合成関数の使用方法を示している。

例 5-4　アルファ合成関数の使用

```
IppStatus acomp( void ) {
   Ipp8u imga[8*8], imgb[8*8], imgc[8*8];
   IppiSize roi = { 8, 8 };
   IppStatus st;
   ippiImageRamp_8u_C1R( imga, 8, roi, 0, 1, ippAxsHorizontal
);
   ippiImageRamp_8u_C1R( imgb, 8, roi, 0, 2, ippAxsHorizontal
);
   st = ippiAlphaCompC_8u_C1R( imga, 8, 255/3, imgb, 8, 255,
imgc, 8, roi, ippAlphaOver );
   printf( "over: a=%d,A=255/3; b=%d,B=255; c=%d //
c=a*A+b*(1-A)*B\n",imga[6],imgb[6],imgc[6] );
   return st;
}
出力
over: a=6,A=255/3; b=12,B=255; c=10 // c=a*A+b*B*(1-A)
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*src1Step*、*src2Step*、または *dstStep* の値が 0 または負の場合のエラー状態を示す。

AlphaPremul

イメージのピクセル値とアルファ値の積を
事前に求める。

事例 1：非インプレース操作

```
IppStatus ippiAlphaPremul_<mod>(const Ipp<datatype>* pSrc, int
      srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
      roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_AC4R     16u_AC4R
```

```
IppStatus ippiAlphaPremul_<mod>(const Ipp<datatype>* const
      pSrc[4],
      int srcStep, Ipp<datatype>* const pDst[4], int dstStep,
      IppiSize
      roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_AP4R     16u_AP4R
```

事例 2：インプレース操作

```
IppStatus ippiAlphaPremul_<mod>(Ipp<datatype>* pSrcDst, int
      srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_AC4IR    16u_AC4IR
```

```
IppStatus ippiAlphaPremul_<mod>(Ipp<datatype>* const
      pSrcDst[4], int
      srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_AP4IR    16u_AP4IR
```

引数

pSrc	（ピクセル順序データの場合）ソース・バッファへのポインタ。 （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列

srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ。 （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ、または個別のソースおよびデスティネーション・カラー・プレーンへのポインタの配列
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiAlphaPremul は、ippi.h ファイルの中で宣言される。この関数は、RGBA ソース・イメージ（ピクセル順序または平面）を、アルファ値を事前に乗算した形式に変換する。ピクセルの赤、緑、青、およびアルファの値が（r, g, b, a）である場合、この関数を実行すると、新しいピクセル値は（r* α , g* α , b* α , a）となる。ここで、 α は、0 ～ 1 の範囲で正規化されたピクセルのアルファ値である。

関数 ippiAlphaPremul は、符号なしピクセル・データにのみ使用できる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

AlphaPremulC

一定のアルファ（透過度）値を使用して、
イメージのピクセル値を事前に乗算する。

事例 1：非インプレース操作

```
IppStatus ippiAlphaPremulC_<mod>(const Ipp<datatype>* pSrc, int
      srcStep, Ipp<datatype> alpha, Ipp<datatype>* pDst, int
      dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R
8u_C3R	16u_C3R
8u_C4R	16u_C4R
8u_AC4R	16u_AC4R

```
IppStatus ippiAlphaPremulC_<mod>(const Ipp<datatype>* const
      pSrc[4],
      int srcStep, Ipp<datatype> alpha, Ipp<datatype>* const
      pDst[4],
      int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_AP4R	16u_AP4R

事例 2：インプレース操作

```
IppStatus ippiAlphaPremulC_<mod>(Ipp<datatype> alpha, Ipp<datatype>*
      pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR	16u_C1IR
8u_C3IR	16u_C3IR
8u_C4IR	16u_C4IR
8u_AC4IR	16u_AC4IR

```
IppStatus ippiAlphaPremulC_<mod>(Ipp<datatype> alpha, Ipp<datatype>*
      const pSrcDst[4], int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_AP4IR	16u_AP4IR

引数

pSrc	（ピクセル順序データの場合）ソース・バッファへのポインタ。 （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ。 （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ、または個別のソースおよびデスティネーション・カラー・プレーンへのポインタの配列
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
alpha	ピクセル値を事前に乗算するグローバルなアルファ値

説明

関数 ippiAlphaPremulC は、ippi.h ファイルの中で宣言される。この関数は、グローバルなアルファ値 *alpha* を使用して、1、3、4 チャネル・イメージまたはプレーン RGBA イメージを、アルファ値を事前に乗算した形式に変換する。
1、3、4 チャネル・イメージ・バッファの場合、各チャネルのピクセル値が α で乗算される。RGBA（ピクセル順序および平面）イメージのピクセル値が（r, g, b, a）である場合、この関数を実行すると、新しいピクセル値は（r* α , g* α , b* α , *alpha*）になる。
ここで、α は、0 ～ 1 の範囲で正規化された *alpha* の値である。

関数 ippiAlphaPremulC は、符号なしピクセル・データにのみ使用できる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

画像のカラー変換 6

本章では、カラー・スペース変換を実行するインテル® IPP 画像処理関数について説明する。インテル IPP ソフトウェアは、以下のカラー・スペース変換をサポートしている。

- カラー・モデルの変換
- カラーからグレー・スケールへの変換
- 高い量子化ビット数のカラーから低い量子化ビット数のカラーへの変換
- ピクセル順序形式からプレーン形式およびプレーン形式からピクセル順序形式への変換
- カラー・ツイスト
- ガンマ補正

すべてのインテル IPP カラー変換関数は、ソース・イメージのピクセルに対する点操作を実行する。特定のデスティネーション・ピクセルの変換後のチャネル値は、対応するソース・ピクセルのチャネル値のみを使用して計算され、隣接ピクセルの値は使用されない。したがって、関数の実行に使用される矩形処理対象領域（ROI）は、画面全体のサイズに合わせて拡張されることがある。

表 6-1 に、インテル IPP のカラー・スペース変換関数の一覧を示す。

表 6-1　カラー・スペース変換関数

変換のタイプ	関数の基本名	説明
カラー・モデルの変換	RGBToYUV, YUVToRGB	RGB 画像と YUV カラー・モデルを変換する。
	RGBToYUV422, YUV422ToRGB	4:2:2 サンプリング形式を使用して、RGB 画像と YUV カラー・モデルを変換する。
	RGBToYUV420, YUV420ToRGB	4:2:0 サンプリング形式を使用して、RGB 画像と YUV カラー・モデルを変換する。
	YUV420ToRGB565, YUV420ToRGB555, YUV420ToRGB444	4:2:0 サンプリング形式の YUV 画像を 16 ビットの RGB 画像に変換する。

続く

表 6-1　カラー・スペース変換関数（続き）

変換のタイプ	関数の基本名	説明
	YUV420ToBGR565, YUV420ToBGR555, YUV420ToBGR444	4:2:0 サンプリング形式の YUV 画像を 16 ビットの BGR 画像に変換する。
	RGBToYCbCr YCbCrToRGB	RGB 画像と YCbCr カラー・モデルを変換する。
	YCbCrToRGB565, YCbCrToRGB555, YCbCrToRGB444	YCbCr 画像を 16 ビット / ピクセルの RGB 画像に変換する。
	YCbCrToBGR565, YCbCrToBGR555, YCbCrToBGR444	YCbCr 画像を 16 ビット / ピクセルの BGR 画像に変換する。
	RGBToYCbCr422 YCbCr422ToRGB	4:2:2 サンプリング形式を使用して、RGB 画像と YCbCr カラー・モデルを変換する。
	RGBToCbYCr422 CbYCr422ToRGB	RGB 画像と 4:2:2 サンプリング形式の CbYCr カラー・モデルの間の変換を実行する。
	YCbCr422ToRGB565, YCbCr422ToRGB555, YCbCr422ToRGB444	16 ビット / ピクセルの YCbCr 画像を 16 ビット / ピクセルの RGB 画像に変換する。
	YCbCr422ToBGR565, YCbCr422ToBGR555, YCbCr422ToBGR444	16 ビット / ピクセルの YCbCr 画像を 16 ビット / ピクセルの BGR 画像に変換する。
	RGBToYCbCr420, YCbCr420ToRGB	RGB 画像と 4:2:0 サンプリング形式の YCbCr カラー・モデルの間の変換を実行する。
	YCbCr420ToBGR	4:2:0 サンプリング形式の YCbCr 画像を RGB カラー・モデルに変換する。
	YCbCr411ToBGR	4:1:1 サンプリング形式の YCbCr 画像を RGB カラー・モデルに変換する。
	RGBToXYZ XYZToRGB	RGB 画像と XYZ カラー・モデルを変換する。
	RGBToLUV LUVToRGB	RGB 画像と LUV カラー・モデルを変換する。
	RGBToYCC YCCToRGB	RGB 画像と YCC カラー・モデルを変換する。
	RGBToHLS HLSToRGB	RGB 画像と HLS カラー・モデルを変換する。
	BGRToHLS	BGR 画像を HLS カラー・モデルに変換する。
	HLSToBGR	HLS 画像を RGB カラー・モデルに変換する。

続く

表 6-1　カラー・スペース変換関数（続き）

変換のタイプ	関数の基本名	説明
	RGBToHSV HSVToRGB	RGB 画像と HSV カラー・モデルの間の変換を実行する。
カラーからグレー・スケールへの変換	RGBToGray	固定変換係数を使用して、RGB 画像をグレー・スケールに変換する。
	ColorToGray	独自の変換係数を使用して、RGB 画像をグレー・スケールに変換する。
ルックアップ・テーブル変換	LUT	輝度変換を適用して、画像をマッピングする。
	LUT_Linear	1 次補間による輝度変換を適用して、画像をマッピングする。
	LUT_Cubic	3 次補間による輝度変換を適用して、画像をマッピングする。
量子化ビット数の削減	ReduceBits	画像の量子化ビット数を減らす。
フォーマット変換	Join	422 プレーン形式の画像を 2 チャネルのピクセル順序画像に変換する。
	Split	422 形式の 2 チャネルのピクセル順序画像をプレーン形式に変換する。
カラー・ツイスト	ColorTwist	整数ピクセル値を含む画像にカラー・ツイスト・マトリックスを適用する。
	ColorTwist32f	浮動小数点ピクセル値を含む画像にカラー・ツイスト・マトリックスを適用する。
ガンマ補正	GammaFwd	ソースの RGB 画像のガンマ補正を実行する。
	GammaInv	ガンマ補正された R'G'B ' 画像を元の RGB 画像に変換する。

本章の前半では、各種のカラー・スペース・モデルについて説明する。インテル IPP カラー変換関数について理解するには、カラー・スペース・モデルの基礎知識が必要である。

カラー・スペースおよびカラー変換手法の詳細は、[Jack01]、[Rogers85]、[Foley90] を参照のこと。

ガンマ補正

ディスプレイが生成する輝度は、ほとんどの場合、印加される信号の 1 次関数ではなく、信号電圧の累乗（*ガンマ*と呼ばれる）に比例する。このため、高輝度の範囲は拡張され、低輝度の範囲は圧縮される。正確な色の再生を実現するには、このような非線形の性質を補正する必要がある。そのために、逆方向の変換を使用して、非線形形式に合わせて赤、緑、青の線形成分の輝度を下げる。このプロセスは「ガンマ補正」と呼ばれる。

インテル IPP 関数は、以下の基本方程式を使用して、RGB 画像をガンマ補正された R'G'B' 画像に変換する。

R,G,B < 0.018 の場合

$$R' = 4.5R$$
$$G' = 4.5G$$
$$B' = 4.5B$$

R,G,B ≥ 0.018 の場合

$$R' = 1.099R^{0.45} - 0.099$$
$$G' = 1.099G^{0.45} - 0.099$$
$$B' = 1.099B^{0.45} - 0.099$$

各チャネルの輝度の値は、[0..1] の範囲で正規化される。ガンマ値は、ITU Rec.709 仕様に従って、1/0.45 = 2.22 になる（[ITU709] を参照）。

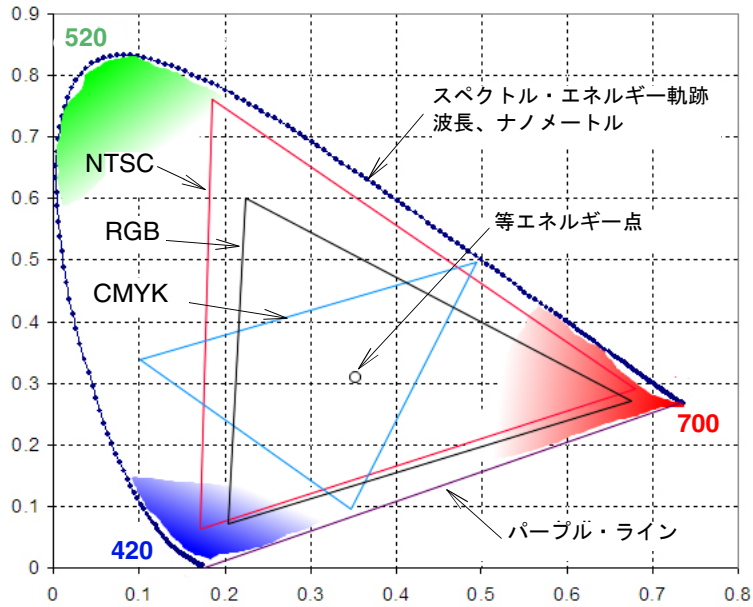
CIE の色度図と色域

図 6-1 は、すべての可視色を示す図である。この図は色度図と呼ばれ、国際照明学会（CIE）が実施した実験研究によって開発されたものである（http://members.eunet.at/cie/ を参照）。色度図は、*色度座標*と呼ばれる x（赤）成分と y（緑）成分の関数として、可視色を表現する。さまざまなスペクトル・カラー（バイオレットから赤まで）の位置は、*スペクトル軌跡*と呼ばれる舌状の形をした曲線上の点として指定される。この曲線の終点を結ぶ直線は、*パープル・ライン*と呼ばれる。等エネルギー点は、CIE による白色光の規格を表す。図中のすべての点は、スペクトル・カラーの何らかの混合を表している。純粋色（すなわち、最大彩度の色）は、スペクトル軌跡上の点に対応する。図中の任意の 2 点を結ぶ直線は、それらの 2 色を加法的に混合することで得られるすべてのカラー・バリエーションを定義する。任意の 3 点を頂点とする三角形は、各頂点に対応する 3 色を混合することで得られる色の全域を指定する。

人間の目の構造は 3 つの異なる刺激を識別できるため、色は 3 次元の性質を持つことになる。それぞれの色は、三刺激値（または三刺激成分）と呼ばれる 3 つのパラメータで記述される。これらの値は、例えば、主波長、純度、輝度や、光の三原色（赤、緑、青）である。

色度図は、どの 3 つの固定色の色域も、すべての可視色を表現できないことを示している。例えば、図 6-1 は、一般的なカラー CRT モニタの RGB 三原色、CMYK カラー印刷、NTSC テレビの再現可能な色域を示している。

図 6-1　CIE の *xyY* 色度図と色域



カラー・モデル

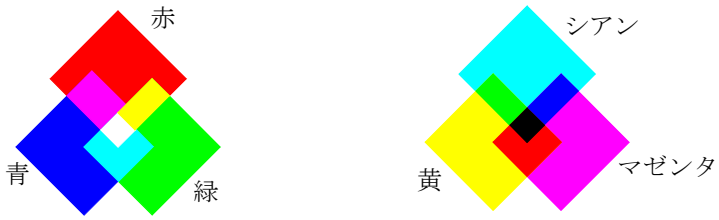
カラー・モデルの目的は、広く受け入れられる標準的な方法で、色の仕様を策定することである。基本的に、カラー・モデルは、3 次元座標系と、それぞれの色が 1 つの点で表現される部分空間の仕様である。

色を利用するそれぞれの業界では、その業界に最も合ったカラー・モデルを採用している。例えば、コンピュータ・グラフィックスには RGB カラー・モデル、ビデオ・システムには YUV または YCb カラー・モデル、PhotoCD* の製造には PhotoYCC* カラー・モデルが使用されている。異なる業界間でカラー情報をやり取りするには、カラー・モデルの値を変換する必要がある。インテル IPP は、各種のカラー・スペースと RGB の間の変換を実行する、多くの関数を用意している。

RGB カラー・モデル

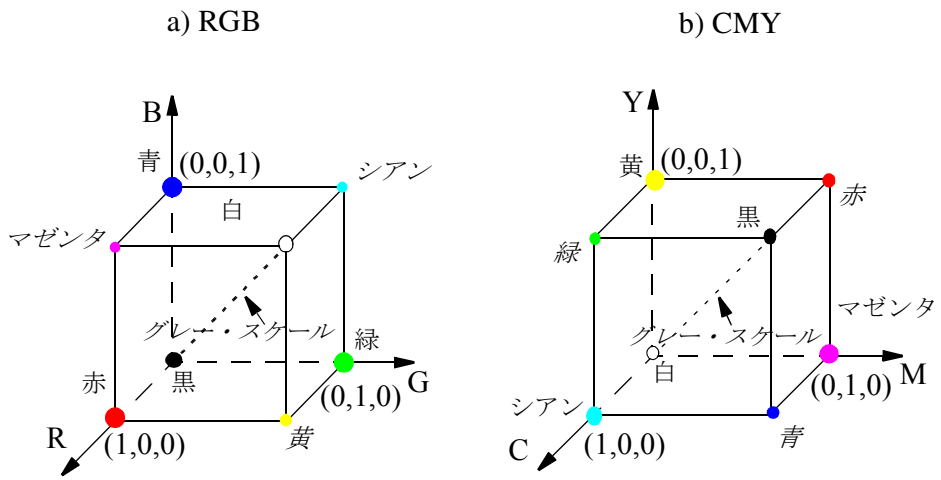
RGB モデルでは、それぞれの色は、赤、緑、青を混合したものとして表現される。このモデルは加法的モデルと呼ばれ、この 3 色は光の三原色と呼ばれる。三原色の加法によって、光の 2 次色が生成される（図 6-2 を参照）。光の 2 次色は、マゼンタ（赤＋青）、シアン（緑＋青）、および黄（赤＋緑）である。最大輝度の赤、緑、青を混合すると、白が生成される。RGB のカラー・サブスペースは、図 6-3 に示す立方体である（RGB の値は 0..1 の範囲で正規化されている）。この図では、RGB の値は立方体の 3 つの頂点、シアン、マゼンタ、黄はそれ以外の 3 つの頂点、黒は原点、白は原点から最も遠い頂点に対応する。

図 6-2　RGB モデルと CMYK モデルの三原色と 2 次色



グレー・スケールは、黒と白の 2 点を結ぶ対対角線に対応する。それぞれの色は、原点からのベクトルによって定義される、立方体の表面または内部の点で表現される。

図 6-3　RGB カラー・モデルと CMY カラー・モデル



したがって、RGB カラー・モデルの画像は、3 つの独立した画像プレーンで構成される（各プレーンが三原色のそれぞれに対応する）。

原則として、インテル IPP カラー変換関数は、ガンマ補正された非線形画像 R'G'B' を操作する。

RGB カラー・モデルの重要性は、人間の目が色を認識する方法と非常に密接に関連している点にある。カラー・ディスプレイは赤、緑、青を使用して必要な色を生成するため、RGB はコンピュータ・グラフィックスの基本的なカラー・モデルである。したがって、RGB カラー・スペースを選択すれば、コンピュータ・システムのアーキテクチャと設計が簡単になる。また、RGB カラー・スペースは早くから普及していたため、RGB カラー・スペースを使用して設計されたシステムは、多数の既存ソフトウェア・ルーチンを利用できる。

しかし、現実の画像を扱う場合、RGB はあまり効率的ではない。RGB カラー・キューブの中の色を生成するには、RGB の 3 つの成分のピクセル分解能と表示解像度がすべて同じ値でなければならない。また、画像を修正する場合は、3 つのプレーンをすべて修正しなければならない。

CMYK カラー・モデル

CMYK カラー・モデルは、RGB モデルのサブセットであり、主にカラー印刷に使用される。CMYK は、シアン、マゼンタ、黄、黒（K と表記される）の略語である。CMYK カラー・スペースは、減法的モデルである。つまり、シアン、マゼンタ、黄、黒の顔料またはインクを白の表面に適用することで、白の表面から何らかの色を引き、最終的な色を生成する。例えば（図 6-2 を参照）、シアンは白から赤を引いた色、マゼンタは白から緑を引いた色、黄は白から青を引いた色である。最大彩度の CMY を混合することによってすべての色を引くと、理論的には黒が描かれるはずである。しかし、実際の CMY インクには不純物が含まれているため、等しい最大彩度は得られない。また、RGB 光の一部は漏洩するため、実際には濁った茶色が描かれてしまう。したがって、CMY に黒のインクが追加される。CMY キューブを図 6-3 に示す。この図では、CMY の値は 3 つの頂点、赤、緑、青はそれ以外の 3 つの頂点、白は原点、黒は原点から最も遠い頂点に対応する。

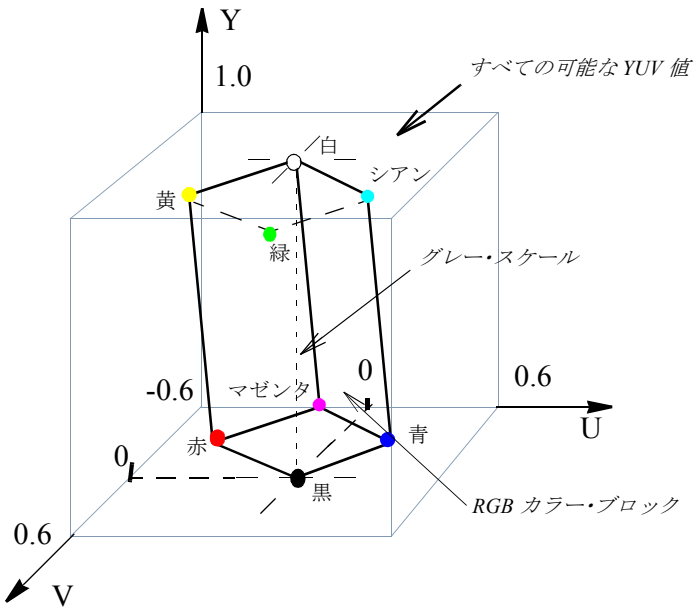
YUV カラー・モデル

YUV カラー・モデルは、アナログ・カラーテレビ放送に使用される基本的なカラー・モデルである。YUV は、伝送の効率性を向上し（帯域幅を最小限に抑え）、モノクロ・テレビとの下方互換性を保証するために、RGB を再コード化したモデルである。YUV カラー・スペースは、RGB スペースから「派生」したものである。YUV スペースは、*輝度*（Y）成分と 2 つの色差（U, V）成分で構成される。輝度は、赤、緑、青の成分の重み付けされた和として計算される。2 つの色差（すなわち、*クロミナンス*）成分は、それぞれ青と赤から輝度を引いて得られる。

画像処理における YUV モデルの主な利点は、輝度情報とカラー情報を分離したことである。これによって、画像のカラー成分に影響を与えずに、輝度成分だけを処理できる。例えば、YUV フォーマットのカラー画像のヒストグラム等頻度化を実行するには、画像の Y 成分にヒストグラム等頻度化を適用すればよい。

RGB で表現可能な色は、公称範囲によって制限される YUV スペースの一部だけを占める。したがって、YUV の公称範囲内に、無効な RGB 値に対応する YUV 値の組み合わせが多数存在する。図 6-4 は、RGB カラー・キューブに対応する、YUV スペース内の有効なカラー・ブロックを示している（RGB 値は [0..1] の範囲で正規化されている）。

図 6-4　YUV カラー・スペース内の RGB カラー・キューブ



Y'U'V' の表記は、各成分がガンマ補正された R'G'B' から得られたという意味である。これらの非線形成分の重み付けされた和から、*luma* Y' と呼ばれる輝度を表す信号が得られる（*luma* は、多くの場合、おおまかに輝度（*luminance*）と呼ばれる。したがって、*輝度*の用語については、線形と非線形のどちらを意味するのかに注意する必要がある）。

インテル IPP 関数は、以下の基本方程式 [Jack01] を使用して、ガンマ補正された R'G'B' モデルと Y'U'V' モデルの間の変換を実行する。

```
Y'=  0.299*R' + 0.587*G' + 0.114*B'
U'= -0.147*R' - 0.289*G' + 0.436*B' = 0.492*(B'- Y')
V'=  0.615*R' - 0.515*G' - 0.100*B' = 0.877*(R'- Y')

R' = Y' + 1.140*V'
G' = Y' - 0.394*U' - 0.581*V'
B' = Y' + 2.032*U'
```

YUV モデルには、4:4:4、4:2:2、および 4:2:0 など、複数のサンプリング形式がある。インテル IPP カラー変換関数は、これらのサンプリング形式をサポートしている。これらの形式については、本章後半で説明する（「画像のダウンサンプリング」を参照）。

YCbCr スペース内の RGB カラー・キューブ

YCbCr カラー・スペースは、ITU-R BT.601 勧告の一部として開発され、コンポーネント・デジタル・ビデオに使用されている。YCbCr カラー・スペースは、YUV カラー・スペースをスケーリングし、オフセットしたものである。

インテル IPP 関数は、以下の基本方程式 [Jack01] を使用して、0 ～ 255 の範囲の R′G′B′ と Y′Cb′Cr′ の間の変換を実行する（Y′Cb′Cr′ の表記は、すべての成分がガンマ補正された R′G′B′ から得られたという意味である）。

```
Y′  =   0.257*R' + 0.504*G' + 0.098*B' + 16
Cb' = -0.148*R' - 0.291*G' + 0.439*B' + 128
Cr' =  0.439*R' - 0.368*G' - 0.071*B' + 128

R' = 1.164*(Y′-16) + 1.596*(Cr'-128)
G' = 1.164*(Y′-16) - 0.813*(Cr'-128) - 0.392*(Cb'-128)
B' = 1.164*(Y′-16) + 2.017*(Cb'-128)
```

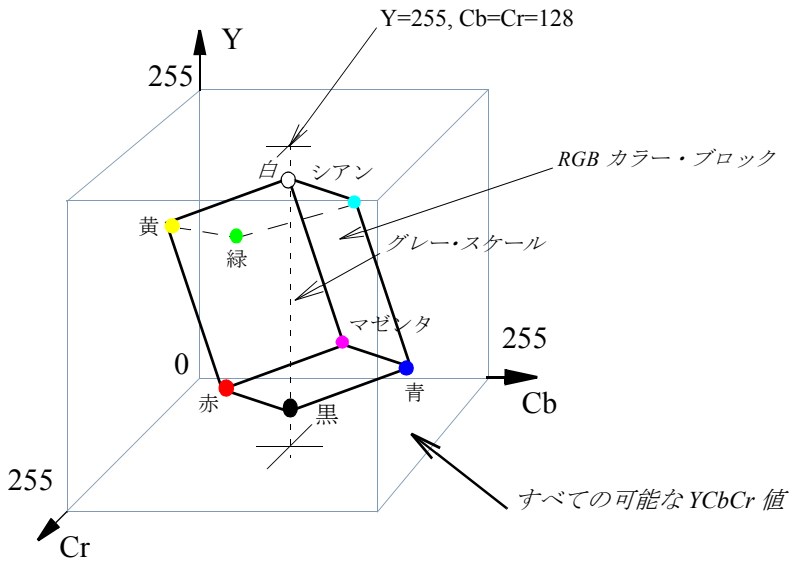
JPEG コーデック専用のインテル IPP カラー変換関数は、以下の異なる方程式を使用する。

```
Y  =   0.299*R + 0.5587*G + 0.114*B
Cb = -0.116874*R - 0.33126*G + 0.5*B + 128
Cr =  0.5*R - 0.41869*G - 0.08131*B + 128


R =  Y + 1.402*Cr - 179,456)
G =  Y - 0.34414*Cb - 0.71414*Cr + 135.45984)
B =  Y  + 1.772*Cb - 226.816
```

YCCK モデルは、JPEG 画像圧縮専用のモデルである。YCCK は、YCbCr モデルの変種であり、追加の *K* チャネル（黒）を含む。輝度情報とカラー情報が分離されている場合、JPEG コーデックはより効率的に実行される。したがって、JPEG 圧縮を実行する前に、CMYK 画像を YCCK に変換する必要がある（詳細は、第 15 章の ippiCMYKToYCCK_JPEG 関数の説明を参照）。

RGB で表現可能な色は、公称範囲によって制限される YCbCr カラー・スペース（図 6-5 を参照）の一部だけを占める。したがって、無効な RGB 値に対応する YCbCr 値の組み合わせが多数存在する。

図 6-5　YCbCr スペース内の RGB カラー・キューブ



YCbCr モデルには、4:4:4、4:2:2、4:1:1、4:2:0 など、複数のサンプリング形式がある。インテル IPP カラー変換関数は、これらのサンプリング形式をサポートしている。これらの形式については、本章後半で説明する（「画像のダウンサンプリング」を参照）。

PhotoYCC カラー・モデル

Kodak* PhotoYCC* は、Photo CD* 画像データのエンコード用に開発された。このモデルは、ITU 勧告 601 と 709 の両方に基づいており、BT.601 YCbCr および BT.709（[ITU709]）と同様に、色を輝度とクロミナンスで表現する。このモデルは、輝度（Y）成分と 2 つの色差（すなわち、クロミナンス）（C1, C2）成分で構成される。PhotoYCC は、カラー写真の感光材向けに最適化され、現在ディスプレイ上で表示可能な色域より大きな色域を提供する。

インテル IPP 関数は、以下の基本方程式 [Jack01] を使用して、ガンマ補正された非線形 R′G′B′ を Y′C′C′ に変換する。

```
Y′ =  0.213*R' + 0.419*G' + 0.081*B'
C1′ = -0.131*R' - 0.256*G' + 0.387*B'+ 0.612
C2′ =  0.373*R' - 0.312*R' - 0.061*B' + 0.537
```

上の方程式は、R'、G'、B' の値が [0..1] の範囲で正規化されていることを前提としている。

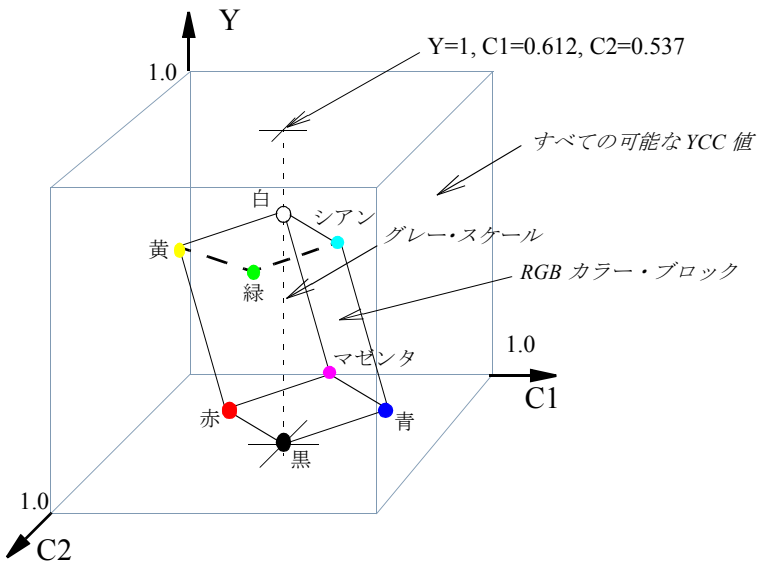
PhotoYCC モデルはフィルムのダイナミック・レンジを維持しようとするため、PhotoYCC 画像をデコードするには、出力デバイスに合ったカラー・スペースと範囲を選択する必要がある。したがって、デコード方程式は、エンコード方程式を完全に逆にしたものであるとは限らない。インテル IPP は、以下の方程式 [Jack01] を使用して、CRT ディスプレイを発光させる R′G′B′ 値を生成する。これらの方程式は、エンコードされる画像の輝度と表示される画像の輝度が一致することを前提としている。

```
R' = 0.981 * Y + 1.315 * (C2 - 0.537)
G' = 0.981 * Y - 0.311 * (C1 - 0.612) - 0.669 * (C2 - 0.537)
B' = 0.981 * Y + 1.601 * (C1 - 0.612)
```

上の方程式は、Y、C1、C2 のソース値が [0..1] の範囲で正規化され、ディスプレイの三原色の色度値が [ITU709] 仕様に適合することを前提としている。

RGB で表現可能な色は、公称範囲によって制限される YCC カラー・スペース （図 6-6 を参照）の一部だけを占める。したがって、無効な RGB 値に対応する YCbCr 値の組み合わせが多数存在する。

図 6-6　YCC カラー・スペース内の RGB カラー



HSV および HLS カラー・モデル

HLS（色相 hue、明度 lightness、彩度 saturation）および HSV（色相 hue、彩度 saturation、明度 value）カラー・モデルは、より「直感的に」色を操作するために開発された。このモデルは、人間による色の知覚と解釈によく似た構造を持つ。

色相（*Hue*）は、色それ自体を定義する。色相軸の値は、0 ～ 360 の範囲で変化する。色相の値は、0 の赤から始まり、緑、青、およびすべての中間色を通って、360 の赤で終わる。

彩度（*Saturation*）は、色相がニュートラル・グレーからどの程度離れているかを示す。彩度の値は、0 カラー彩度なし）～ 1（特定の照度での特定の色相の最大彩度）の範囲で変化する。

輝度 （Intensity）成分、すなわち*明度* （HLS モデルでは *lightness*、HSV モデルでは *value*）は、照度のレベルを示す。明度の値は、0 （黒、光なし）～ 1 （白、最大照度）の範囲で変化する。HSV モデルと HLS モデルの相違点は次のとおりである。HSV カラー・モデルでは、色相の最大彩度の条件は S=1 および最大照度 （V=1）であるが、HLS カラー・モデルでは、最大彩度の条件は明度 L=0.5 である。

HSV カラー・スペースは、基本的に円柱形であるが、通常は円錐形または六角錐で表現される（図 6-7 を参照）。この六角錐は、有効な RGB 値に対応する HSV の部分空間を定義する。*明度*（*value*）V が垂直軸であり、頂点 V=0 は黒に対応する。同様に、HLS モデルのカラー・ソリッド（3D 表現）は二重の六角錐になる（図 6-8）。*明度*（*lightness*）が垂直軸になり、2 番目の六角錐の頂点が白に対応する。

図 6-7　HSV ソリッド

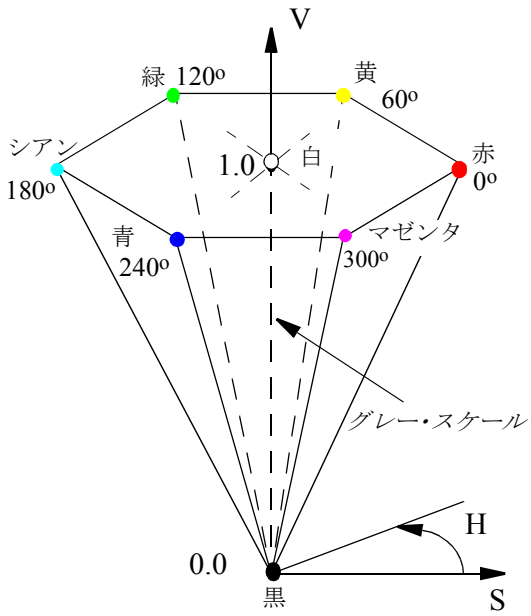
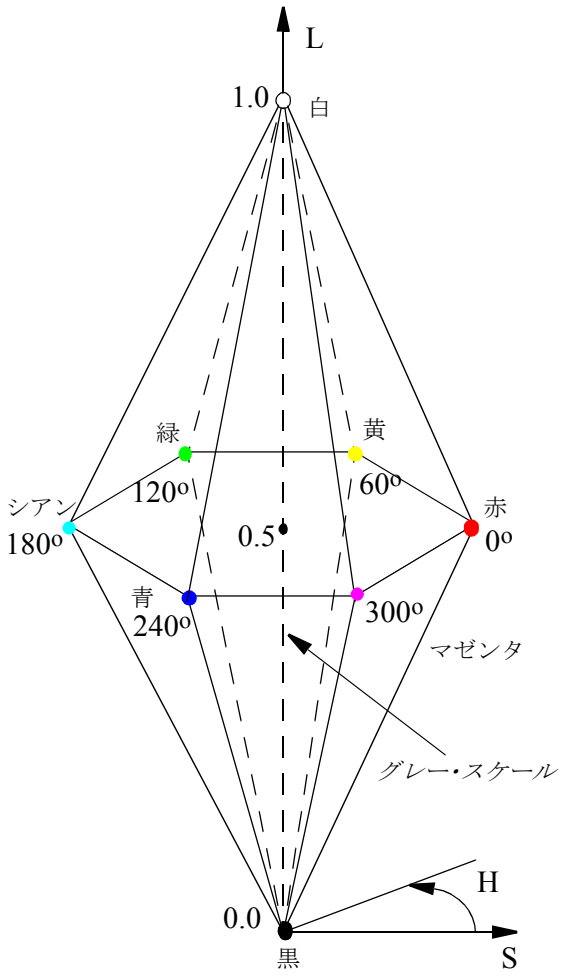


図 6-8　HLS ソリッド



いずれのカラー・モデルも、輝度成分とカラー情報が分離されている。HSV カラー・スペースの方が、HLS より彩度のダイナミック・レンジが大きくなる。

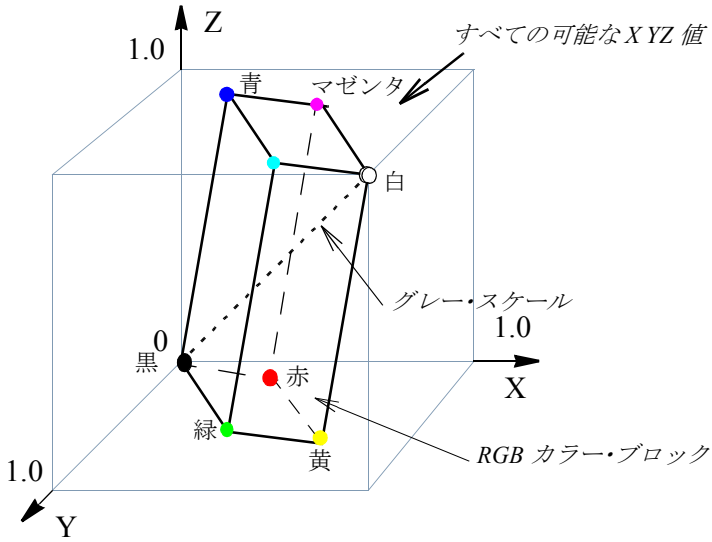
インテル IPP 関数は、RGB と HSV/HLS の間の変換を、各関数の疑似コード・アルゴリズム [Rogers85] に従って実行する。このアルゴリズムは、対応する変換関数の説明に記載されている。

CIE XYZ カラー・モデル

XYZ カラー・スペースは、国際照明学会（CIE）が開発した国際規格である。このモデルは、仮定上の三原色 XYZ に基づくもので、すべての可視色を X、Y、Z の正の値だけで表現できる。CIE XYZ 三原色は、現実の光波長に対応しない、仮定上の色である。原色 Y は、できるだけ輝度に一致するように定義されている。原色 X と Z は、カラー情報を指定する。
CIE XYZ スペース（およびそれに基づくすべてのカラー・スペース）の主な利点は、このスペースがデバイスに全く依存しないことである。図 6-1 の色度図は、実際には、CIE XYZ サブスペースを 2 次元に投影したものである。
公称範囲内の X、Y、Z の値を任意に混合すると、可視色スペクトルの外側の「色」が簡単に生成されることに注意する必要がある。

XYZ スペース内の RGB で表現可能な色のブロックの位置を図 6-9 に示す。

図 6-9　XYZ カラー・スペース内の RGB カラー・キューブ



インテル IPP 関数は、以下の基本方程式 [Rogers85] を使用して、ガンマ補正された R'G'B' モデルと CIE XYZ モデルの間の変換を実行する。

```
X =  0.412453*R′ + 0.35758 *G′ + 0.180423*B′
Y =  0.212671*R′ + 0.71516 *G′ + 0.072169*B′
Z =  0.019334*R′ + 0.119193*G′ + 0.950227*B′
```

X, Y, Z の計算用の方程式は、R′、G′、B′ の値が [0..1] の範囲で正規化されていることを前提としている。

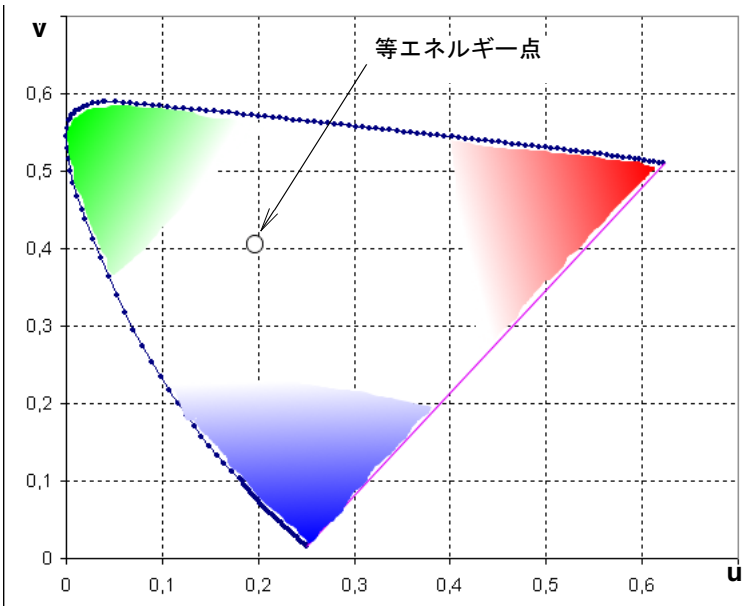
```
R′ =  3.240479 * X - 1.53715  * Y  - 0.498535 * Z
G′ = -0.969256 * X + 1.875991 * Y  + 0.041556 * Z
B′ =  0.055648 * X - 0.204043 * Y  + 1.057311 * Z
```

R′,G′,B′ の計算用の方程式は、X、Y、Z の値が [0..1] の範囲で正規化されていることを前提としている。

CIE LUV カラー・モデル

CIE LUV カラー・スペースは、標準の CIE XYZ スペースから派生する、知覚的に均等のモデルである（「知覚的に均等」とは、カラー・スペース内で等しい距離にある 2 つの色は、知覚的にも等しい距離にあるという意味である）。これを実現するために、CIE は、UCS （Uniform Chromaticity Scale）ダイアグラムを提案している （図 6-10）。UCS ダイアグラムは、数学的公式を使用して、XYZ の値または x, y 座標 （図 6-1）を、視覚的により正確な 2 次元モデルを表現する新しい値 （u, v）に変換する。明度のスケール *Y* は、*L* と呼ばれる新しいスケールで置き換えられる。このスケールは、ほぼ均等な間隔を持つが、実際の目に見える違いをより正確に表している。

図 6-10　CIE の *u′,v′* UCS （Uniform Chromaticity Scale）ダイアグラム



CIE LUV カラー・スペースは、次のように CIE XYZ から得られる（Rogers85）。

$L = 116. * (Y/Y_n)^{1/3}. - 16.$

$U = 13. * L * (u - u_n)$

$V = 13. * L * (v - v_n)$

ここで、

$u = 4.*X / (X + 15.*Y + 3.*Z)$

$v = 9.*Y / (X + 15.*Y + 3.*Z)$

$u_n = 4.*xn / (-2.*x_n + 12.*y_n + 3.)$

$v_n = 9.*yn / (-2.*x_n + 12.*y_n + 3.)$

逆方向の変換は、以下の方程式に従って実行される。

$Y = Y_n * ((L + 16.) / 116.)^3.$

$X = -9.* Y * u / ((u - 4.)* v - u * v)$

$Z = (9.* Y - 15*v*Y - v*X) / 3. * v$

ここで、

$u = U / (13.* L) + u_n$

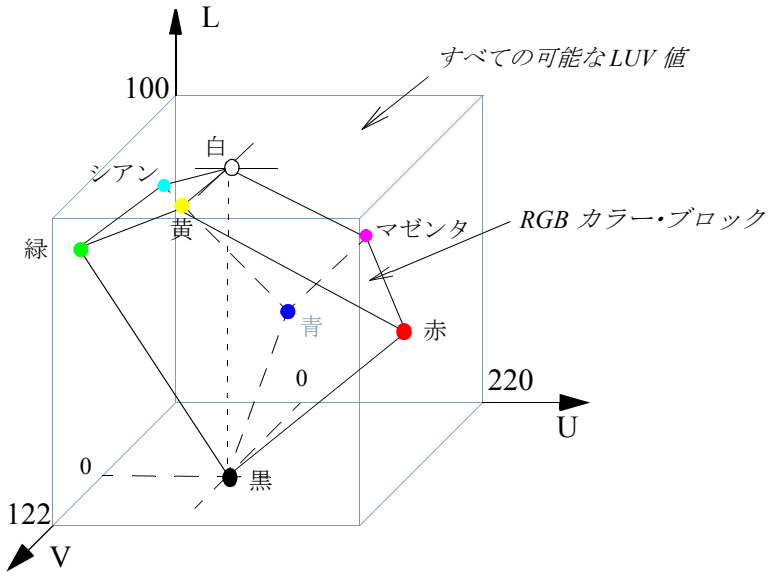
$v = V / (13.* L) + v_n$

u_n, v_n は前の式で定義されている。

ここで、$x_n = 0.312713$, $y_n = 0.329016$ は、D65 白色点の CIE 色度座標である（[ITU709]）。$Y_n = 1.0$ は、D65 白色点の輝度である。L 成分の計算値の範囲は [0..100]、U 成分の範囲は [-134..220]、V 成分の範囲は [-140..122] である。

RGB で表現可能な色は、公称範囲によって制限される LUV カラー・スペースの一部だけを占める（図 6-11 を参照）。したがって、無効な RGB 値に対応する LUV 値の組み合わせが多数存在する。

図 6-11　CIE LUV カラー・スペース内の RGB カラー・キューブ



画像のダウンサンプリング

通常、デジタル・カラー画像は、各ピクセルをカラー・スペース座標の特定の値に設定することで表現される。輝度座標とクロミナンス座標を分離した（YUV タイプの）カラー・スペースを使用すると、画像の有効なカラー定義に必要なビット数を減らせる。人間の目はクロミナンスの変化より輝度の変化に敏感であるため、ビット数の削減が可能になる。この方法の基礎となる考え方は、輝度成分の値は各ピクセルに個別に設定し、クロミナンス成分（カラー）は、特定の規則に従ってピクセルのグループ（*マクロピクセル*と呼ばれる）に同じ値を割り当てることである。このプロセスは*ダウンサンプリング*と呼ばれる。基礎となる方式によって、さまざまなサンプリング形式がある。

インテル IPP 画像処理関数（JPEG 関数を除く）は、以下のサンプリング形式をサポートしている。

4:4:4 YUV （YCbCr）- 標準の形式。ダウンサンプリングを行わない。すべてのピクセルの Y、U（Cb）、V（Cr）成分をサンプリングする。各成分が 8 ビットの場合、各ピクセルに 24 ビットが必要である。この形式は、多くの場合、"4:4:4" の記述子を省略して、YUV （YCbCr）形式と呼ばれる。

4:2:2 YUV （YCbCr）- 2:1 の水平方向のダウンサンプリングを使用する。すなわち、すべてのピクセルの Y 成分をサンプリングし、U（Cb）成分と V（Cr）成分は水平方向に 1 ピクセルおきにサンプリングする。各成分が 8 ビットの場合、ピクセルのペアに 32 ビットが必要である。

4:1:1 YCbCr - 4:1 の水平方向のダウンサンプリングを使用する。すなわち、すべてのピクセルの Y 成分をサンプリングし、Cb 成分と Cr 成分は水平方向に 3 ピクセルおきにサンプリングする。各成分が 8 ビットの場合、水平方向の 4 ピクセルごとに 48 ビットが必要である。

4:2:0 YUV（YCbCr）- 2:1 の水平方向のダウンサンプリングと 2:1 の垂直方向のダウンサンプリングを使用する。すべてのピクセルの Y 成分をサンプリングし、U（Cb）成分と V（Cr）成分は 2 × 2 ピクセルのブロックごとにサンプリングする。各成分が 8 ビットの場合、4 ピクセルのブロックごとに 48 ビットが必要である。

JPEG 圧縮では、ダウンサンプリングは異なる特徴を持ち、表記方法が多少異なる。JPEG 関数の領域では、サンプリング形式は MCU（Minimal Coded Units）の構造を指定する。したがって、JPEG コーデック専用のインテル IPP 関数は、以下のサンプリング形式をサポートしている。

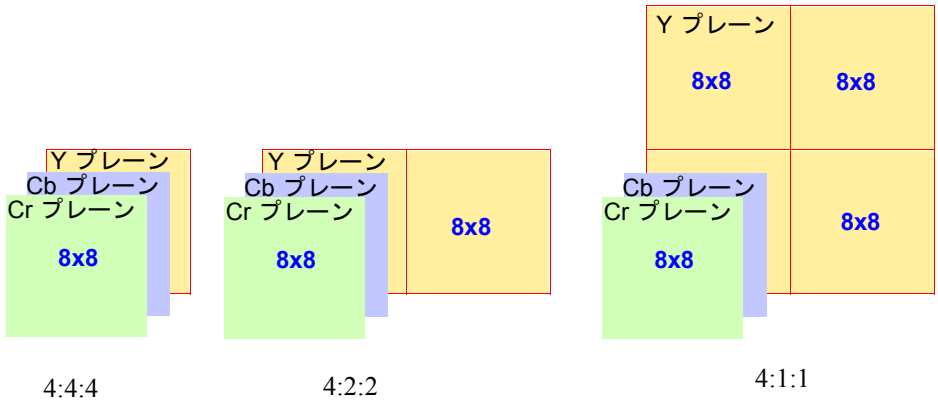
4:4:4 YCbCr - Y サンプルの各 8 × 8 ブロックに、Cb サンプルの 8 × 8 ブロックと Cr サンプルの 8 × 8 ブロックが 1 つずつ対応する。

4:2:2 YCbCr - 水平方向に 1 つおきの Y サンプルの 8x8 ブロックに、Cb サンプルの 8 × 8 ブロックと Cr サンプルの 8 × 8 ブロックが 1 つずつ対応する。

4:1:1 YCbCr - Y サンプルの 8 × 8 ブロック 4 つ（水平方向に 2 つ、垂直方向に 2 つ）ごとに、Cb サンプルの 8x8 ブロックと Cr サンプルの 8 × 8 ブロックが 1 つずつ対応する。

それぞれのサンプリング形式に対応する MCU の構造を図 6-12 に示す。

図 6-12　各種の JPEG サンプリング形式の MCU 構造



RGB 画像フォーマット

インテル IPP カラー変換関数は、24 ビット / ピクセルの RGB/BGR 画像フォーマット以外に、32 ビット / ピクセルの RGB / BGR フォーマットをサポートしている（このフォーマットは、RGB 3 チャネル＋アルファ・チャネルで構成される）。24 ビット・フォーマットでは、各カラーは 1 バイト、各ピクセルは 3 バイトである。32 ビット・フォーマットでは、各カラーは 1 バイト、アルファ成分は 1 バイトで、1 ピクセル当たり 4 バイトになる。これらのフォーマットのメモリ・レイアウトを表 6-2 に示す。

16 ビット・フォーマットでは、各ピクセルは 2 バイトで、各カラーは指定されたビット数を使用する。図 6-13 は、サポートしているすべての 16 ビット / ピクセル・フォーマットと、メモリ・レイアウト （ビット・オーダ）を示している。

図 6-13　16 ビット・ピクセル・フォーマット

16 ビット・ピクセル・フォーマット

	上位バイト	下位バイト
RGB565	B4 B3 B2 B1 B0 G5 G4 G3	G2 G1 G0 R4 R3 R2 R1 R0
RGB555	X B4 B3 B2 B1 B0 G4 G3	G2 G1 G0 R4 R3 R2 R1 R0
RGB444	X X X X B3 B2 B1 B0	G3 G2 G1 G0 R3 R2 R1 R0
BGR565	R4 R3 R2 R1 R0 G5 G4 G3	G2 G1 G0 B4 B3 B2 B1 B0
BGR555	X R4 R3 R2 R1 R0 G4 G3	G2 G1 G0 B4 B3 B2 B1 B0
BGR444	X X X X R3 R2 R1 R0	G3 G2 G1 G0 B3 B2 B1 B0

R - 赤、G - 緑、B - 青、X - 空きビット

ピクセル画像フォーマットとプレーン画像フォーマット

画像データの格納形式は、ピクセル指向またはプレーン指向（プレーン）である。ピクセル順序の画像では、各ピクセルのすべてのチャネル値がまとめられ、連続して格納される。メモリ・レイアウトは、カラー・モデルとダウンサンプリング方式によって異なる。

表 6-2 は、インテル IPP カラー変換関数がサポートしているすべてのピクセル順序の画像フォーマットと、それに対応するチャネル値の順序を示している（ここで、下線付きの記号のグループが 1 つのピクセルを表す。記号 A はアルファ・チャネル値を示す）。この表の最後の欄は、その画像フォーマットを使用するインテル IPP カラー変換関数の例を示している。

表 6-2　ピクセル順序の画像フォーマット

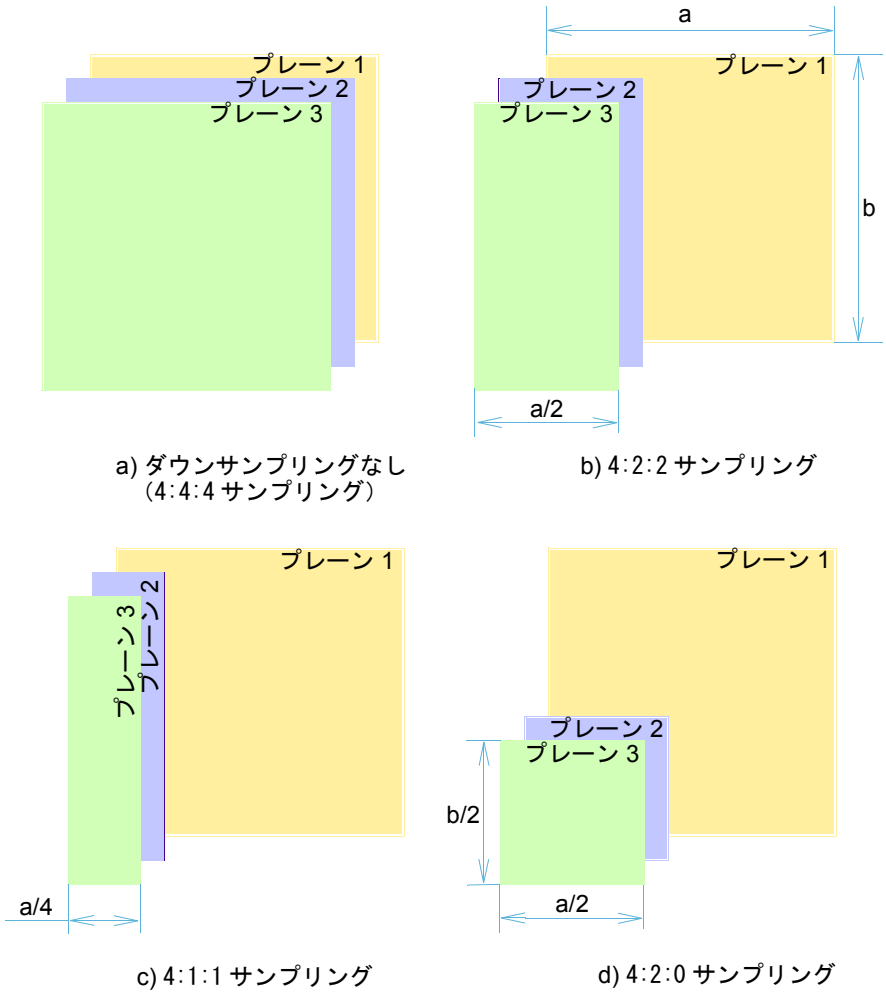
画像フォーマット	チャネル数	チャネル値の順序	例
RGB	3	R0 G0 B0 R1 G1 B1 R2 G2 B2	ippiRGBToYUV_8u_C3R
RGB444			ippiYCbCrToRGB444_8u16u_C3R
RGB555			ippiYCbCrToRGB555_8u16u_C3R
RGB565			ippiYCbCrToRGB565_8u16u_C3R
RGB	4	R0 G0 B0 A0 R1 G1 B1 A1	ippiRGBToYUV_8u_AC4R
BGR	3	B0 G0 R0 B1 G1 R1 B2 G2 R2	ippiYCbCrToBGR_8u_P3C3R
BGR444			ippiYCbCrToBGR444_8u16u_C3R
BGR555			ippiYCbCrToBGR555_8u16u_C3R
BGR565			ippiYCbCrToBGR565_8u16u_C3R
BGR	4	B0 G0 R0 A0 B1 G1 R1 A1	ippiBGRToHLS_8u_AC4R
YUV	3	Y0 U0 V0 Y1 U1 V1 Y2 U2 V2	ippiYUVToRGB_8u_C3R
YUV	4	Y0 U0 V0 A0 Y1 U1 V1 A1	ippiYUVToRGB_8u_AC4R
4:2:2 YUV	2	Y0 U0 Y1 V0 Y2 U1 Y3 V1	ippiYUV422T0RGB_8u_C2C3R
YCbCr	3	Y0 Cb0 Cr0 Y1 Cb1 Cr1	ippiYCbCrToRGB_8u_C3R
YCbCr	4	Y0 Cb0 Cr0 A0 Y1 Cb1 Cr1 A1	ippiYCbCrToRGB_8u_AC4R
4:2:2 YCbCr	2	Y0 Cb0 Y1 Cr0 Y2 Cb1 Y3 Cr1	ippiYCbCr422ToRGB_8u_C2C3R
4:2:2 CbYCr	2	Cb0 Y0 Cr0 Y1 Cb1 Y2 Cr1 Y3	ippiCbYCr422ToRGB_8u_C2C3R
XYZ	3	X0 Y0 Z0 X1 Y1 Z1 X2 Y2 Z2	ippiXYZToRGB_8u_C3R
XYZ	4	X0 Y0 Z0 X1 Y1 Z1 X2 Y2 Z2	ippiXYZToRGB_16u_AC4R
LUV	3	L0 U0 V0 L1 U1 V1 L2 U2 V2	ippiLUVToRGB_16s_C3R
LUV	4	L0 U0 V0 A0 L1 U1 V1 A1	ippiLUVToRGB_32f_AC4R
YCC	3	Y0 C0 C0 Y1 C1 C1 Y2 C2 C2	ippiYCCToRGB_8u_C3R
YCC	4	Y0 C0 C0 A0 Y1 C1 C1 A1	ippiYCCToRGB_8u_AC4R
HLS	3	H0 L0 S0 H1 L1 S1 H2 L2 S2	ippiHLSToRGB_16u_C3R
HLS	4	H0 L0 S0 A0 H1 L1 S1 A0	ippiHLSToRGB_16u_AC4R
HSV	3	H0 S0 V0 H1 S1 V1 H2 S2 V2	ippiHSVToRGB_16s_C3R
HSV	4	H0 S0 V0 A0 H1 S1 C1 A1	ippiHSVToRGB_16s_AC4R

表 6-3 は、インテル IPP カラー変換関数がサポートしているプレーン画像フォーマットと、そのフォーマットを使用するインテル IPP 関数の例を示している。各プレーンのレイアウトと相対的な大きさを図 6-14 に示す。

表 6-3　プレーン画像フォーマット

画像フォーマット	プレーン数	プレーンのレイアウト pl1、pl2、pl3				例
RGB	3	R,	G,	B	図 6-14、a を参照	ippiRGBToYUV_8u_P3R
YUV	3	Y,	U,	V	図 6-14、a を参照	ippiYUVToRGB_8u_P3R
4:2:2 YUV	3	Y,	U,	V	図 6-14、b を参照	ippiYUV422ToRGB_8u_P3
4:2:0 YUV	3	Y,	U,	V	図 6-14、d を参照	ippiYUV420ToRGB_8uP3C3R
YCbCr	3	Y,	Cb,	Cr,	図 6-14、a を参照	ippiYCbCrToRGB_8u_P3R
4:2:2 YCbCr	3	Y,	Cb,	Cr,	図 6-14、b を参照	ippiYCbCr422ToRGB_8u_P3C3R
4:1:1 YCbCr	3	Y,	Cb,	Cr,	図 6-14、c を参照	ippiYCbCr411ToRGB_8u_P3C3R
4:2:0 YCbCr	3	Y,	Cb,	Cr,	図 6-14、d を参照	ippiRGBToYCbCr420_8u_C3P3R

図 6-14　プレーンのサイズとレイアウト



a) ダウンサンプリングなし（4:4:4 サンプリング）

b) 4:2:2 サンプリング

c) 4:1:1 サンプリング

d) 4:2:0 サンプリング

RGBToYUV

RGB 画像を YUV カラー・モデルに変換する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiRGBToYUV_<mod>(const Ipp8u* pSrc, int srcStep,
      Ipp8u*
      pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R
8u_AC4R
```

事例 2：プレーン・データの操作

```
IppStatus ippiRGBToYUV_8u_P3R(const Ipp8u* const pSrc[3], int
      srcStep,
      Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

事例 3：ピクセル順序データからプレーン・データへの変換

```
IppStatus ippiRGBToYUV_8u_C3P3R(const Ipp8u* pSrc, int
      srcStep, Ipp8u*
      pDst[3], int dstStep, IppiSize roiSize);
```

引数

pSrc	（ピクセル順序データの場合）ソース・バッファへのポインタ （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRGBToYUV は、ippi.h ファイルの中で宣言される。この関数は、次の公式に従って、ガンマ補正された R'G'B' 画像 *pSrc* を Y'U'V' 画像 *pDst* に変換する。

```
Y' =  0.299*R' + 0.587*G' + 0.114*B'
U' = -0.147*R' - 0.289*G' + 0.436*B' = 0.492*(B'-Y')
V' =  0.615*R' - 0.515*G' - 0.100*B' = 0.877*(R'-Y')
```

[0 .. 255] の範囲のデジタル RGB 値の場合、Y' の範囲は [0..255] であり、U は [-112..+112] の範囲内で変化し、V は [-157..+157] の範囲内で変化する。結果を [0..255] の範囲内に収めるために、計算された U と V の値に定数 128 を加算し、V を飽和処理する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YUVToRGB

YUV 画像を RGB カラー・モデルに変換する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiYUVToRGB_<mod>(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R
8u_AC4R
```

事例 2：プレーン・データの操作

```
IppStatus ippiYUVToRGB_8u_P3R(const Ipp8u* const pSrc[3], int
      srcStep,
      Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

事例 3：プレーン順序データからピクセル順序データへの変換

```
IppStatus ippiYUVToRGB_8u_P3C3R(const Ipp8u* const pSrc[3],
      int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	（ピクセル順序データの場合）ソース・バッファへのポインタ （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYUVToRGB は、ippi.h ファイルの中で宣言される。この関数は、次の公式に従って、Y'U'V' 画像 *pSrc* をガンマ補正された R'G'B' 画像 *pDst* に変換する。

```
R' = Y' + 1.140*V'
      G' = Y' - 0.394*U' - 0.581*V'
      B' = Y' + 2.032*U'
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

RGBToYUV422

RGB 画像を YUV カラー・モデルに変換する。4:2:2 サンプリングを使用する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiRGBToYUV422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

事例 2：プレーン・データの操作（ROI を使用する場合）

```
IppStatus ippiRGBToYUV422_8u_P3R(const Ipp8u* const pSrc[3], int
      srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

事例 3：プレーン・データの操作（ROI を使用しない場合）

```
IppStatus ippiRGBToYUV422_8u_P3(const Ipp8u* const pSrc[3], Ipp8u*
      pDst[3], IppiSize imgSize);
```

事例 4：ピクセル順序データからプレーン・データへの変換（ROI を使用する場合）

```
IppStatus ippiRGBToYUV422_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

事例 5：ピクセル順序データからプレーン・データへの変換（ROI を使用しない場合）

```
IppStatus ippiRGBToYUV422_8u_C3P3(const Ipp8u* pSrc, Ipp8u*
      pDst[3],
      IppiSize imgSize);
```

引数

pSrc	（ピクセル順序データの場合）ソース・イメージへのポインタ （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	（ROI を使用する処理の場合）ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・イメージへのポインタ （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	（ROI を使用する処理の場合）デスティネーション・イメージ・バッファ内のステップ（バイト単位）。プレーン・イメージの場合は、3 つのステップ値の配列
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
imgSize	（ROI を使用しない処理の場合）ソース・イメージとデスティネーション・イメージのサイズ（ピクセル単位）

説明

関数 ippiRGBToYUV422 は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiRGBToYUV と同じ公式に従って、4:2: サンプリング形式でガンマ補正された R'G'B' 画像 *pSrc* を Y'U'V' 画像 *pDst* に変換する。サンプリング形式の詳細は、表 6-2 および表 6-3 を参照のこと。

[0..255] の範囲のデジタル RGB 値の場合、Y' の範囲は [0..255] であり、U は [-112..+112] の範囲内で変化し、V は [-157..+157] の範囲内で変化する。結果を [0..255] の範囲内に収めるために、計算された U と V の値に定数 128 を加算し、V を飽和処理する。

ROI を使用しない関数タイプは、ソース・イメージとデスティネーション・イメージが同じサイズであり、連続するメモリ領域を占める（つまり、画像の行が 0 で埋められていない）ことを前提として機能する。この場合は、ステップ・パラメータは不要である。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* または *imgSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YUV422ToRGB

4:2:2 サンプリング形式の YUV 画像を RGB カラー・モデルに変換する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiYUV422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

事例 2：プレーン・データの操作（ROI を使用する場合）

```
IppStatus ippiYUV422ToRGB_8u_P3R(const Ipp8u* const pSrc[3],
      int srcStep[3], Ipp8u* pDst[3], int dstStep, IppiSize
      roiSize);
```

事例 3：プレーン・データの操作（ROI を使用しない場合）

```
IppStatus ippiYUV422ToRGB_8u_P3(const Ipp8u* const pSrc[3],
      Ipp8u* pDst[3], IppiSize imgSize);
```

事例 4：プレーン順序データからピクセル順序データへの変換（ROI を使用する場合）

```
IppStatus ippiYUV422ToRGB_<mod>(const Ipp8u* const pSrc[3],
      int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_P3C3R    8u_P3AC4R
```

事例 5：プレーン順序データからピクセル順序データへの変換（ROI を使用しない場合）

```
IppStatus ippiYUV422ToRGB_8u_P3C3( const Ipp8u* const pSrc[3],
      Ipp8u* pDst, IppiSize imgSize);
```

引数

pSrc	（ピクセル順序データの場合）ソース・イメージへのポインタ （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	（ROI を使用する処理の場合）デスティネーション・イメージ・バッファ内のステップ（バイト単位）。プレーン・イメージの場合は、3 つのステップ値の配列
pDst	（ピクセル順序データの場合）デスティネーション・イメージへのポインタ （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	（ROI を使用する処理の場合）デスティネーション・イメージ・バッファ内のステップ （バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
imgSize	（ROI を使用しない処理の場合）ソース・イメージとデスティネーション・イメージのサイズ （ピクセル単位）

説明

関数 ippiYUV422ToRGB は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiYUVToRGB と同じ公式に従って、Y'U'V' 画像 *pSrc* をガンマ補正された R'G'B' 画像 *pDst* に変換する。相違点は、ippiYUV422ToRGB では、入力データが 4:2: サンプリング形式とされることである（詳細は、表 6-2 および表 6-3 を参照）。

関数 ippiYUV422ToRGB_P3AC4R はさらに、アルファ値を 0 に設定してデスティネーション・イメージの中にアルファ・チャネルを作成する。

ROI を使用しない関数タイプは、ソース・イメージとデスティネーション・イメージが同じサイズであり、連続するメモリ領域を占める （つまり、画像の行が 0 で埋められていない）ことを前提として機能する。この場合は、ステップ・パラメータは不要である。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。

ippStsSizeErr	*roiSize* または *imgSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

RGBToYUV420

RGB 画像を YUV カラー・モデルに変換する。4:2:0 サンプリングを使用する。

事例 1：プレーン・データの操作 （ROI を使用する場合）

```
IppStatus ippiRGBToYUV420_8u_P3R(const Ipp8u* const pSrc[3], int
     srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

事例 2：プレーン・データの操作 （ROI を使用しない場合）

```
IppStatus ippiRGBToYUV420_8u_P3(const Ipp8u* const pSrc[3], Ipp8u*
     pDst[3], IppiSize imgSize);
```

事例 3：ピクセル順序データからプレーン・データへの変換 （ROI を使用する場合）

```
IppStatus ippiRGBToYUV420_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
     Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

事例 4：ピクセル順序データからプレーン・データへの変換 （ROI を使用しない場合）

```
IppStatus ippiRGBToYUV420_8u_C3P3(const Ipp8u* pSrc, Ipp8u*
     pDst[3],
     IppiSize imgSize);
```

引数

pSrc	（ピクセル順序データの場合）ソース・イメージへのポインタ （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	（ROI を使用する処理の場合）ソース・イメージ・バッファ内のステップ（バイト単位）

pDst	デスティネーション・イメージ内のカラー・プレーンを区切るポインタの配列
dstStep	(ROI を使用する処理の場合) デスティネーション・イメージ・バッファ内の 3 つのステップ値の配列 (バイト単位)
roiSize	ソース ROI とデスティネーション ROI のサイズ (ピクセル単位)
imgSize	(ROI を使用しない処理の場合) ソース・イメージとデスティネーション・イメージのサイズ (ピクセル単位)

説明

関数 ippiRGBToYUV420 は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiRGBToYUV と同じ 公式に従って、ガンマ補正された R'G'B' 画像 *pSrc* を Y'U'V' 画像 *pDst* に変換する。相違点は、ippiRGBToYUV420 では、変換された画像に 4:2:0 サンプリング形式が使用されることである (詳細は、表 6-3 を参照)。

[0..255] の範囲のデジタル RGB 値の場合、Y' の範囲は [0..255] であり、U は [-112..+112] の範囲内で変化し、V は [-157..+157] の範囲内で変化する。
結果を [0..255] の範囲内に収めるために、計算された U と V の値に定数 128 を加算し、V を飽和処理する。
ROI を使用しない関数タイプは、ソース・イメージとデスティネーション・イメージが同じサイズであり、連続するメモリ領域を占める (つまり、画像の行が 0 で埋められていない) ことを前提として機能する。この場合は、ステップ・パラメータは不要である。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* または *imgSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YUV420ToRGB

4:2:0 サンプリング形式の YUV 画像を RGB カラー・モデルに変換する。

事例 1：プレーン・データの操作 （ROI を使用する場合）

```
IppStatus ippiYUV420ToRGB_8u_P3R(const Ipp8u* const pSrc[3],
     int srcStep[3], Ipp8u* pDst[3], int dstStep, IppiSize
     roiSize);
```

事例 2：プレーン・データの操作 （ROI を使用しない場合）

```
IppStatus ippiYUV420ToRGB_8u_P3(const Ipp8u* const pSrc[3],
     Ipp8u* pDst[3], IppiSize imgSize);
```

事例 3：プレーン順序データからピクセル順序データへの変換 （ROI を使用する場合）

```
IppStatus ippiYUV420ToRGB_<mod>(const Ipp8u* const pSrc[3],
     int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_P3C3R    8u_P3AC4R
```

事例 4：プレーン順序データからピクセル順序データへの変換 （ROI を使用しない場合）

```
IppStatus ippiYUV420ToRGB_8u_P3C3(const Ipp8u* const pSrc[3],
     Ipp8u* pDst, IppiSize imgSize);
```

引数

pSrc	ソース・イメージ内のカラー・プレーンを区切るポインタの配列
srcStep	（ROI を使用する処理の場合）ソース・イメージ・バッファ内の 3 つのステップ値の配列 （バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・イメージへのポインタ （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	（ROI を使用する処理の場合）ソース・イメージ・バッファ内のステップ（バイト単位）

roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
imgSize	（ROI を使用しない処理の場合）ソース・イメージとデスティネーション・イメージのサイズ （ピクセル単位）

説明

関数 ippiYUV420ToRGB は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiYUVToRGB と同じ公式に従って、Y'U'V' 画像 *pSrc* をガンマ補正された R'G'B' 画像 *pDst* に変換する。相違点は、ippiYUV420ToRGB では、入力データが 4:2:0 サンプリング形式とされることである （詳細は、表 6-3 を参照）。

関数 ippiYUV420ToRGB_P3AC4R はさらに、アルファ値を 0 に設定してデスティネーション・イメージのアルファ・チャネルを作成する。

ROI を使用しない関数タイプは、ソース・イメージとデスティネーション・イメージが同じサイズであり、連続するメモリ領域を占める （つまり、画像の行が 0 で埋められていない）ことを前提として機能する。この場合は、ステップ・パラメータは不要である。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* または *imgSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YUV420ToRGB565, YUV420ToRGB555, YUV420ToRGB444

4:2:0 サンプリング形式の YUV 画像を ピクセル当たり 16 ビットの RGB 画像に変換する。

```
IppStatus ippiYUV420ToRGB565_8u16u_P3C3R(const Ipp8u* const
    pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToRGB555_8u16u_P3C3R(const Ipp8u* const
    pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToRGB444_8u16u_P3C3R(const Ipp8u* const
    pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	ソース・イメージ内のカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内の 3 つのステップ値の配列 （バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYUV420ToRGB444、ippiYUV420ToRGB555、ippiYUV420ToRGB565 は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiYUVToRGB と同じ 公式 に従って、Y'U'V' 画像 *pSrc* をガンマ補正された R'G'B' 画像 *pDst* に変換する。相違点は、入力データが 4:2: サンプリング形式となることである （詳細は、表 6-3 を参照）。デスティネーション・イメージ *pDst* の色分解能は、16 ビット / ピクセルに減らされる。デスティネーション・イメージは、RGB565 （赤に 5 ビット、緑に 6 ビット、青に 5 ビット）、RGB555 （赤、緑、青に 5 ビット）、または RGB444 （赤、緑、青に 4 ビット）の 3 つの可能なフォーマットのうち 1 つである（図 6-13 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YUV420ToBGR565, YUV420ToBGR555, YUV420ToBGR444

4:2:0 サンプリング形式の YUV 画像を ピクセル当たり 16 ビットの RGB 画像に変換する

```
IppStatus ippiYUV420ToBGR565_8u16u_P3C3R(const Ipp8u* const
    pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToBGR555_8u16u_P3C3R(const Ipp8u* const
    pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToBGR444_8u16u_P3C3R(const Ipp8u* const
    pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	ソース・イメージ内のカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内の 3 つのステップ値の配列 （バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYUV420ToBGR565、ippiYUV420ToBGR555、ippiYUV420ToBGR444 は、ippi.h ファイルの中で宣言される。これらの関数は、関数 ippiYUVToRGB と同じ公式に従って、Y'U'V' 画像 *pSrc* をガンマ補正された B'G'R' 画像 *pDst* に変換する。相違点は、入力データに 4:2: サンプリング形式を使用することである （詳細は、表 6-3 を参照）。デスティネーション・イメージ *pDst* の色分解能は、16 ビット / ピクセルに減らされる。デスティネーション・イメージは、BGR565 （青に 5 ビット、緑に 6 ビット、赤に 5 ビット）、BGR555 （青、緑、赤に 5 ビット）、または BGR444（青、緑、赤に 4 ビット）の 3 つの可能なフォーマットのうち 1 つである（図 6-13 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

RGBToYCbCr

RGB 画像を YCbCr カラー・モデルに変換する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiRGBToYCbCr_<mod>(const Ipp8u* pSrc, int srcStep,
      Ipp8u*
      pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R
8u_AC4R
```

事例 2：プレーン・データの操作

```
IppStatus ippiRGBToYCbCr_8u_P3R(const Ipp8u* const pSrc[3], int
      srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

引数

pSrc	（ピクセル順序データの場合）ソース・バッファへのポインタ （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRGBToYCbCr は、ippi.h ファイルの中で宣言される。この関数は、次の公式に従って、[0..255] の範囲内の値を持つガンマ補正された R'G'B' 画像 *pSrc* を Y'Cb'Cr' 画像 *pDst* に変換する。

```
Y'  =   0.257*R' + 0.504*G' + 0.098*B' + 16
Cb' = -0.148*R' - 0.291*G' + 0.439*B' + 128
Cr' =  0.439*R' - 0.368*G' - 0.071*B' + 128
```

YCbCr モデルでは、Y の公称範囲は [16..235] と定義され、Cb と Cr の範囲は [16..240] と定義される。128 の値が 0 に対応する。

ソース・イメージとデスティネーション・イメージは、同じ色分解能を持つ。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCrToRGB

YCbCr 画像を RGB カラー・モデルに変換する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiYCbCrToRGB_<mod>(const Ipp8u* pSrc, int srcStep,
      Ipp8u*
      pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      8u_AC4R
```

事例 2：プレーン・データの操作

```
IppStatus ippiYCbCrToRGB_<mod>(const Ipp8u* const pSrc[3], int
      srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_P3R      8u_P3C3R
```

引数

pSrc	（ピクセル順序データの場合）ソース・バッファへのポインタ （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYCbCrToRGB は、ippi.h ファイルの中で宣言される。この関数は、Y'Cb'Cr' 画像 *pSrc* をガンマ補正された 24 ビット R'G'B' 画像 *pDst* に変換する。変換には次の公式が使用される。

```
R' = 1.164*(Y' - 16) + 1.596*(Cr' - 128)
G' = 1.164*(Y' - 16) - 0.813*(Cr' - 128) - 0.392*(Cb' - 128)
B' = 1.164*(Y' - 16) + 2.017*(Cb' - 128)
```

出力される R'G'B' 値は、[0..255] の範囲に合わせて飽和処理される。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。

ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCrToRGB565, YCbCrToRGB555, YCbCrToRGB444

YCbCr 画像を ピクセル当たり 16 ビットの RGB 画像に変換する

```
IppStatus ippiYCbCrToRGB565_8u16u_C3R(const Ipp8u* pSrc, int
    srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB555_8u16u_C3R(const Ipp8u* pSrc, int
    srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB444_8u16u_C3R(const Ipp8u* pSrc, int
    srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYCbCrToRGB565、ippiYCbCrToRGB555、ippiYCbCrToRGB444 は、ippi.h ファイルの中で宣言される。これらの関数は、関数 ippiYCbCrToRGB と同じ公式に従って、Y'Cb'Cr' 画像 *pSrc* をガンマ補正された R'G'B' 画像 *pDst* に変換する。
デスティネーション・イメージ *pDst* の色分解能は、16 ビット / ピクセルに減らされる。デスティネーション・イメージは、RGB565 （赤に 5 ビット、緑に 6 ビット、青に 5 ビット）、RGB555 （赤、緑、青に 5 ビット）、または RGB444 （赤、緑、青に 4 ビット）の 3 つの可能なフォーマットのうち 1 つである （詳細は、図 6-13 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCrToBGR565, YCbCrToBGR555, YCbCrToBGR444

YCbCr 画像を ピクセル当たり 16 ビットの BGR 画像に変換する

```
IppStatus ippiYCbCrToBGR565_8u16u_C3R(const Ipp8u* pSrc, int
    srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR555_8u16u_C3R(const Ipp8u* pSrc, int
    srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR444_8u16u_C3R(const Ipp8u* pSrc, int
    srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYCbCrToBGR565、ippiYCbCrToBGR555、ippiYCbCrToBGR444 は、ippi.h ファイルの中で宣言される。これらの関数は、関数 ippiYCbCrToRGB と同じ公式 に従って、Y'Cb'Cr' 画像 *pSrc* をガンマ補正された B'G'R' 画像 *pDst* に変換する。
デスティネーション・イメージ *pDst* の色分解能は、16 ビット / ピクセルに減らされる。デスティネーション・イメージは、BGR565 （青に 5 ビット、緑に 6 ビット、赤に 5 ビット）、BGR555 （青、緑、赤に 5 ビット）、または BGR444 （青、緑、赤に 4 ビット）の 3 つの可能なフォーマットのうち 1 つである （図 6-13 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

RGBToYCbCr422

ピクセル当たり 24 ビットの RGB 画像をピクセル当たり 16 ビットの YCbCr 画像に変換する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiRGBToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int
     srcStep,
     Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

事例 2：プレーン・データの操作

```
IppStatus ippiRGBToYCbCr422_8u_P3C2R(const Ipp8u* const
     pSrc[3], int
     srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	（ピクセル順序データの場合）ソース・バッファへのポインタ （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRGBToYCbCr422 は、ippi.h ファイルの中で宣言される。この関数は、次の公式に従って、ガンマ補正された R'G'B' 画像 *pSrc* を Y'Cb'Cr' 画像 *pDst* に変換する。相違点は、ippiRGBToYCbCr422 では、変換された画像に 4:2: サンプリング形式が使用されることである （詳細は、表 6-2 および表 6-3 を参照）。

変換されたバッファの色分解能は、16 ビット / ピクセルに減らされる。ソース・バッファの色分解能は 24 ビットである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCr422ToRGB

ピクセル当たり 16 ビットの YCbCr 画像をピクセル当たり 24 ビットの RGB 画像に変換する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiYCbCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int
      srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

事例 2：ピクセル順序データからプレーン・データへの変換

```
IppStatus ippiYCbCr422ToRGB_8u_C2P3R(const Ipp8u* pSrc, int
      srcStep,
      Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

事例 3：プレーン順序データからピクセル順序データへの変換

```
IppStatus ippiYCbCr422ToRGB_8u_P3C3R(const Ipp8u* const
     pSrc[3], int
     srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	（ピクセル順序データの場合）ソース・バッファへのポインタ （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYCbCr422ToRGB は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiYCbCrToRGB と同じ公式に従って、4:2: サンプリング形式（詳細は表 6-2 と表 6-3を参照）でパックされた Y'Cb'Cr' 画像 *pSrc* を、ガンマ補正された24ビット R'G'B' 画像 *pDst* に変換する。
出力される R'G'B' 値は、[0..255] の範囲に合わせて飽和処理される。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

RGBToCbYCr422 RGBToCbYCr422Gamma

ピクセル当たり 24 ビットの RGB 画像をピクセル当たり 16 ビットの CbYCr 画像に変換する。

```
IppStatus ippiRGBToCbYCr422_8u_C3C2R(const Ipp8u* pSrc, int
      srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToCYbCr422Gamma_8u_C3C2R(const Ipp8u* pSrc,
      int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRGBToCbYCr422 は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiRGBToYCbC と同じ公式に従って、ガンマ補正された R'G'B' 画像 *pSrc* を Cb'Y'Cr' 画像 *pDst* に変換する。
関数 ippiRGBToCbYCr422Gamma は、関数 ippiGammaFwd と同じ公式に従って、ソース RGB 画像 *pSrc* のガンマ補正を実行した後、関数 ippiRGBToYCbCr と同じ公式に従って、その結果を Cb'Y'Cr' 画像 *pDst* に変換する。
関数 ippiRGBToCbYCr422 と ippiRGBToCbYCr422Gamma は、変換された画像に 4:2: サンプリング形式を使用する。

CbYCr 画像は、以下のバイト・シーケンスを持つ。Cb0Y0Cr0Y1, Cb1Y2Cr1Y3,....

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

CbYCr422ToRGB

ピクセル当たり 16 ビットの CbYCr 画像をピクセル当たり 24 ビットの RGB 画像に変換する。

```
IppStatus ippiCbYCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int
      srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiCbYCr422ToRGB は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiYCbCrToRGB と同じ公式に従って、4:2: サンプリング形式でパックされた Cb'Y'Cr' 画像 *pSrc* を、ガンマ補正された 24 ビット R'G'B' 画像 *pDst* に変換する。
CbYCr 画像は、以下のバイト・シーケンスを持つ。Cb0Y0Cr0Y1, Cb1Y2Cr1Y3,....

出力される R'G'B' 値は、[0..255] の範囲に合わせて飽和処理される。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCr422ToRGB565, YCbCr422ToRGB555, YCbCr422ToRGB444

ピクセル当たり 16 ビットの YCbCr 画像をピクセル当たり 16 ビットの RGB 画像に変換する。

```
IppStatus ippiYCbCr422ToRGB565_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB555_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB444_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）

roiSize ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYCbCr422ToRGB565、ippiYCbCr422ToRGB565、ippiYCbCr422ToRGB565 は、ippi.h ファイルの中で宣言される。これらの関数は、関数 ippiYCbCrToRGB と同じ公式に従って、4:2: サンプリング形式（詳細は表 6-2 を参照）でパックされた Y'Cb'Cr' 画像 *pSrc* を、ガンマ補正された R'G'B' 画像 *pDst* に変換する。デスティネーション・イメージ *pDst* の色分解能は、16 ビット / ピクセルに減らされる。デスティネーション・イメージは、RGB565（赤に 5 ビット、緑に 6 ビット、青に 5 ビット）、RGB555（赤、緑、青に 5 ビット）、RGB444（赤、緑、青に 4 ビット）の 3 つの可能なフォーマットのうち 1 つである（図 6-13 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCr422ToBGR565, YCbCr422ToBGR555, YCbCr422ToBGR444

ピクセル当たり 16 ビットの YCbCr 画像をピクセル当たり 16 ビットの BGR 画像に変換する。

```
IppStatus ippiYCbCr422ToBGR565_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR555_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR444_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYCbCr422ToBGR565、ippiYCbCr422ToBGR565、ippiYCbCr422ToBGR565 は、ippi.h ファイルの中で宣言される。これらの関数は、関数 ippiYCbCrToRGB と同じ公式に従って、4:2: サンプリング形式（詳細は表 6-2 を参照）でパックされた Y'Cb'Cr' 画像 *pSrc* を、ガンマ補正された B'G'R' 画像 *pDst* に変換する。
デスティネーション・イメージ *pDst* の色分解能は、16 ビット / ピクセルに減らされる。デスティネーション・イメージは、BGR565 （青に 5 ビット、緑に 6 ビット、赤に 5 ビット）、BGR555 （青、緑、赤に 5 ビット）、BGR444 （青、緑、赤に 4 ビット）の 3 つの可能なフォーマットのうち 1 つである （図 6-13 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

RGBToYCbCr420

RGB 画像を YCbCr カラー・モデルに変換する。4:2:0 サンプリングを使用する。

```
IppStatus ippiRGBToYCbCr420_8u_C3P3R,(const Ipp8u* pSrc, int
    srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRGBToYCbCr420 は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiRGBToYCbCr と同じ公式に従って、ガンマ補正された R'G'B' 画像 *pSrc* を Y'Cb'Cr' 画像 *pDst* に変換する。相違点は、ippiRGBToYCbCr420 では、変換された画像に4:2:サンプリング形式が使用されることである（詳細は、表 6-3を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCr420ToRGB, YCbCr420ToBGR

4:2:0 サンプリング形式の YCbCr 画像を RGB/BGR カラー・モデルに変換する。

```
IppStatus ippiYCbCr420ToRGB_8u_P3C3R,(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToBGR_8u_P3C3R,(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	ソースのカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYCbCr420ToRGB、ippiYCbCr420ToBGR は、ippi.h ファイルの中で宣言される。これらの関数は、関数 ippiYCbCrToRGB と同じ公式に従って、Y'Cb'Cr' 画像 *pSrc* をガンマ補正された R'G'B' （B'G'R'）画像 *pDst* に変換する。相違点は、関数 ippiYCbCr420ToRGB は、4:2: サンプリング形式の入力データを使用することである。このサンプリング形式では、Cb サンプルと Cr サンプルの数は、垂直方向と水平方向に 1/2 に減らされる （詳細は表 6-3 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCr411ToBGR

4:1:1 サンプリング形式の YCbCr 画像を
RGB カラー・モデルに変換する。

```
IppStatus ippiYCbCr411ToBGR_8u_P3C3R,(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	ソースのカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYCbCr411ToBGR は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiYCbCrToRGB と同じ公式に従って、Y'Cb'Cr' 画像 *pSrc* をガンマ補正された R'G'B' （B'G'R'）画像 *pDst* に変換する。相違点は、ippiYCbCr411ToRGB では、入力データが 4:1:サンプリング形式とされることである（詳細は、表 6-3 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

RGBToXYZ

RGB 画像を XYZ カラー・モデルに変換する。

```
IppStatus ippiRGBToXYZ_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース ROI バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション ROI バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRGBToXYZ は、ippi.h ファイルの中で宣言される。この関数は、次の基本的な式に従って、RGB 画像 *pSrc* を CIE XYZ 画像 *pDst* に変換する。

```
X =  0.412453*R + 0.35758 *G + 0.180423*B
Y =  0.212671*R + 0.71516 *G + 0.072169*B
Z =  0.019334*R + 0.119193*G + 0.950227*B
```

上の式は、R、G、B の値が [0..1] の範囲に合わせて正規化されていることを前提としている。浮動小数点データ・タイプの場合は、入力の RGB 値はすでに [0..1] の範囲内に収まっていなければならない。整数データ・タイプの場合は、変換関数が正規化を内部で実行する。
計算された XYZ 値が [0..1] の範囲から外れる場合、その値は飽和処理される。
整数関数タイプの場合、求められた値は、デスティネーションのデータ・タイプの全範囲に合わせてスケーリングされる （第 2 章の表 2-2 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

XYZToRGB

XYZ 画像を RGB カラー・モデルに変換する。

```
IppStatus ippiXYZToRGB_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16u_C3R     16s_C3R     32f_C3R
8u_AC4R     16u_AC4R    16s_AC4R    32f_AC4R
```

引数

pSrc	ソース ROI バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション ROI バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiXYZToRGB は、ippi.h ファイルの中で宣言される。この関数は、次の基本的な式に従って、CIE XYZ 画像 *pSrc* を RGB 画像 *pDst* に変換する。

```
R =  3.240479 * X - 1.53715  * Y  - 0.498535 * Z
G = -0.969256 * X + 1.875991 * Y  + 0.041556 * Z
B =  0.055648 * X - 0.204043 * Y  + 1.057311 * Z
```

上の式は、X、Y、Z の値が [0..1] の範囲内にあることを前提としている。浮動小数点データ・タイプの場合は、入力の XYZ 値はすでに [0..1] の範囲内に収まっていなければならない。整数データ・タイプの場合は、変換関数が正規化を内部で実行する。
計算された RGB 値が [0..1] の範囲から外れる場合、その値は飽和処理される。
整数関数タイプの場合、求められた値は、デスティネーションのデータ・タイプの全範囲に合わせてスケーリングされる （第 2 章の表 2-2 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

RGBToLUV

RGB 画像を LUV カラー・モデルに変換する。

```
IppStatus ippiRGBToLUV_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース ROI バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション ROI バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRGBToLUV は、ippi.h ファイルの中で宣言される。この関数は、RGB 画像 *pSrc* から CIE LUV 画像 *pDst* への変換を 2 段階で実行する。最初に、ippiRGBToXYZ 関数について定義された式を使用して、CIE XYZ 形式への変換を実行する。その後、次の式に従って、LUV 画像への変換を実行する。

$$L = 116. * (Y/Y_n)^{1/3}. - 16.$$
$$U = 13. * L * (u - u_n)$$
$$V = 13. * L * (v - v_n)$$

ここで、

$$u = 4.*X / (X + 15.*Y + 3.*Z)$$
$$v = 9.*Y / (X + 15.*Y + 3.*Z)$$
$$un = 4.*x_n / (-2.*x_n + 12.*y_n + 3.)$$
$$vn = 9.*y_n / (-2.*x_n + 12.*y_n + 3.)$$

ここで、$x_n = 0.312713$、$y_n = 0.329016$ は D65 ホワイト・ポイントの CIE 色度座標、$Y_n = 1.0$ は D65 ホワイト・ポイントの輝度である。
計算された L 成分の値の範囲は [0..100]、U 成分の範囲は [-134..220]、V 成分の範囲は [-140..122] である。

上の式は、R、G、B の値が [0..1] の範囲に合わせて正規化されていることを前提としている。浮動小数点データ・タイプの場合は、入力の RGB 値はすでに [0..1] の範囲内に収まっていなければならない。整数データ・タイプの場合は、変換関数が正規化を内部で実行する。
8u データ・タイプの場合は、次のように、計算された L、U、V の値は量子化され、[0..IPP_MAX_8U] の範囲内に収まるように変換される。

```
L =   L * IPP_MAX_8U / 100.
U = (U + 134.) * IPP_MAX_8U / 354.
V = (V + 140.) * IPP_MAX_8U / 256.
```

16u データ・タイプの場合は、次のように、計算された L、U、V の値は量子化され、[0..IPP_MAX_16U] の範囲内に収まるように変換される。

```
L =   L * IPP_MAX_16U / 100.
U = (U + 134.) * IPP_MAX_16U / 354.
V = (V + 140.) * IPP_MAX_16U / 256.
```

16s データ・タイプの場合は、次のように、計算された L、U、V の値は量子化され、[IPP_MIN_16S..IPP_MAX_16S] の範囲内に収まるように変換される。

```
L =   L * IPP_MAX_16U / 100. + IPP_MIN_16S
U = (U + 134.) * IPP_MAX_16U / 354. + IPP_MIN_16S
V = (V + 140.) * IPP_MAX_16U / 256. + IPP_MIN_16S
```

32f データ・タイプの場合は、追加の変換は行われず、L、U、V 成分の範囲は、それぞれ [0..100]、[-134..220]、[-140..122] のままになる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

LUVToRGB

LUV 画像を RGB カラー・モデルに変換する。

```
IppStatus ippiLUVToRGB_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース ROI バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション ROI バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiLUVToRGB は、ippi.h ファイルの中で宣言される。この関数は、CIE LUV 画像 *pSrc* から RGB 画像 *pDst* への変換を 2 段階で実行する。最初に、CIE XYZ 形式への変換を実行する。

これを行うために、LUV の成分が元の範囲に戻される。各データ・タイプについて、この処理は次の方法で実行される。

8u データ・タイプの場合

```
L  =   L * 100./ IPP_MAX_8U
U  = (U * 354./ IPP_MAX_8U) - 134.
V  = (V * 256./ IPP_MAX_8U) - 140.
```

16u データ・タイプの場合

```
L  =   L * 100./ IPP_MAX_16U
U  = (U * 354./ IPP_MAX_16U) - 134.
V  = (V * 256./ IPP_MAX_16U) - 140.
```

16s データ・タイプの場合

```
L  = (L - IPP_MIN_16S)* 100./ IPP_MAX_16U
U  = ((U - IPP_MIN_16S)* 354./ IPP_MAX_16U) - 134.
V  = ((V - IPP_MIN_16S) * 256./ IPP_MAX_16U) - 140.
```

その後、XYZ 形式への変換が次のように行われる。

$Y = Y_n * ((L + 16.) / 116.)**3.$

$X = -9.* Y * u / ((u - 4.)* v - u * v)$

$Z = (9.* Y - 15*v*Y - v*X) / 3. * v$

ここで、

$u = U / (13.* L) + u_n$
$v = V / (13.* L) + v_n$

また

$u_n = 4.*x_n / (-2.*x_n + 12.*y_n + 3.)$
$v_n = 9.*y_n / (-2.*x_n + 12.*y_n + 3.)$

ここで、$x_n = 0.312713$、$y_n = 0.329016$ は D65 ホワイト・ポイントの CIE 色度座標、$Y_n = 1.0$ は D65 ホワイト・ポイントの輝度である。
この中間変換の実行後、得られた XYZ 画像は、ippiXYZToRGB 関数について定義された式を使用して、デスティネーションの RGB 形式に変換される。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

RGBToYCC

RGB 画像を YCC カラー・モデルに変換する。

```
IppStatus ippiRGBToYCC_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース ROI バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション ROI バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRGBToYCC は、ippi.h ファイルの中で宣言される。この関数は、次の基本的な式に従って、ガンマ補正された R'G'B' 画像 *pSrc* を PhotoY'C'C' 画像 *pDst* に変換する。

```
Y'  =  0.299*R' + 0.587*G' + 0.114*B'
C1' = -0.299*R' - 0.587*G' + 0.886*B' = B'- Y
C2' =  0.701*R' - 0.587*G' - 0.114*B' = R'- Y
```

上の式は、R'、G'、B' の値が [0..1] の範囲に合わせて正規化されていることを前提としている。浮動小数点データ・タイプの場合は、入力の R'G'B' 値はすでに [0..1] の範囲内に収まっていなければならない。整数データ・タイプの場合は、変換関数が正規化を内部で実行する。

計算された Y'、C1'、C2' の値は、次のように量子化され、[0..1] の範囲内に収まるように変換される。

```
Y'  = 1. / 1.402 * Y'
C1' = 111.4 / 255. * C1' + 156. / 255.
C2' = 135.64 /255. * C2' + 137. / 255.
```

整数関数タイプの場合、求められた値は、デスティネーションのデータ・タイプの全範囲に合わせてスケーリングされる（第 2 章の表 2-2 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YCCToRGB

YCC 画像を RGB カラー・モデルに変換する。

```
IppStatus ippiYCCToRGB_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース ROI バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション ROI バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYCCToRGB は、ippi.h ファイルの中で宣言される。この関数は、PhotoY'C'C' 画像 *pSrc* を R'G'B' 画像 *pDst* に変換する。関数 ippiYCCToRGB は、最初に通常の輝度と色度のデータを次のように復元する。

```
Y'  = 1.3584 * Y'
C1' = 2.2179 * (C1' - 156./255.)
C2' = 1.8215 * (C2' - 137./255.)
```

上の式は、ソースの Y、C1、C2 の値が [0..1] の範囲に合わせて正規化されていることを前提としている。浮動小数点データ・タイプの場合は、入力の YCC 値はすでに [0..1] の範囲内に収まっていなければならない。整数データ・タイプの場合は、変換関数が正規化を内部で実行する。

その後、次の基本的な式に従って、YCC データが RGB 形式に変換される。

```
R' = Y' + C2'
G' = Y' - 0.194*C1' - 0.509*C2'
B' = Y' + C1'
```

整数関数タイプの場合、計算された R'G'B' 値は、デスティネーションのデータ・タイプの全範囲に合わせてスケーリングされる （第 2 章の表 2-2 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

RGBToHLS

RGB 画像を HLS カラー・モデルに変換する。

```
IppStatus ippiRGBToHLS_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース ROI バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション ROI バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRGBToHLS は、ippi.h ファイルの中で宣言される。この関数は、R'G'B' 画像 *pSrc* を HLS 画像 *pDst* に変換する。浮動小数点データを操作する関数タイプの場合、ソースの RGB 値は [0..1] の範囲内に収まっていなければならない。

RGB から HLS への変換アルゴリズムは、疑似コードでは次のように表現される。

```
// Lightness:
      M1 = max(R,G,B); M2 = max(R,G,B); L = (M1+M2)/2
      // Saturation:
      if M1 = M2 then // achromatics case
          S = 0
          H = 0
      else // chromatics case
          if L <= 0.5 then
               S = (M1-M2) / (M1+M2)
          else
               S = (M1-M2) / (2-M1-M2)
      // Hue:
      Cr = (M1-R) / (M1-M2)
```

```
Cg = (M1-G) / (M1-M2)
Cb = (M1-B) / (M1-M2)
if R = M2 then H = Cb - Cg
if G = M2 then H = 2 + Cr - Cb
if B = M2 then H = 4 + Cg - Cr
H = 60*H
if H < 0 then H = H + 360
```

浮動小数点関数タイプの場合、計算された H、L、S の値は [0..1] の範囲に合わせてスケーリングされる。整数関数タイプの場合、求められた値は、デスティネーションのデータ・タイプの全範囲に合わせてスケーリングされる（第2章の表 2-2を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

HLSToRGB

HLS 画像を RGB カラー・モデルに変換する。

```
IppStatus ippiHLSToRGB_<mod>(const Ipp<datatype>* pSrc, int
     srcStep,
     Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース ROI バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション ROI バッファへのポインタ

dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiHLSToRGB は、ippi.h ファイルの中で宣言される。この関数は、HLS 画像 *pSrc* を R'G'B' 画像 *pDst* に変換する。浮動小数点データを操作する関数タイプの場合、ソースの HLS 値は [0..1] の範囲内に収まっていなければならない。HLS から RGB への変換アルゴリズムは、疑似コードでは次のように表現される。

```
if L <= 0.5 then
          M2 = L *(1 + S)
     else
          M2 = L + S - L * S
     M1 = 2 * L - M2
     if S = 0 then
        R = G = B = L
     else
          h = H + 120
          if h > 360 then
               h = h - 360
          if h  <  60 then
               R = ( M1 + ( M2 - M1 ) * h / 60)
          else if h < 180 then
               R = M2
          else if h < 240 then
               R = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
          else
               R = M1
          h = H
          if h  <  60 then
               G = ( M1 + ( M2 - M1 ) * h / 60
          else if h < 180 then
               G = M2
          else if h < 240 then
               G = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
          else
               G  = M1
          h = H - 120
          if h < 0 then
               h += 360
          if h  <  60 then
               B = ( M1 + ( M2 - M1 ) * h / 60
          else if h < 180 then
               B = M2
          else if h < 240 then
```

```
            B = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
      else
            B = M1
```

浮動小数点関数タイプの場合、計算された R', G', B' の値は [0..1] の範囲に合わせてスケーリングされる。整数関数タイプの場合、求められた値は、デスティネーションのデータ・タイプの全範囲に合わせてスケーリングされる （第 2 章の表 2-2 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

BGRToHLS

BGR 画像を HLS カラー・モデルに変換する。

```
IppStatus ippiBGRToHLS_8u_AC4R(const Ipp8u* pSrc, int srcStep,
      Ipp8u*
      pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_AC4P4R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_P3C3R(const Ipp8u* const pSrc[3],
      int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_AP4C4R(const Ipp8u* const pSrc[4],
      int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_P3R(const Ipp8u* const pSrc[3], int
      srcStep,
      Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToHLS_8u_AP4R(const Ipp8u* const pSrc[4], int
      srcStep,
      Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
```

引数

pSrc	（ピクセル順序データの場合）ソース・バッファへのポインタ （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiBGRToHLS は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiRGBToHLS と同じ公式に従って、B'G'R' 画像 *pSrc* を HLS 画像 *pDst* に変換する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

HLSToBGR

HLS 画像を RGB カラー・モデルに変換する。

```
IppStatus ippiHLSToBGR_8u_C3P3R(const Ipp8u* pSrc, int
      srcStep, Ipp8u*
      pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_AC4P4R(const Ipp8u* pSrc, int
      srcStep, Ipp8u*
      pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_AP4R(const Ipp8u* const pSrc[4], int
      srcStep,
      Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_P3R(const Ipp8u* const pSrc[3], int
      srcStep,
      Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_AP4C4R(const Ipp8u* const pSrc[4],
      int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_P3C3R(const Ipp8u* const pSrc[3],
      int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	（ピクセル順序データの場合）ソース・バッファへのポインタ （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiHLSToBGR は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiHLSToRGB と同じ公式に従って、HLS 画像 *pSrc* を B'G'R' 画像 *pDst* に変換する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

RGBToHSV

RGB 画像を HSV カラー・モデルに変換する。

```
IppStatus ippiRGBToHSV_<mod>(const Ipp<datatype>* pSrc, int
    srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16u_C3R
8u_AC4R     16u_AC4R
```

引数

pSrc	ソース ROI バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション ROI バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRGBToHSV は、ippi.h ファイルの中で宣言される。この関数は、R'G'B' 画像 *pSrc* を HSV 画像 *pDst* に変換する。
RGB から HSV への変換アルゴリズムは、疑似コードでは次のように表現される。

```
//  Value:
     V = max(R,G,B);
     //  Saturation:
     temp = min(R,G,B);
     if V = 0 then // achromatics case
         S = 0//          H = 0
     else // chromatics case
         S = (V - temp)/V
     //  Hue:
     Cr = (V - R) / (V - temp)
     Cg = (V - G) / (V - temp)
     Cb = (V - B) / (V - temp)
     if R = V then H = Cb - Cg
     if G = V then H = 2 + Cr - Cb
     if B = V then H = 4 + Cg - Cr
     H = 60*H
     if H < 0 then H = H + 360
```

計算された H, S, V 値は、デスティネーションのデータ・タイプの全範囲に合わせてスケーリングされる （第 2 章の表 2-2 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

HSVToRGB

HSV 画像を RGB カラー・モデルに変換する。

```
IppStatus ippiHSVToRGB_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16u_C3R
8u_AC4R     16u_AC4R
```

引数

pSrc	ソース ROI バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション ROI バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiHSVToRGB は、ippi.h ファイルの中で宣言される。この関数は、R'G'B' 画像 *pSrc* を HSV 画像 *pDst* に変換する。
HSV から RGB への変換アルゴリズムは、疑似コードでは次のように表現される。

```
if S = 0 then
        R = G = B = V
     else
         if H = 360 then
             H = 0
         else
             H = H/60
          I = floor(H)
          F = H - I;
          M = V * ( 1 - S);
          N = V * ( 1 - S * F);
          K = V * ( 1 - S * (1 - F));
          if(I == 0)then{ R = V;G = K;B = M;}
          if(I == 1)then{ R = N;G = V;B = M;}
          if(I == 2)then{ R = M;G = V;B = K;}
```

```
if(I == 3)then{ R = M;G = N;B = V;}
if(I == 4)then{ R = K;G = M;B = V;}
if(I == 5)then{ R = V;G = M;B = N;}
```

計算された R', G', B' 値は、デスティネーションのデータ・タイプの全範囲に合わせてスケーリングされる （第 2 章の表 2-2 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

RGBToGray

固定変換係数を使用して、RGB 画像を グレー・スケールに変換する。

```
IppStatus ippiRGBToGray_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3C1R    16u_C3C1R   16s_C3C1R   32f_C3C1R
8u_AC4C1R   16u_AC4C1R  16s_AC4C1R  32f_AC4C1R
```

引数

pSrc	ソース ROI バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション ROI バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）

roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRGBToGray は、ippi.h ファイルの中で宣言される。この関数は、次の基本的な式を使用して、ガンマ補正された赤、緑、青のノンリニア値から輝度を計算する。

```
Y´ = 0.299 * R´ + 0.587 * G´ + 0.114 * B´
```

この変換係数は、赤、緑、青の CRT 蛍光体に関する NTSC の規格に適合している。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

ColorToGray

独自の変換係数を使用して、RGB 画像をグレー・スケールに変換する。

```
IppStatus ippiColorToGray_<mod>(const Ipp<datatype>* pSrc, int
      srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
      roiSize, const Ipp32f coeffs[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3C1R    16u_C3C1R   16s_C3C1R   32f_C3C1R
8u_AC4C1R   16u_AC4C1R  16s_AC4C1R  32f_AC4C1R
```

引数

pSrc	ソース ROI バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション ROI バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
coeffs	変換係数

説明

関数 ippiColorToGray は、ippi.h ファイルの中で宣言される。この関数は、次の式を使用して、RGB 画像をグレー・スケールに変換する。

```
Y = coeffs[0] * R + coeffs[1] * G + coeffs[2] * B
```

coeffs 配列は、ユーザ定義の変換係数を保持する。この係数は、以下の条件を満たす負でない値である。

$coeffs[0] + coeffs[1] + coeffs[2] \leq 1$

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

LUT

輝度変換を適用して、画像をマッピングする。

事例 1：1 チャネル・データの操作

```
IppStatus ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R
8u_C1R	16s_C1R

事例 2：マルチチャネル・データの操作

```
IppStatus ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int
    nLevels[3]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16s_C3R
8u_AC4R	16s_AC4R

```
IppStatus ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int
    nLevels[4]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4R	16s_C4R

事例 3：1 チャネル浮動小数点データの操作

```
IppStatus ippiLUT_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst,
    int dstStep, IppiSize roiSize, const Ipp32f* pValues,
    const Ipp32f* pLevels, int nLevels);
```

事例 4：マルチチャネル浮動小数点データの操作

```
IppStatus ippiLUT_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst,
    int dstStep, IppiSize roiSize, const Ipp32f* pValues[3],
    const Ipp32f* pLevels[3], int nLevels[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiLUT_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst,
    int dstStep, IppiSize roiSize, const Ipp32f* pValues[4], const
    Ipp32f* pLevels[4], int nLevels[4]);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ （ピクセル単位）
pValues	輝度の値の配列へのポインタ。マルチチャネル・データの場合、*pValues* は、各チャネルの輝度の値の配列へのポインタの配列になる。
pLevels	レベル値の配列へのポインタ。マルチチャネル・データの場合、*pLevels* は、各チャネルのレベル値の配列へのポインタの配列になる。
nLevels	レベルの数 （各チャネル別）

説明

関数 ippiLUT は、ippi.h ファイルの中で宣言される。この関数は、配列 *pLevels* と *pValues* で指定されるルックアップ・テーブル （LUT）を使用して、ソース・イメージ *pSrc* の輝度変換を実行する。[*pLevels*[*k*], *pLevels*[*k*+1]] の範囲内の各ソース・ピクセル pSrc （x,y）が、デスティネーション・ピクセル *pDst* （*x*,*y*）にマッピングされる。*pDst* （*x*,*y*）の値は、輝度 *pValues*[*k*] に等しくなる。

pLevels 配列と *pValues* 配列の長さは、*nLevels* パラメータによって定義される。レベル値と輝度の値の数は、*nLevels* より 1 つ少ない。*pSrc* 画像内のピクセルのうち、[*pLevels*[0], *pLevels*[*nLevels*-1]) の範囲から外れているものは、変換されずに *pDst* 画像にコピーされる。
図 6-15 は、ippiLUT 関数のすべての変種でマッピングに使用される曲線を示している。レベル値は、0、64、128、192、256 である。輝度の値は、20、60、160、180、230 である。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタが NULL の場合のエラーを示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsMemAllocErr	内部バッファのメモリが足りない場合のエラーを示す。
ippStsLUTNofLevelsErr	*nLevels* が 2 より小さい場合のエラーを示す。

LUT_Linear

1 次補間による輝度変換を適用して、画像をマッピングする。

事例 1：1 チャネル・データの操作

```
IppStatus ippiLUT_Linear_<mod>(const Ipp<datatype>* pSrc, int
    srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R　　16s_C1R

事例 2：マルチチャネル・データの操作

```
IppStatus ippiLUT_Linear_<mod>(const Ipp<datatype>* pSrc, int
    srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int
    nLevels[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R
8u_AC4R     16s_AC4R
```

```
IppStatus ippiLUT_Linear_<mod>(const Ipp<datatype>* pSrc, int
    srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int
    nLevels[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R
```

事例 3：1 チャネル浮動小数点データの操作

```
IppStatus ippiLUT_Linear_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f*
    pValues,
    const Ipp32f* pLevels, int nLevels);
```

事例 4：マルチチャネル浮動小数点データの操作

```
IppStatus ippiLUT_Linear_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize,
    const Ipp32f* pValues[3], const Ipp32f* pLevels[3], int
    nLevels[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiLUT_Linear_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize,
    const Ipp32f* pValues[4], const Ipp32f* pLevels[4], int
    nLevels[4]);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ （ピクセル単位）

pValues	輝度の値の配列へのポインタ。マルチチャネル・データの場合、*pValues* は、各チャネルの輝度の値の配列へのポインタの配列になる。
pLevels	レベル値の配列へのポインタ。マルチチャネル・データの場合、*pLevels* は、各チャネルのレベル値の配列へのポインタの配列になる。
nLevels	レベルの数 （各チャネル別）

説明

関数 ippiLUT_Linear は、ippi.h ファイルの中で宣言される。この関数は、ルックアップ・テーブル （LUT） を使用して、隣接する 2 つのレベル間の 1 次補間によってソース・イメージ *pSrc* の輝度変換を実行する。LUT は、配列 *pLevels* と *pValues* によって指定される。[*pLevels*[*k*], *pLevels*[*k*+1]] の範囲内の各ソース・ピクセル *pSrc* （*x*,*y*） が、デスティネーション・ピクセル *pDst* （*x*,*y*） にマッピングされる。*pDst* （*x*,*y*） の値は、以下の式に従って計算される。

$$pDst(x,y) = pValues[k] + (pSrc(x,y) - pLevels[k]) \cdot \frac{pValues[k+1] - pValues[k]}{pLevels[k+1] - pLevels[k]}$$

pLevels 配列と *pValues* 配列の長さは、*nLevels* パラメータによって定義される。*pSrc* 画像内のピクセルのうち、[*pLevels*[0], *pLevels*[*nLevels*-1]） の範囲から外れているものは、変換されずに *pDst* 画像にコピーされる。
図 6-15 は、ippiLUT 関数のすべての変種でマッピングに使用される曲線を示している。レベル値は、0、64、128、192、256 である。輝度の値は、20、60、160、180、230 である。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタが NULL の場合のエラーを示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsMemAllocErr	内部バッファのメモリが足りない場合のエラーを示す。
ippStsLUTNofLevelsErr	*nLevels* が 2 より小さい場合のエラーを示す。

LUT_Cubic

3 次補間による輝度変換を適用して、画像
をマッピングする。

事例 1 : 1 チャネル・データの操作

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp<datatype>* pSrc, int
    srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R
```

事例 2 : マルチチャネル・データの操作

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp<datatype>* pSrc, int
    srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int
    nLevels[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R
8u_AC4R     16s_AC4R
```

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp<datatype>* pSrc, int
    srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int
    nLevels[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R
```

事例 3 : 1 チャネル浮動小数点データの操作

```
IppStatus ippiLUT_Cubic_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues,
    const Ipp32f* pLevels, int nLevels);
```

事例 4：マルチチャネル浮動小数点データの操作

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize,
    const Ipp32f* pValues[3], const Ipp32f* pLevels[3], int
    nLevels[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiLUT_Cubic_32f_C4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize,
    const Ipp32f* pValues[4], const Ipp32f* pLevels[4], int
    nLevels[4]);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ （ピクセル単位）
pValues	輝度の値の配列へのポインタ。マルチチャネル・データの場合、*pValues* は、各チャネルの輝度の値の配列へのポインタの配列になる。
pLevels	レベル値の配列へのポインタ。マルチチャネル・データの場合、*pLevels* は、各チャネルのレベル値の配列へのポインタの配列になる。
nLevels	レベルの数 （各チャネル別）

説明

関数 ippiLUT_Cubic は、ippi.h ファイルの中で宣言される。この関数は、ルックアップ・テーブル（LUT）を使用して、隣接するレベル間の 3 次補間によってソース・イメージ *pSrc* の輝度変換を実行する。LUT は、配列 *pLevels* と *pValues* によって指定される。

[*pLevels*[*k*], *pLevels*[*k*+1]] の範囲内の各ソース・ピクセル *pSrc* (x,y) が、デスティネーション・ピクセル *pDst* (x,y) にマッピングされる。*pDst* (x,y) の値は、以下の式に従って計算される。

$$pDst(x,y) = A*pSrc(x,y)^3 + B*pSrc(x,y)^2 + C*pSrc(x,y) + D$$

この関数は、3 次多項式曲線が以下の 4 点を通ることを前提としている。

(*pLevels*[*k*-1], *pValues*[*k*-1]),
(*pLevels*[*k*], *pValues*[*k*]),
(*pLevels*[*k*+1], *pValues*[*k*+1]),
(*pLevels*[*k*+2], *pValues*[*k*+2])

この前提に基づいて、以下の連立 1 次方程式を解くことによって、係数 A、B、C、D が計算される。

$$A*pLevels[k-1]^3 + B*pLevels[k-1]^2 + C*pLevels[k-1] + D = pValues[k-1]$$
$$A*pLevels[k]^3 + B*pLevels[k]^2 + C*pLevels[k] + D = pValues[k]$$
$$A*pLevels[k+1]^3 + B*pLevels[k+1]^2 + C*pLevels[k+1] + D = pValues[k+1]$$
$$A*pLevels[k+2]^3 + B*pLevels[k+2]^2 + C*pLevels[k+2] + D = pValues[k+2]$$

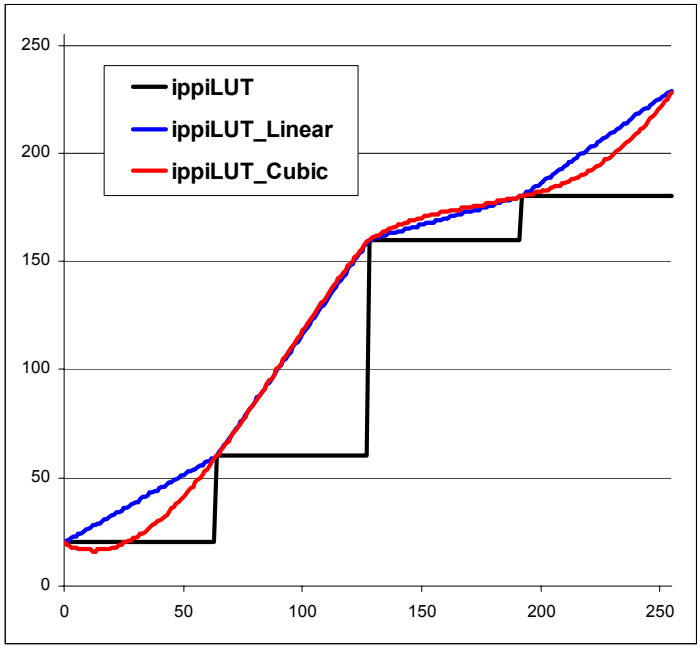
pLevels 配列と *pValues* 配列の長さは、*nLevels* パラメータによって定義される。*pSrc* 画像内のピクセルのうち、[*pLevels*[0], *pLevels*[*nLevels*-1]) の範囲から外れているものは、変換されずに *pDst* 画像にコピーされる。

図 6-15 は、ippiLUT 関数のすべての変種でマッピングに使用される曲線を示している。レベル値は、0、64、128、192、256 である。輝度の値は、20、60、160、180、230 である。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタが NULL の場合のエラーを示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsMemAllocErr	内部バッファのメモリが足りない場合のエラーを示す。
ippStsLUTNofLevelsErr	*nLevels* が 2 より小さい場合のエラーを示す。

図 6-15　ippiLUT 関数の変種で使用されるマッピング曲線の例



ReduceBits

画像の量子化ビット数を減らす。

事例 1：ソースとデスティネーションの色分解能が同じデータの操作

```
IppStatus ippiReduceBits_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
      int noise, IppiDitherType dtype, int levels);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

事例2: ソースとデスティネーションの色分解能が異なるデータの操作

```
IppStatus ippiReduceBits_<mod>(const Ipp<srcDatatype>* pSrc, int
      srcStep,  Ipp<dstDatatype>* pDst, int dstStep, IppiSize
      roiSize,
      int noise, IppiDitherType dtype, int levels);
```

サポートされる *mod* の値は、次のとおりである。

16u8u_C1R	16s8u_C1R	32f8u_C1R	32f16u_C1R	32f16s_C1R
16u8u_C3R	16s8u_C3R	32f8u_C3R	32f16u_C3R	32f16s_C3R
16u8u_C4R	16s8u_C4R	32f8u_C4R	32f16u_C4R	32f16s_C4R
16u8u_AC4R	16s8u_AC4R	32f8u_AC4R	32f16u_AC4R	32f16s_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
noise	追加されるノイズの量を指定する数値。このパラメータは、[0..100] の範囲の割合（%）として設定される。
dtype	使用するディザリングのタイプ。以下のタイプをサポートしている。

ippDitherNone	ディザリングを実行しない。
ippDitherStucki	Stucki 誤差拡散ディザリング・アルゴリズムを使用する。
ippDitherFS	Floy-Steinberg 誤差拡散ディザリング・アルゴリズムを使用する。
ippDitherJJN	Jarvice-Judice-Ninke 誤差拡散ディザリング・アルゴリズムを使用する。
ippDitherBayer	Bayer のしきい値ディザリング・アルゴリズムを使用する。

levels	ハーフトーン（ディザリング）の出力レベルの数。次の範囲内の値である。 [2..(1<<*depth*)], depth は、デスティネーション・イメージの色分解能である。

説明

関数 ippiReduceBits は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ *pSrc* の各チャネルの強さのレベル数を減らし、その結果をデスティネーション・イメージ *pDst* の各チャネルに格納する。ただし、ソースのデータ・タイプが浮動小数点型の場合は、RGB 値は [0..1] の範囲内に収まっていなければならない。

levels パラメータは、デスティネーション・イメージの各チャネルの強さのレベル数を設定する。

noise の値が 0 より大きい場合は、計算に使用されるしきい値のレベルに若干のランダム・ノイズが追加される。ノイズ信号の振幅は、*noise* パラメータで指定される。このパラメータは、デスティネーション・イメージの輝度の範囲に対する割合 （%）として設定される。4 × 4 順序付きディザリング・モードの場合、しきい値は使用されるディザ・マトリックスによって決まる。

誤差拡散ディザリング・モードの場合は、入力しきい値は次の *range* の値の 1/2 に設定される。

range = ((1<<*depth*) - 1)/(*levels* – 1)

depth は、ソース・イメージの色分解能である。

浮動小数点データ・タイプの場合は、*range* =1.0/（*levels*-1）になる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsNoiseValErr	*noise* の値が無効な場合のエラー状態を示す。
ippStsDitherLevelsErr	*levels* の値が許容範囲から外れている場合のエラー状態を示す。

Join

422 形式のプレーン順序画像を 2 チャネルのピクセル順序画像に変換する。

```
IppStatus ippiJoin422_8u_P3C2R(const Ipp8u* pSrc[3], int
      srcStep[3],
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	ソース・イメージ・バッファへのポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ値の配列（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiJoin422_8u_P3C2R は、ippi.h ファイルの中で宣言される。この関数は、4:2:2 プレーン形式のソース画像 *pSrc* を、2 チャネルのピクセル順序画像 *pDst* に変換する（4:2:2プレーン形式およびピクセル順序形式の詳細は、表 6-2と表 6-3を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	最初のプレーンの *roiSize.width* が 2 より小さいか、*roiSize.height* がゼロまたはそれより小さい場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsStrideErr	ステップのいずれかが画像の幅より小さい場合のエラー状態を示す。

Split

422 形式の 2 チャネルのピクセル順序画像
をプレーン形式に変換する。

```
IppStatus ippiSplit422_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
     Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・イメージ・バッファへのポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ （バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiSplit422_8u_C2P3R は、ippi.h ファイルの中で宣言される。この関数は、4:2:2 形式の 2 チャネルのピクセル順序ソース画像 *pSrc* を、プレーン形式の画像 *pDst* に変換する （4:2:2 プレーン形式およびピクセル順序形式の詳細は、表 6-2 と表 6-3 を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	最初のプレーンの *roiSize.width* が 2 より小さいか、*roiSize.height* がゼロまたはそれより小さい場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsStrideErr	ステップが画像の幅より小さい場合のエラー状態を示す。

カラー・ツイスト

カラー・ツイスト変換関数は、ソース・ピクセルのすべてのカラー・チャネルの値を使用して、デスティネーションのチャネル値を計算する。デスティネーションのチャネル値は、ソース・ピクセルのチャネル値のベクトルと、それに対応するカラー・ツイスト・マトリックスの行の積を求めることで得られる。

例えば、ソース・ピクセルが (r,g,b) の場合、デスティネーション・ピクセルの値 (R,G,B) は、次のように計算される。

$$R = t_{11}*r + t_{12}*g + t_{13}*b + t_{14}$$
$$G = t_{21}*r + t_{22}*g + t_{23}*b + t_{24}$$
$$B = t_{31}*r + t_{32}*g + t_{33}*b + t_{34}$$

ここで、

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \end{bmatrix}$$

このマトリックスがカラー・ツイスト・マトリックスである。

インテル IPP 関数で使用されるカラー・ツイスト・マトリックスは、浮動小数点要素で構成される 3 × 4 サイズまたは 4 × 4 サイズのマトリックスである。マトリックスの要素は、カラー変換のタイプごとに固有である。

ColorTwist

浮動小数点ピクセル値を含む画像に カラー・ツイスト・マトリックスを 適用する。

事例 1 : ピクセル順序データに対する非インプレース操作

```
IppStatus ippiColorTwist_<mod>(const Ipp32f* pSrc, int srcStep,
      Ipp32f* pDst, int dstStep, IppiSize roiSize,
      const Ipp32f twist[3][4]);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiColorTwist_32f_C4R(const Ipp32f* pSrc, int srcStep,
      Ipp32f* pDst, int dstStep, IppiSize roiSize,
      const Ipp32f twist[4][4]);
```

事例 2：プレーン・データに対する非インプレース操作

```
IppStatus ippiColorTwist_32f_P3R(const Ipp32f* const pSrc[3], int
      srcStep,  Ipp32f* const pDst[3], int dstStep, IppiSize
      roiSize,
      const Ipp32f twist[3][4]);
```

事例 3：ピクセル順序データに対するインプレース操作

```
IppStatus ippiColorTwist_<mod>(Ipp32f* pSrcDst, int srcDstStep,
      IppiSize roiSize, const Ipp32f twist[3][4]);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3IR
32f_AC4IR
```

事例 4：プレーン・データに対するインプレース操作

```
IppStatus ippiColorTwist_32f_IP3R(Ipp32f* const pSrcDst[3], int
      srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
twist	カラー・ツイスト・マトリックスの要素を保持する配列

説明

関数 ippiColorTwist は、ippi.h ファイルの中で宣言される。この関数は、浮動小数点ピクセル値を含むソース・イメージ内の 3 つのカラー・チャネルすべてにカラー・ツイスト・マトリックスを適用し、得られたデータをデスティネーション・イメージに格納する。デスティネーションのチャネル値は、ソース・ピクセルのチャネル値のベクトルと、それに対応するカラー・ツイスト・マトリックスの行の積を求めることで得られる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

ColorTwist32f

整数ピクセル値を含む画像にカラー・ツイスト・マトリックスを適用する。

事例 1：ピクセル順序データに対する非インプレース操作

```
IppStatus ippiColorTwist32f_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,  Ipp<datatype>* pDst, int dstStep, IppiSize
      roiSize,
      const Ipp32f twist[3][4]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

事例 2：プレーン・データに対する非インプレース操作

```
IppStatus ippiColorTwist32f_<mod>(const Ipp<datatype>* const
     pSrc[3],
     int srcStep,  Ipp<datatype>* const pDst[3], int dstStep,
     IppiSize
     roiSize, const Ipp32f twist[3][4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_P3R      16u_P3R     16s_P3R
```

事例 3：ピクセル順序データに対するインプレース操作

```
IppStatus ippiColorTwist32f_<mod>(Ipp<datatype>* pSrcDst, int
     srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3IR     16u_C3IR    16s_C3IR
8u_AC4IR    16u_AC4IR   16s_AC4IR
```

事例 4：プレーン・データに対するインプレース操作

```
IppStatus ippiColorTwist32f_<mod>(Ipp<datatype>* const
     pSrcDst[3], int
     srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_IP3R     16u_IP3R    16s_IP3R
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
twist	カラー・ツイスト・マトリックスの要素を保持する配列

説明

関数 ippiColorTwist32f は、ippi.h ファイルの中で宣言される。この関数は、整数ソース・イメージ内の 3 つのカラー・チャネルの値すべてにカラー・ツイスト・マトリックスを適用し、得られたデータをデスティネーション・イメージに格納する。例えば、RGB 画像から YCbCr 形式への変換を、次のように実行できる。

```
Y  =0.299*R   + 0.587*G    + 0.114*B
Cb =- 0.16874*R -  0.33126*G + 0.5*B     + 0.5
Cr =0.5*R     -  0.41869*G  - 0.08131*B + 0.5
```

これらの式は、次のカラー・ツイスト・マトリックスによって記述できる。

```
          0.29900f   0.58700f   0.11400f  0.000f
         -0.16874f  -0.33126f   0.50000f  128.0f
          0.50000f  -0.41869f  -0.08131f  128.0f
```

これ以外にも、カラー・ツイスト・マトリックスを使用して、多くのカラー変換を実行できる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

GammaFwd

RGB データを含むソース・イメージの ガンマ補正を実行する。

事例 1：整数ピクセル順序データに対する非インプレース操作

```
IppStatus ippiGammaFwd_<mod>(const Ipp<datatype>* pSrc, int
      srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
      roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16u_C3R
8u_AC4R     16u_AC4R
```

事例 2：整数プレーン・データに対する非インプレース操作

```
IppStatus ippiGammaFwd_<mod>(const Ipp<datatype>* const
      pSrc[3],
      int srcStep, Ipp<datatype>* const pDst[3], int dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_P3R      16u_P3R
```

事例 3：浮動小数点ピクセル順序データに対する非インプレース操作

```
IppStatus ippiGammaFwd_<mod>(const Ipp32f* pSrc, int srcStep,
      Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f vMin,
      Ipp32f vMax);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

事例 4：浮動小数点プレーン・データに対する非インプレース操作

```
IppStatus ippiGammaFwd_32f_P3R (const Ipp32f* const pSrc[3],
      int srcStep, Ipp32f* const pDst[3], int dstStep, IppiSize
      roiSize, Ipp32f vMin, Ipp32f vMax);
```

事例 5：整数ピクセル順序データに対するインプレース操作

```
IppStatus ippiGammaFwd_<mod>(Ipp<datatype>* pSrcDst, int
      srcDstStep,  IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3IR　16u_C3IR
8u_AC4IR　16u_AC4IR

事例 6：整数プレーン・データに対するインプレース操作

```
IppStatus ippiGammaFwd_<mod>(Ipp<datatype>* const pSrcDst[3],
     int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_IP3R　16u_IP3R

事例 7：浮動小数点ピクセル順序データに対するインプレース操作

```
IppStatus ippiGammaFwd_<mod>(Ipp32f* pSrcDst, int srcDstStep,
     IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

サポートされる *mod* の値は、次のとおりである。

32f_C3IR
32f_AC4IR

事例 8：浮動小数点プレーン・データに対するインプレース操作

```
IppStatus ippiGammaFwd_32f_IP3R (Ipp32f* const pSrcDst[3], int
     srcDstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

引数

pSrc	（ピクセル順序データの場合）ソース・バッファへのポインタ （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ

srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ （バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
vMin, *vMax*	入力される浮動小数点データの最小値と最大値

説明

関数 ippiGammaFwd は、ippi.h ファイルの中で宣言される。この関数は、RGB データを持つソース・イメージのガンマ補正を実行する。次の基本的な式を使用して、RGB 画像をガンマ補正された R'G'B' 画像に変換する。

R, G, B ＜0.018 の場合

$$R' = 4.5 * R$$
$$G' = 4.5 * G$$
$$B' = 4.5 * B$$

R, G, B ≧ 0.018 の場合

$$R' = 1.099 * R^{0.45} - 0.099$$
$$G' = 1.099 * G^{0.45} - 0.099$$
$$B' = 1.099 * B^{0.45} - 0.099$$

ただし、チャネルの輝度の値は、[0..1] の範囲内に収まるように正規化される。[ITU709] 仕様に従って、ガンマ値は 1/0.45 = 2.22 である。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsGammaRangeErr	入力データの上下限が不適当な場合（つまり、*vMax* が *vMin* より小さいかまたは等しい場合）のエラー状態を示す。

GammaInv

ガンマ補正された R'G'B' 画像を元の RGB 画像に変換する。

事例 1：整数ピクセル順序データに対する非インプレース操作

```
IppStatus ippiGammaInv_<mod>(const Ipp<datatype>* pSrc, int
      srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16u_C3R
8u_AC4R     16u_AC4R
```

事例 2：整数プレーン・データに対する非インプレース操作

```
IppStatus ippiGammaInv_<mod>(const Ipp<datatype>* const
      pSrc[3],
      int srcStep, Ipp<datatype>* const pDst[3], int dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_P3R      16u_P3R
```

事例 3：浮動小数点ピクセル順序データに対する非インプレース操作

```
IppStatus ippiGammaInv_<mod>(const Ipp32f* pSrc, int srcStep,
      Ipp32f*
      pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f
      vMax);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

事例 4：浮動小数点プレーン・データに対する非インプレース操作

```
IppStatus ippiGammaInv_32f_P3R (const Ipp32f* const pSrc[3], int
      srcStep, Ipp32f* const pDst[3], int dstStep, IppiSize
      roiSize,
      Ipp32f vMin, Ipp32f vMax);
```

事例 5：整数ピクセル順序データに対するインプレース操作

```
IppStatus ippiGammaInv_<mod>(Ipp<datatype>* pSrcDst, int
     srcDstStep,  IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3IR     16u_C3IR
8u_AC4IR    16u_AC4IR
```

事例 6：整数プレーン・データに対するインプレース操作

```
IppStatus ippiGammaInv_<mod>(Ipp<datatype>* const pSrcDst[3],
     int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_IP3R     16u_IP3R
```

事例 7：浮動小数点ピクセル順序データに対するインプレース操作

```
IppStatus ippiGammaInv_<mod>(Ipp32f* pSrcDst, int srcDstStep,
     IppiSize
     roiSize, Ipp32f vMin, Ipp32f vMax);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3IR
32f_AC4IR
```

事例 8：浮動小数点プレーン・データに対するインプレース操作

```
IppStatus ippiGammaInv_32f_IP3R (Ipp32f* const pSrcDst[3], int
     srcDstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

引数

pSrc	（ピクセル順序データの場合）ソース・バッファへのポインタ （プレーン・データの場合）ソースのカラー・プレーンを区切るポインタの配列
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	（ピクセル順序データの場合）デスティネーション・バッファへのポインタ （プレーン・データの場合）デスティネーションのカラー・プレーンを区切るポインタの配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）

pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
vMin, *vMax*	入力される浮動小数点データの最小値と最大値

説明

関数 ippiGammaInv は、ippi.h ファイルの中で宣言される。この関数は、ガンマ補正された R'G'B' 画像を元の RGB 画像に変換する。以下の式を使用する。

R´,G´,B´ < 0.0812 の場合

```
R = R´/4.5
G = G´/4.5
B = B´/4.5
```

R´,G´,B´ ≧ 0.0812 の場合

$$R = \left(\frac{R' + 0.099}{1.099}\right)^{2.22}$$

$$G = \left(\frac{G' + 0.099}{1.099}\right)^{2.22}$$

$$B = \left(\frac{B' + 0.099}{1.099}\right)^{2.22}$$

ただし、チャネルの輝度の値は、[0..1] の範囲内に収まるように正規化される。[ITU709] 仕様に従って、ガンマ値は 1/0.45 = 2.22 である。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーを示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。

ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsGammaRangeErr	入力データの上下限が不適当な場合 （つまり、*vMax* が *vMin* より小さいかまたは等しい場合）のエラー状態を示す。

7 しきい値演算と比較演算

本章では、ピクセル単位で画像処理するインテル® IPP 画像処理関数（しきい値関数と比較関数）について説明する。

表 7-1 に、このグループの関数の一覧を示す。

表 7-1　画像のしきい値関数と比較関数

関数の基本名	操作
しきい値	
Threshold	指定された比較演算を使用して、画像のピクセル値のしきい値演算を実行する。
Threshold_GT	「より大きい」の比較演算を使用して、画像のピクセル値のしきい値演算を実行する。
Threshold_LT	「より小さい」の比較演算を使用して、画像のピクセル値のしきい値演算を実行する。
Threshold_Val	画像のピクセル値のしきい値演算を実行する。比較条件を満たすピクセルが、指定の値に設定される。
Threshold_GTVal	画像のピクセル値のしきい値演算を実行する。しきい値より大きい値のピクセルが、指定の値に設定される。
Threshold_LTVal	画像のピクセル値のしきい値演算を実行する。しきい値より小さい値のピクセルが、指定の値に設定される。
Threshold_LTValGTVal	画像のピクセル値のダブルしきい値演算を実行する。
比較	
Compare	指定された比較演算を使用して、2 つの画像のピクセル値を比較する。
CompareC	指定された比較演算を使用して、ソース・イメージのピクセル値を指定の値と比較する。
CompareEqualEps	浮動小数点データを持つ 2 つの画像を比較し、両方のピクセル値が等しい（誤差が *eps* 以内）かどうかテストする。
CompareEqualEpsC	画像の浮動小数点型のピクセル値が、指定の値と等しい（誤差が *eps* 以内）かどうかテストする。

しきい値

しきい値関数は、ピクセル値が指定のしきい値より大きいか小さいかを調べて、ピクセル値を変更する。

ピクセルのしきい値演算で使用する比較演算のタイプは、*ippCmpOp* パラメータで指定する。指定できるのは、「より大きい」か「より小さい」のいずれかである（詳細については、第 2 章の「構造体と列挙子」を参照）。しきい値関数の一部は、比較演算のタイプが固定のものもある。

入力ピクセル値が比較条件を満たすと、それに対する出力ピクセル値は、各しきい値関数に固有の固定値に設定される。比較条件が満たされなければ、入力ピクセル値がそのまま出力されるか、別の固定値に設定される（詳細は、各関数の説明を参照のこと）。

マルチチャネル・データの画像の場合は、ピクセルのすべてのカラー・チャネルの値が比較条件を満たすときのみ、そのピクセルの比較の結果が真になる。

Threshold

イメージ・バッファ内のピクセル値に対してしきい値演算を実行する。

事例 1：1 チャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
      Ipp<datatype> threshold, IppCmpOp ippCmpOp);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R     32f_C1R
```

事例 2：マルチチャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
      const Ipp<datatype> threshold[3], IppCmpOp ippCmpOp);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R     32f_C3R
8u_AC4R     16s_AC4R    32f_AC4R
```

事例 3：1 チャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_<mod>(Ipp<datatype>* pSrcDst, int
srcDstStep,
      IppiSize roiSize, Ipp<datatype> threshold, IppCmpOp
ippCmpOp);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IR     16s_C1IR    32f_C1IR
```

事例 4：マルチチャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_<mod>(Ipp<datatype>* pSrcDst, int
srcDstStep,
      IppiSize roiSize, const Ipp<datatype> threshold[3], IppCmpOp
      ippCmpOp);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3IR     16s_C3IR    32f_C3IR
8u_AC4IR    16s_AC4IR   32f_AC4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
threshold	各ピクセルに対するしきい値。マルチチャネル・データの場合は、各カラー・チャネルに対するしきい値配列を使用する。
ippCmpOp	ピクセル値と *threshold* 値を比較する演算。「より小さい」か「より大きい」のいずれかを使用できる。

説明

関数 ippiThreshold は、ippi.h ファイルの中で宣言される。この関数は、*threshold* に指定されたしきい値を使用して、ソース・イメージ *pSrc* 内のピクセルのしきい値演算を実行する。ソース・イメージ内のピクセル値は、*ippCmpOp* 比較演算を使用して、*threshold* 値と比較される。

比較の結果が真ならば、それに対応する出力ピクセルが *threshold* 値に設定される。偽ならば、ソース・ピクセル値がそのまま出力ピクセル値となる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

Threshold_GT

「より大きい」の比較を使用して、イメージ・バッファ内のピクセル値に対するしきい値演算を実行する。

事例 1：1 チャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_GT_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize,
      Ipp<datatype> threshold);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1R      16s_C1R      32f_C1R
```

事例 2：マルチチャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_GT_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize,
      const Ipp<datatype> threshold[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R     32f_C3R
8u_AC4R     16s_AC4R    32f_AC4R
```

事例 3：1 チャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_GT_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IR     16s_C1IR    32f_C1IR
```

事例 4：マルチチャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_GT_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, const Ipp<datatype>
      threshold[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3IR     16s_C3IR    32f_C3IR
8u_AC4IR    16s_AC4IR   32f_AC4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

threshold	各ピクセルに対するしきい値。マルチチャネル・データの場合は、各カラー・チャネルに対するしきい値配列を使用する。

説明

関数 ippiThreshold_GT は、ippi.h ファイルの中で宣言される。この関数は、*threshold* に指定されたしきい値を使用して、ソース・イメージ *pSrc* 内のピクセルのしきい値演算を実行する。ソース・イメージ内のピクセル値は、「より大きい」の比較演算を使用して、*threshold* 値と比較される。

比較の結果が真ならば、それに対応する出力ピクセルが *threshold* 値に設定される。偽ならば、ソース・ピクセル値がそのまま出力ピクセル値となる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

Threshold_LT

「より小さい」の比較を使用して、イメージ・バッファ内のピクセル値に対するしきい値演算を実行する。

事例 1：1 チャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_LT_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize,
      Ipp<datatype> threshold);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1R      16s_C1R      32f_C1R
```

事例 2：マルチチャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_LT_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize,
      const Ipp<datatype> threshold[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R     32f_C3R
8u_AC4R     16s_AC4R    32f_AC4R
```

事例 3：1 チャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_LT_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IR     16s_C1IR    32f_C1IR
```

事例 4：マルチチャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_LT_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, const Ipp<datatype>
      threshold[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3IR     16s_C3IR    32f_C3IR
8u_AC4IR    16s_AC4IR   32f_AC4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

threshold	各ピクセルに対するしきい値。マルチチャネル・データの場合は、各カラー・チャネルに対するしきい値配列を使用する。

説明

関数 ippiThreshold_LT は、ippi.h ファイルの中で宣言される。この関数は、*threshold* に指定されたしきい値を使用して、ソース・イメージ *pSrc* 内のピクセルのしきい値演算を実行する。ソース・イメージ内のピクセル値は、「より小さい」の比較演算を使用して、*threshold* 値と比較される。比較の結果が真ならば、それに対応する出力ピクセルが *threshold* 値に設定される。
偽ならば、ソース・ピクセル値がそのまま出力ピクセル値となる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

Threshold_Val

イメージ・バッファ内のピクセル値に対してしきい値演算を実行する。比較条件を満たすピクセルを、指定の値に設定する。

事例 1：1 チャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize,
      Ipp<datatype> threshold, Ipp<datatype> value, IppCmpOp
      ippCmpOp);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1R      16s_C1R      32f_C1R
```

事例 2：マルチチャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize,
      const Ipp<datatype> threshold[3], const Ipp<datatype>
value[3],
      IppCmpOp ippCmpOp);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R     32f_C3R
8u_AC4R     16s_AC4R    32f_AC4R
```

事例 3：1 チャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold,
      Ipp<datatype> value, IppCmpOp ippCmpOp);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IR     16s_C1IR    32f_C1IR
```

事例 4：マルチチャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, const Ipp<datatype>
      threshold[3], const Ipp<datatype> value[3],
      IppCmpOp ippCmpOp);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3IR     16s_C3IR    32f_C3IR
8u_AC4IR    16s_AC4IR   32f_AC4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ

srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
threshold	各ピクセルに対するしきい値。マルチチャネル・データの場合は、3 つのカラー・チャネルに対応する 3 要素のしきい値配列を使用する。
value	比較条件を満たすピクセルに設定する出力値。マルチチャネル・データの場合は、3 つのカラー・チャネルに対応する 3 要素の出力値配列を使用する。
ippCmpOp	ピクセル値と *threshold* 値を比較する演算。「より小さい」か「より大きい」のどちらかを使用できる。

説明

関数 ippiThreshold_Val は、ippi.h ファイルの中で宣言される。この関数は、*threshold* に指定されたしきい値を使用して、ソース・イメージ *pSrc* 内のピクセルのしきい値演算を実行する。ソース・イメージ内のピクセル値は、*ippCmpOp* 比較演算を使用して、*threshold* 値と比較される。比較の結果が真ならば、それに対する出力ピクセルが、*value* に指定された値に設定される。偽ならば、ソース・ピクセル値がそのまま出力ピクセル値となる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

Threshold_GTVal

イメージ・バッファ内のピクセル値に対してしきい値演算を実行する。しきい値より大きいピクセルを、指定の値に設定する。

事例 1：1 チャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize,
      Ipp<datatype> threshold, Ipp<datatype> value);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R　　16s_C1R　　32f_C1R

事例 2：マルチチャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize,
      const Ipp<datatype> threshold[3], const Ipp<datatype>
value[3]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R　　16s_C3R　　32f_C3R
8u_AC4R　　16s_AC4R　　32f_AC4R

```
IppStatus ippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize,
      const Ipp<datatype> threshold[4], const Ipp<datatype>
value[4]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4R　　16s_C4R　　32f_C4R

事例 3：1 チャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold,
      Ipp<datatype> value);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR　　16s_C1IR　　32f_C1IR

事例 4：マルチチャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, const Ipp<datatype>
      threshold[3], const Ipp<datatype> value[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3IR     16s_C3IR    32f_C3IR
8u_AC4IR    16s_AC4IR   32f_AC4IR
```

```
IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, const Ipp<datatype>
      threshold[4], const Ipp<datatype> value[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4IR     16s_C4IR    32f_C4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
threshold	各ピクセルに対するしきい値。マルチチャネル・データの場合は、各チャネルに対応するしきい値配列を使用する。
value	比較条件を満たすピクセルに設定する出力値。マルチチャネル・データの場合は、各チャネルに対応する出力値配列を使用する。

説明

関数 ippiThreshold_GTVal は、ippi.h ファイルの中で宣言される。この関数は、*threshold* に指定されたしきい値を使用して、ソース・イメージ *pSrc* 内のピクセルのしきい値演算を実行する。ソース・イメージ内のピクセル値は、「より大きい」の比較演算を使用して、*threshold* 値と比較される。

比較の結果が真ならば、それに対応する出力ピクセルが、*value* に指定された値に設定される。偽ならば、ソース・ピクセル値がそのまま出力ピクセル値となる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

Threshold_LTVal

イメージ・バッファ内のピクセル値に対してしきい値演算を実行する。しきい値より小さいピクセルを、指定の値に設定する。

事例 1：1 チャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize,
      Ipp<datatype> threshold, Ipp<datatype> value);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1R      16s_C1R      32f_C1R
```

事例 2：マルチチャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize,
      const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R     32f_C3R
8u_AC4R     16s_AC4R    32f_AC4R
```

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize,
      const Ipp<datatype> threshold[4], const Ipp<datatype>
value[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R     32f_C4R
```

事例 3：1 チャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold,
      Ipp<datatype> value);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IR     16s_C1IR    32f_C1IR
```

事例 4：マルチチャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, const Ipp<datatype>
      threshold[3], const Ipp<datatype> value[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3IR     16s_C3IR    32f_C3IR
8u_AC4IR    16s_AC4IR   32f_AC4IR
```

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, const Ipp<datatype>
      threshold[4], const Ipp<datatype> value[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4IR     16s_C4IR    32f_C4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
threshold	各ピクセルに対するしきい値。マルチチャネル・データの場合は、各チャネルに対応するしきい値配列を使用する。
value	比較条件を満たすピクセルに設定する出力値。マルチチャネル・データの場合は、各チャネルに対応する出力値配列を使用する。

説明

関数 ippiThreshold_LTVal は、ippi.h ファイルの中で宣言される。この関数は、*threshold* に指定されたしきい値を使用して、ソース・イメージ *pSrc* 内のピクセルのしきい値演算を実行する。ソース・イメージ内のピクセル値は、「より小さい」の比較演算を使用して、*threshold* 値と比較される。比較の結果が真ならば、それに対する出力ピクセルが、*value* に指定された値に設定される。偽ならば、ソース・ピクセル値がそのまま出力ピクセル値となる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

Threshold_LTValGTVal

イメージ・バッファ内のピクセル値に対し
てダブルしきい値演算を実行する。

事例 1：1 チャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>*
pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
      roiSize, Ipp<datatype> thresholdLT, Ipp<datatype> valueLT,
      Ipp<datatype> thresholdGT, Ipp<datatype> valueGT);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1R      16s_C1R     32f_C1R
```

事例 2：マルチチャネル・データに対する非インプレース操作

```
IppStatus ippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>*
pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize,
      const Ipp<datatype> thresholdLT[3], const Ipp<datatype>
valueLT[3],
      const Ipp<datatype> thresholdGT[3], const Ipp<datatype>
valueGT[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C3R      16s_C3R     32f_C3R
      8u_AC4R     16s_AC4R    32f_AC4R
```

事例 3：1 チャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, Ipp<datatype> thresholdLT,
      Ipp<datatype> valueLT, Ipp<datatype> thresholdGT,
      Ipp<datatype> valueGT);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1IR     16s_C1IR    32f_C1IR
```

事例 4：マルチチャネル・データに対するインプレース操作

```
IppStatus ippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, const Ipp<datatype>
thresholdLT[3],
      const Ipp<datatype> valueLT[3], const Ipp<datatype>
thresholdGT[3],
      const Ipp<datatype> valueGT[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3IR     16s_C3IR    32f_C3IR
8u_AC4IR    16s_AC4IR   32f_AC4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
thresholdLT	各ピクセルに対する下限しきい値。マルチチャネル・データの場合は、3 つのカラー・チャネルに対応する 3 要素の下限しきい値配列を使用する。
valueLT	*thresholdLT* より小さいピクセルに設定する下限出力値。マルチチャネル・データの場合は、3 つのカラー・チャネルに対応する 3 要素の下限出力値配列を使用する。
thresholdGT	各ピクセルに対する上限しきい値。マルチチャネル・データの場合は、3 つのカラー・チャネルに対応する 3 要素の上限しきい値配列を使用する。
valueGT	*thresholdGT* より大きいピクセルに設定する上限出力値。マルチチャネル・データの場合は、3 つのカラー・チャネルに対応する 3 要素の上限出力値配列を使用する。

説明

関数 ippiThreshold_LTValGTVal は、ippi.h ファイルの中で宣言される。この関数は、*thresholdLT* と *thresholdGT* に指定された 2 つのしきい値を使用して、ソース・イメージ *pSrc* 内のピクセルのしきい値演算を実行する。ソース・イメージ内のピクセル値は、これらの 2 つのしきい値と比較される。*thresholdLT* より小さいピクセルの出力ピクセル値は *valueLT* に設定され、*thresholdGT* より大きいピクセルの出力ピクセル値は、*valueGT* に設定される。それ以外のピクセルでは、ソース・ピクセル値がそのまま出力ピクセル値となる。
levelLT の値は、*levelGT* より小さいか等しくなければならない。

次のコード例は、2 つのレベルを使用したしきい値演算を示している。

例 7-1　画像のしきい値演算

```
IppStatus threshold( void ) {
   Ipp8u x[5*4];
   IppiSize roi = {5,4};
   int i;
   for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;
   return ippiThreshold_LTValGTVal_8u_C1IR( x, 5, roi,
2,1,6,7, );
}
```

デスティネーション・イメージ *x* には、次のデータが含まれる。

```
01 01 02 03 04
05 06 07 07 07
07 07 07 07 07
07 07 07 07 07
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsThresholdErr	*levelLT* が *levelGT* より大きい場合のエラーを示す。

ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

比較演算

本節では、画像を比較する関数について説明する。各比較関数は、比較の結果を 1 チャネルの Ipp8u 出力画像に書き込む。入力ピクセルが比較条件を満たす場合は、それに対する出力ピクセルが 0 以外に設定され、満たさない場合は 0 に設定される。「より大きい」、「以上」、「より小さい」、「以下」、または「等しい」のいずれかの比較条件を使用して、2 つの画像、または画像と定数を比較できる。比較条件は、IppCmpOp 型の関数の引数で指定する（詳細については、第 2 章の「構造体と列挙子」」を参照）。
浮動小数点データを持つ画像の場合も、許容誤差 eps を指定することで、等しい（誤差が *eps* 以内である）かどうかテストできる。

マルチチャネル・データの画像の場合は、ピクセルのすべてのカラー・チャネルの値が比較条件を満たすときのみ、そのピクセルの比較の結果が真になる。

Compare

指定された比較演算を使用して、2 つの画像内のピクセル値を比較する。

```
IppStatus ippiCompare_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp
ippCmpOp);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc1, *pSrc2*	ソース・イメージ・バッファへのポインタ
src1Step, *src2Step*	ソース・イメージ・バッファ内のステップ（バイト単位）

pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
ippCmpOp	ピクセル値の比較に使用する比較演算

説明

関数 ippiCompare は、ippi.h ファイルの中で宣言される。この関数は、*ippCmpOp* 比較演算を使用して、2 つの入力画像 *pSrc1* および *pSrc2* を比較し、比較の結果を 1 チャネルの Ipp8u 画像 pDst に書き込む。比較の結果が真ならば、それに対する出力ピクセルが 0 以外に設定され、偽ならば 0 に設定される。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc1*、*pSrc2*、または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*src1Step*、*src2Step*、または *dstStep* の値が 0 または負の場合のエラー状態を示す。

CompareC

指定された比較演算を使用して、ソース・イメージのピクセル値を指定の値と比較する。

事例 1：1 チャネル・データに対する操作

```
IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int
srcStep,
     Ipp<datatype> value, Ipp8u* pDst, int dstStep, IppiSize
roiSize,
     IppCmpOp ippCmpOp);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R     32f_C1R
```

事例 2：マルチチャネル・データに対する操作

```
IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int
srcStep,
      const Ipp<datatype> value[3], Ipp8u* pDst, int dstStep,
      IppiSize roiSize, IppCmpOp ippCmpOp);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R     32f_C3R
8u_AC4R     16s_AC4R    32f_AC4R
```

```
IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int
srcStep,
      const Ipp<datatype> value[4], Ipp8u* pDst, int dstStep,
      IppiSize roiSize, IppCmpOp ippCmpOp);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R     32f_C4R
```

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
value	各ピクセルと比較する値。マルチチャネル・データの場合は、各チャネルに対応する個別の値配列を使用する。
pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
ippCmpOp	ピクセル値の比較に使用する比較演算

説明

関数 ippiCompareC は、ippi.h ファイルの中で宣言される。この関数は、*ippCmpOp* 比較演算を使用して、ソース・イメージ *pSrc* のピクセルを、*value* に指定された値と比較し、比較の結果を 1 チャネルの Ipp8u 画像 *pDst* に書き込む。比較の結果が真ならば、それに対する出力ピクセルが 0 以外に設定され、偽ならば 0 に設定される。

次のコード例は、比較関数を使用してマスク・イメージを作成する方法を示している。

例 7-2　画像と定数値の比較

```
IppStatus compare ( void ) {
   Ipp8u x[5*4], y[5*4];
   IppiSize roi = {5,4};
   int i;
   for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;
   return ippiCompareC_8u_C1R ( x, 5, 7, y, 5, roi,
ippCmpGreater );
}
```

マスク・イメージ *y* には、次のデータが含まれる。

```
00 00 00 00 00
00 00 00 FF FF
FF FF FF FF FF
FF FF FF FF FF
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が0または負の場合のエラー状態を示す。

CompareEqualEps

浮動小数点データを持つ 2 つの画像を比較し、各ピクセル値が等しい（誤差が eps 以内）かどうかテストする。

```
IppStatus ippiCompareEqualEps_<mod>(const Ipp32f* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

引数

pSrc1, pSrc2	ソース・イメージ・バッファへのポインタ
src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
eps	許容誤差

説明

関数 ippiCompareEqualEps は、ippi.h ファイルの中で宣言される。この関数は、2 つの入力画像 *pSrc1* および *pSrc2* の対応するピクセルが等しい（誤差が *eps* 以内）かどうかテストし、結果を 1 チャネルの Ipp8u 画像 *pDst* に書き込む。*pSrc1* と *pSrc2* のピクセル値の差の絶対値が *eps* より小さければ、*pDst* 内の対応するピクセルが 0 以外に設定され、そうでなければ 0 に設定される。
マルチチャネル・イメージの場合は、ピクセルのすべてのカラー・チャネル値の差が許容誤差 *eps* 以内のときのみ、そのピクセルの比較の結果が真になる。

この関数は、浮動小数点データを持つ画像処理にのみ使用できる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc1*、*pSrc2*、または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*src1Step*、*src2Step*、または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsEpsValErr	*eps* の値が負の場合のエラー状態を示す。

CompareEqualEpsC

画像の浮動小数点型のピクセル値が指定の値と等しい（誤差が eps 以内）かどうかテストする。

事例 1：1 チャネル・データに対する操作

```
IppStatus ippiCompareEqualEpsC_32fC1R(const Ipp32f* pSrc,
      int srcStep, Ipp32f value, Ipp8u* pDst, int dstStep,
      IppiSize roiSize, Ipp32f eps);
```

事例 2：マルチチャネル・データに対する操作

```
IppStatus ippiCompareEqualEpsC_<mod>(const Ipp32f* pSrc, int
srcStep,
      const Ipp32f value[3], Ippu* pDst, int dstStep, IppiSize
roiSize,
      Ipp32f eps);
```

サポートされる *mod* の値は、次のとおりである。

```
      32f_C3R
      32f_AC4R
```

```
IppStatus ippiCompareEqualEpsC_32f_C4R(const Ipp32f* pSrc, int
srcStep,
     const Ipp32f value[4], Ipp8u* pDst, int dstStep, IppiSize
roiSize,
      Ipp32f eps);
```

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
value	各ピクセルと比較する値。マルチチャネル・データの場合は、各チャネルに対応する個別の値配列を使用する。
pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
eps	許容誤差

説明

関数 ippiCompareEqualEpsC は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ *pSrc* のピクセル値が、*value* に指定された定数値と等しい（誤差が *eps* 以内である）かどうかテストし、その結果を 1 チャネルの Ipp8u 画像 *pDst* に書き込む。pSrc 内のピクセル値と *value* の差の絶対値が *eps* より小さければ、*pDst* 内の対応するピクセルが 0 以外に設定され、そうでなければ 0 に設定される。マルチチャネル・イメージの場合は、ピクセルの各カラー・チャネル値と *value* の対応する要素との差がすべて許容誤差 *eps* 以内のときのみ、そのピクセルの比較の結果が真になる。

この関数は、浮動小数点データを持つ画像処理にのみ使用できる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsEpsValErr	*eps* の値が負の場合のエラー状態を示す。

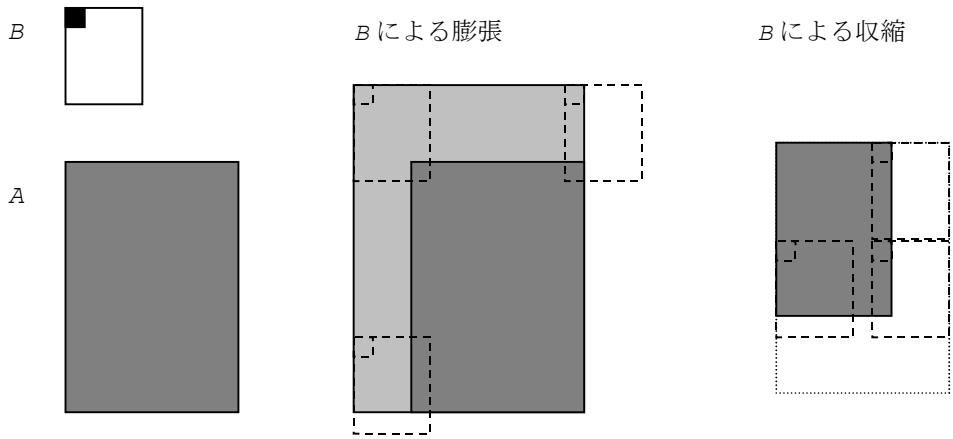
8 モルフォロジー演算

本章では、インテル® インテグレーテッド・パフォーマンス・プリミティブ（インテル® IPP）の画像処理関数のうち、画像に対して収縮 / 膨張のモルフォロジー演算を実行する関数について説明する。

一般的に、収縮と膨張では、オブジェクトの面積を大きく変えずに、境界をスムージングする。収縮と膨張はどちらも、3 × 3 の対称マスク、ユーザ定義の矩形マスク、または構造要素のいずれかを使用する。

一般的に、モルフォロジー演算には、*対象オブジェクト*と呼ばれる画像 *A* と、*構造要素*と呼ばれるカーネル要素 *B* が存在する。画像と構造要素の次元は任意であるが、最もよく使用されるのは 2D バイナリ・イメージと 3D グレー・スケール・イメージである。構造要素 *B* は、ほとんど四角形または円だが、任意の形状を使用できる。たたみ込みの場合と同様に、*B* はアンカ・ポイントを持つカーネルまたはテンプレートである。*B* によるオブジェクト *A* の膨張と収縮を、図 8-1 に示す。この図では、*B* は、左上隅に濃い四角で示したアンカ・ポイントを持つ矩形である。

図 8-1　B による A の膨張と収縮



B の 2D 転置または反転をとし、$\bar{B}$, and B_t を画像周りの *B* の移動とすると、構造要素 *B* によるオブジェクト *A* の膨張は、次のように表される。

$$A \oplus B = \{t : \bar{B}_t \cap A \neq 0\}.$$

これは、共通部分が NULL でない集合内のすべてのピクセルを意味する。すなわち、*B* のピクセルが 1 つでも *A* の内部にあれば、*B* のアンカ・ポイントの下のピクセルが「オン」とマークされる。

$A \oplus nB$ は、膨張を *n* 回実行することを意味する。

構造要素 *B* によるオブジェクト *A* の収縮は、次のように表される。

$A \ominus B = \{t : B_t \subseteq A\}$

すなわち、*B* が完全に *A* の内部にあるときに、*B* のアンカ・ポイントの下のピクセルが「オン」とマークされる。

$A \ominus nB$ は、収縮を *n* 回実行することを意味し、∂A を検出するのに役立つ。*A* の境界は、次のようになる。

$\partial A = A - (A \ominus nB)$

オープンは、次のように表される。

$A \circ B = (A \ominus nB) \oplus nB$

B による *A* のクローズは、次のように表される。

$A \bullet B = (A \oplus nB) \ominus nB,$

ここで、$n > 0$ とする。

グレー・スケールのためのフラットな構造要素

収縮と膨張は、3D 空間、すなわちグレー・レベルで実行ができる。3D 構造要素を使用できるが、最も簡単で最良の方法は、フラットな構造要素 *B* を使用する方法である。図 8-2 は、フラットな構造要素 *B* によるグレー・スケール・イメージ *A* の膨張と収縮の 1D 横断面を示している。この図では、中心よりやや右寄りにある *B* のアンカ・ポイントを、濃いマークで *B* の上に示す。

図 8-2　B による A の膨張と収縮の 1D 横断面

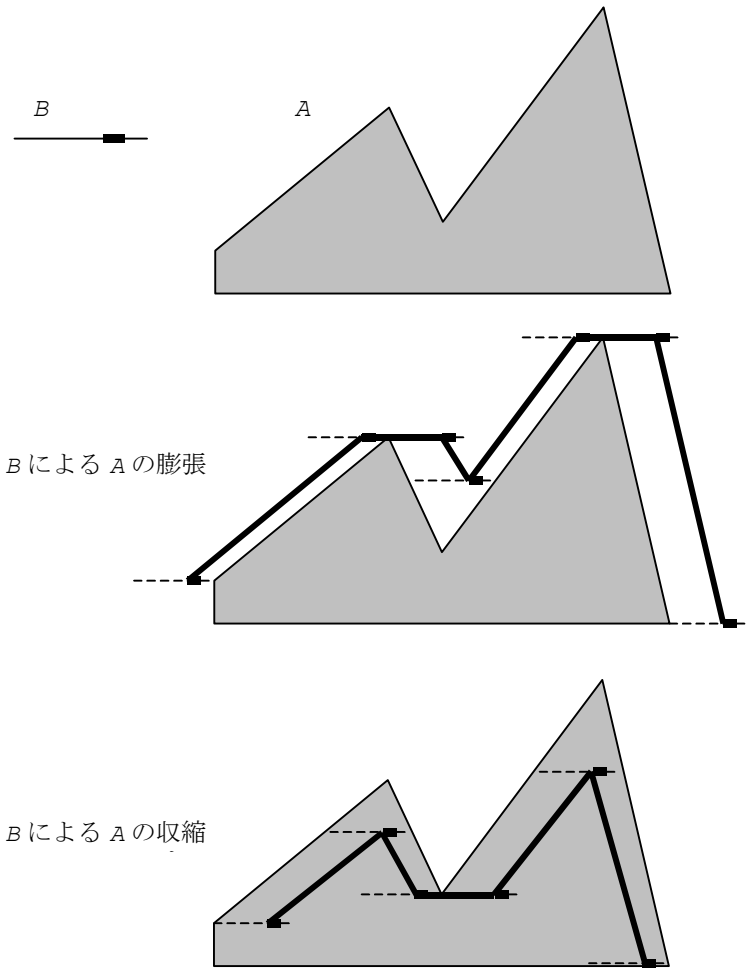


図 8-2 で、膨張の数学的表現は $\sup_{y \in B_t} A$、収縮の数学的表現は $\inf_{y \in B_t} A$ となる。

表 8-1 に、本章で詳しく説明する関数の一覧を示す。

表 8-1　モルフォロジー関数

関数の基本名	操作
Dilate3x3	3 × 3 のマスクを使用して、画像の膨張を実行する。
Erode3x3	3 × 3 のマスクを使用して、画像の収縮を実行する。
Dilate	汎用矩形マスクを使用して、画像の膨張を実行する。
Erode	汎用矩形マスクを使用して、画像の収縮を実行する。
MorphologyInitAlloc	収縮または膨張操作のための内部構造を割り当てて初期化する。
MorphologyFree	収縮または膨張操作で使用した内部構造を解放する。
ErodeStrip	任意の構造要素を使用して、画像の収縮を実行する。
ErodeStrip_Rect	矩形の構造要素を使用して、画像の収縮を実行する。
ErodeStrip_Cross	クロスシェイプの構造要素を使用して、画像の収縮を実行する。
DilateStrip	任意の構造要素を使用して、画像の膨張を実行する。
DilateStrip_Rect	矩形の構造要素を使用して、画像の膨張を実行する。
DilateStrip_Cross	クロスシェイプの構造要素を使用して、画像の膨張を実行する。

Dilate3x3

3 × 3 のマスクを使用して、画像の膨張を実行する。

事例 1：非インプレース操作

```
IppStatus ippiDilate3x3_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      32f_C1R
8u_C3R      32f_C3R
8u_C4R      32f_C4R
8u_AC4R     32f_AC4R
```

事例 2：インプレース操作

```
IppStatus ippiDilate3x3_<mod>(Ipp<datatype>* pSrcDst, int
srcDstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IR     32f_C1IR
8u_C3IR     32f_C3IR
8u_C4IR     32f_C4IR
8u_AC4IR    32f_AC4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiDilate3x3 は、ippi.h ファイルの中で宣言される。この関数は、1、3、または 4 チャネルの 2D 画像内にある矩形 ROI 領域の膨張を実行する。
ソース・イメージとデスティネーション・イメージのサイズは異なっていてもかまわないが、両方の ROI サイズは同じでなければならない。出力ピクセルは、入力ピクセルおよびその 8 つの隣接ピクセルの最大値に設定される。

この膨張関数を使用したコード例を次に示す。

例 8-1　ドットの膨張

```
IppStatus dilate( void ) {
   Ipp8u x[7*5];
   IppiSize roi = {7,5};
   ippiSet_8u_C1R( 0, x, 7, roi );
   x[2*7+3] = 1;
   roi.width = roi.width - 2;
   roi.height = roi.height - 2;
   return ippiDilate3x3_8u_C1IR( x+7+1, 7, roi );
}
```

デスティネーション・イメージ *x* には、次のデータが含まれる。

```
00 00 00 00 00 00 00
00 00 01 01 01 00 00
00 00 01 01 01 00 00
00 00 01 01 01 00 00
00 00 00 00 00 00 00
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

Erode3x3

3 × 3 のマスクを使用して、画像の収縮を
実行する。

事例 1：非インプレース操作

```
IppStatus ippiErode3x3_<mod>(const Ipp<datatype>* pSrc, int
srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      32f_C1R
8u_C3R      32f_C3R
8u_C4R      32f_C4R
8u_AC4R     32f_AC4R
```

事例 2：インプレース操作

```
IppStatus ippiErode3x3_<mod>(Ipp<datatype>* pSrcDst, int
srcDstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IR     32f_C1IR
8u_C3IR     32f_C3IR
8u_C4IR     32f_C4IR
8u_AC4IR    32f_AC4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）

roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiErode3x3 は、ippi.h ファイルの中で宣言される。この関数は、1、3、または 4 チャネルの 2D 画像内にある矩形 ROI 領域の収縮を実行する。
ソース・イメージとデスティネーション・イメージのサイズは異なっていてもかまわないが、両方の ROI サイズは同じでなければならない。出力ピクセルは、入力ピクセルおよびその 8 つの隣接ピクセルの最小値に設定される。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。

Dilate

指定されたマスクを使用して、画像の膨張を実行する。

事例 1：非インプレース操作

```
IppStatus ippiDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const
      char* pMask, IppiSize maskSize, IppiPoint anchor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      32f_C1R
8u_C3R      32f_C3R
8u_C4R      32f_C4R
8u_AC4R     32f_AC4R
```

事例 2：インプレース操作

```
IppStatus ippiDilate_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
      IppiSize roiSize, const char* pMask, IppiSize maskSize,
      IppiPoint anchor));
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IR     32f_C1IR
8u_C3IR     32f_C3IR
8u_AC4IR    32f_AC4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
pMask	マスク値へのポインタ。対応するマスク値が 0 でないピクセルだけが考慮される。
maskSize	マスクのサイズ（ピクセル単位）
anchor	入力ピクセルに対するマスク・アライメントを指定するアンカ・セル

説明

関数 ippiDilate は、ippi.h ファイルの中で宣言される。この関数は、指定されたマスク *Mask*（サイズは *maskSize*、アライメントは *anchor*）を使用して、1、3、または 4 チャネルの 2D 画像内にある矩形 ROI 領域の膨張を実行する。
ソース・イメージとデスティネーション・イメージのサイズは異なっていてもかまわないが、両方の ROI サイズは同じでなければならない。出力ピクセルは、入力ピクセルおよびその 8 つの隣接ピクセルのうち、マスク値が 0 でないものの最大値に設定される。4 チャネル・イメージの場合、アルファ・チャネルは処理されない。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* または *maskSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsZeroMaskValuesErr	マスク値がすべて 0 である場合のエラー状態を表す。

Erode

指定されたマスクを使用して、画像の収縮を実行する。

事例 1：非インプレース操作

```
IppStatus ippiErode_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const
      char* pMask, IppiSize maskSize, IppiPoint anchor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      32f_C1R
8u_C3R      32f_C3R
8u_C4R      32f_C4R
8u_AC4R     32f_AC4R
```

事例 2：インプレース操作

```
IppStatus ippiErode_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
      IppiSize roiSize, const char* pMask, IppiSize maskSize,
      IppiPoint anchor));
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IR     32f_C1IR
8u_C3IR     32f_C3IR
8u_AC4IR    32f_AC4IR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
pMask	マスク値へのポインタ。対応するマスク値が 0 でないピクセルだけが考慮される。
maskSize	マスクのサイズ（ピクセル単位）
anchor	入力ピクセルに対するマスク・アライメントを指定するアンカ・セル

説明

関数 ippiErode は、ippi.h ファイルの中で宣言される。この関数は、指定されたマスク *pMask*（サイズは *maskSize*、アライメントは *anchor*）を使用して、1、3、または 4 チャネルの 2D 画像内にある矩形 ROI 領域の収縮を実行する。
ソース・イメージとデスティネーション・イメージのサイズは異なっていてもかまわないが、両方の ROI サイズは同じ（*dstRoiSize*）でなければならない。出力ピクセルは、入力ピクセルおよびその 8 つの隣接ピクセルのうち、マスク値が 0 でないものの最小値に設定される。4 チャネル・イメージの場合、アルファ・チャネルは処理されない。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* または *maskSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsZeroMaskValuesErr	マスク値がすべて 0 である場合のエラー状態を表す。

MorphologyInitAlloc

収縮または膨張操作のための内部構造体を
割り当てて初期化する。

```
IppStatus ippiMorphologyInitAlloc(int roiWidth, IppDataType
dataType,
    int channels, IppiSize elSize, IppiPoint elAnchor, IppShape
    elShape, const int* pElData, IppiMorphState** ppMorphState);
```

引数

roiWidth	roiWidth 画像 ROI の最大幅（ピクセル単位）。割り当てられた内部構造体を使用して、この範囲の画像を処理できる。
dataType	画像のデータ・タイプ
channels	画像のチャネル数
elSize	構造要素のサイズ
elAnchor	構造要素内のアンカ・ポイント座標。アンカ・ポイントは構造要素内になければならないので、次の条件を満たす必要がある。 $0 \le elAnchor.x < elSize.width$, $0 \le elAnchor.y < elSize.height$.
elShape	構造要素の形状を、次のいずれかの値で指定する。

	ippiShapeRect	矩形の構造要素
	ippiShapeCross	クロスシェイプの構造要素
	ippiShapeEllipse	楕円の構造要素
	ippiShapeCustom	ユーザ定義の構造要素。この場合は、*pElData* にマスクを指定し、考慮に入れる隣接ピクセルを指定する。
pElData	構造要素配列へのポインタ。これは、*elSize.height* 行、*elSize.width* 列のマトリックスである。値が 0 でないポイントだけが、構造要素として意味を持つ。このパラメータは、*elShape* が ippiShapeCustom のときのみ有効である。	
ppMorphState	内部状態構造体へのポインタのポインタ。ここには、モルフォロジー演算で必要とされる、構造要素とテンポラリ・バッファの内部表現が入っている。	

説明

関数 ippiMorphologyInitAlloc は、ippi.h ファイルの中で宣言される。この関数は、割り当てられ、初期化された内部状態構造体への *ppMorphState* ポインタを返す。収縮と膨張を実行する関数は、この内部構造体を使用する。

elShare で指定する構造要素の種類によって、実行できる収縮または膨張の関数が次のように決定される。

ippiShapeRect	ippiDilateStrip_Rect と ippiErodeStrip_Rect,
ippiShapeCross	ippiDilateStrip_Cross と ippiErodeStrip_Cross,
ippiShapeEllipse	ippiDilateStrip と ippiErodeStrip,
ippiShapeCustom	ippiDilateStrip と ippiErodeStrip.

例えば、*elShape* を ippiShapeCross、*dataType* を Ipp8u、*channels* を 3 に引数を設定して、ippiMorphologyInitAlloc を呼び出すと、割り当てられた *ppMorphState* 内部構造体を使用して呼び出せるのは、ippiDilateStrip_Cross_8uC3R だけになる。操作を実行する関数 （この例では、ippiDilateStrip_Cross_8uC3R）は、内部状態構造体が、実行する操作の種類に合っているかどうかチェックする。

収縮 / 膨張操作を実行する関数に渡される画像の ROI 幅は、初期化関数に渡される *roiWidth* を超えてはならないことに注意する。この値も、操作を実行する関数が制御する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*ppMorphState* が NULL、または *elShape* が ippiShapeCustom なのに *pElData* ポインタが NULL であるエラー状態を示す。
ippStsSizeErr	*roiWidth*、*elSize.width*、または *elSize.height* が正の値でないエラー状態を示す。
ippStsDataTypeErr	*dataType* が Ipp8u または Ipp32f でないエラーを示す。
ippStsNumChannelsErr	*channels* が 1、3、または 4 でないエラーを示す。
ippStsOufOfRangeErr	構造要素のアンカが要素の外部にあるか、構造要素の形状が無効であるエラーを示す。
ippStsMemAllocErr	メモリ割り当てに失敗したエラー状態を示す。

MorphologyFree

内部状態構造体を解放し、
MorphologyInitAlloc で割り当てたメモリを
すべて解放する。

```
IppStatus ippiMorphologyFree(IppiMorphState** ppMorphState);
```

引数

ppMorphState	モルフォロジー状態構造体へのポインタのポインタ

説明

関数 ippiMorphologyFree は ippcv.h ファイルの中で宣言される。この関数は、*ppMorphState* 構造体を解放し、関数 ippiMorphologyInitAlloc で収縮 / 膨張操作のために割り当てたメモリをすべて解放する。

ポインタ *ppMorphState* が NULL の場合、この関数は何も実行しない。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*ppMorphState* が NULL であるエラー状態を示す。

ErodeStrip

任意の形状の構造要素を使用して、画像を収縮する。

```
IppStatus ippiErodeStrip_<mod>(const Ipp<DataType>* pSrc, int srcStep,
      Ipp<DataType>* pDst, int dstStep, IppiSize* pRoiSize,
      IppiMorphState* pState, int stage);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R     32f_C1R
8u_C3R     32f_C3R
8u_C4R     32f_C4R
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pRoiSize	ピクセル数で示された入出力の画像 ROI サイズを返す変数へのポインタ。入力パラメータとしては、ソース・イメージの処理部分のサイズを指定する。出力パラメータとしては、出力部分のサイズが返される。*stage* パラメータの説明も参照のこと。
pState	関数 ippiMorphologyInitAlloc に適切な *dataType* と *channels*、および十分な ROI 幅を指定して初期化したモルフォロジー状態構造体へのポインタ。*elShape* 値には制限はない。

stage	次のいずれかで示される処理段階	
	IPPI_WHOLE	関数は画像全体を処理する。
	IPPI_START	関数はサイクリック・バッファを初期化し、大きい画像の上部を処理する。出力部分は、ソース部分より小さいか、構造要素の高さが 1 の場合は等しくなる。
	IPPI_END	IPPI_END 関数はサイクリック・バッファをフラッシュし、大きい画像の下部を処理する。出力部分は、ソース部分より大きいか、構造要素の高さが 1 の場合は等しくなる。
	IPPI_MIDDLE	関数は大きい画像の中央部分を処理する。出力部分は、ソース部分と同じサイズになる。

説明

関数 ippiErodeStrip は ippcv.h ファイルの中で宣言される。この関数は、任意の形状の構造要素を使用して、ソース・イメージ ROI を収縮（すなわち、最小値フィルタを適用）する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてのエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize.width* が正でないか、または *roiSize.height* が負であるエラーを示す。
ippStsStepErr	*srcStep* または *dstStep* が *roiSize.width* * *<pixelSize>* より小さいエラーを示す。
ippStsNotEvenStepErr	浮動小数点画像のステップが 4 で割り切れない場合のエラーを示す。
ippStsOutOfRangeErr	*stage* が IPPI_START、IPPI_MIDDLE、IPPI_END、または IPPI_WHOLE でないエラーを示す。

ippStsContextMatchErr	呼び出された関数のパラメータと、ippiMorphologyInitAlloc に渡されたパラメータが一致しないエラーを示す。

ErodeStrip_Rect

矩形の構造要素を使用して、画像を収縮する。

```
IppStatus ippiErodeStrip_Rect_<mod>(const Ipp<DataType>* pSrc,
      int srcStep, Ipp<DataType>* pDst, int dstStep,
      IppiSize* pRoiSize, IppiMorphState* pState, int stage);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・バッファ内のステップ（バイト単位）
pRoiSize	ピクセル数で示された入出力の画像 ROI サイズを返す変数へのポインタ。入力パラメータとしては、ソース・イメージの処理部分のサイズを指定する。出力パラメータとしては、出力部分のサイズが返される。*stage* パラメータの説明も参照のこと。
pState	関数 ippiMorphologyInitAlloc で、*elShape* に IPPI_SHAPE_RECT を指定し、適切な値の *dataType* と *channels*、および十分な ROI 幅を指定して初期化したモルフォロジー状態構造体へのポインタ
stage	次のいずれかで示される処理段階
	IPPI_WHOLE　関数は画像全体を処理する。

IPPI_START	関数はサイクリック・バッファを初期化し、大きい画像の上部を処理する。出力部分は、ソース部分より小さいか、構造要素の高さが 1 の場合は等しくなる。
IPPI_END	IPPI_END 関数はサイクリック・バッファをフラッシュし、大きい画像の下部を処理する。出力部分は、ソース部分より大きいか、構造要素の高さが 1 の場合は等しくなる。
IPPI_MIDDLE	関数は大きい画像の中央部分を処理する。出力部分は、ソース部分と同じサイズになる。

説明

関数 ippiErodeStrip_Rect は、ippcv.h ファイルの中で宣言される。この関数は、矩形の構造要素を使用して、ソース・イメージ ROI を収縮（すなわち、最小値フィルタを適用）する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてのエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize.width* が正でないか、または *roiSize.height* が負であるエラーを示す。
ippStsStepErr	*srcStep* または *dstStep* が *oiSize.width* * *<pixelSize>* より小さいエラーを示す。
ippStsNotEvenStepErr	浮動小数点画像のステップが 4 の倍数ではないエラーを示す。
ippStsOutOfRangeErr	*stage* が IPPI_START、IPPI_MIDDLE、IPPI_END、または IPPI_WHOLE でないエラーを示す。
ippStsContextMatchErr	呼び出された関数のパラメータと、ippiMorphologyInitAlloc に渡されたパラメータが一致しないエラーを示す。

ErodeStrip_Cross

クロスシェイプの構造要素を使用して、画像を収縮する。

```
IppStatus ippiErodeStrip_Cross_<mod>(const Ipp<DataType>* pSrc,
      int srcStep, Ipp<DataType>* pDst, int dstStep,
      IppiSize* pRoiSize, IppiMorphState* pState, int stage);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pRoiSize	ピクセル数で示された入出力の画像 ROI サイズを返す変数へのポインタ。入力パラメータとしては、ソース・イメージの処理部分のサイズを指定する。出力パラメータとしては、出力部分のサイズが返される。*stage* パラメータの説明も参照のこと。
pState	関数 ippiMorphologyInitAlloc で、*elShape* に IPPI_SHAPE_CROSS を指定し、適切な値の *dataType* と *channels*、および十分な ROI 幅を指定して初期化したモルフォロジー状態構造体へのポインタ
stage	次のいずれかで示される処理段階
IPPI_WHOLE	関数は画像全体を処理する。
IPPI_START	関数はサイクリック・バッファを初期化し、大きい画像の上部を処理する。出力部分は、ソース部分より小さいか、構造要素の高さが 1 の場合は等しくなる。

IPPI_END	関数はサイクリック・バッファをフラッシュし、大きい画像の下部を処理する。出力部分は、ソース部分より大きいか、構造要素の高さが 1 の場合は等しくなる。
IPPI_MIDDLE	関数は大きい画像の中央部分を処理する。出力部分は、ソース部分と同じサイズになる。

説明

関数 ippiErodeStrip_Cross は、ippcv.h ファイルの中で宣言される。この関数は、クロスシェイプの構造要素を使用して、ソース・イメージ ROI を収縮（すなわち、最小値フィルタを適用）する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize.width* が正でないか、または *roiSize.height* が負であるエラーを示す。
ippStsStepErr	*srcStep* または *dstStep* が *roiSize.width* * *<pixelSize>* より小さいエラーを示す。
ippStsNotEvenStepErr	浮動小数点画像のステップが 4 で割り切れない場合のエラーを示す。
ippStsOutOfRangeErr	*stage* が IPPI_START、IPPI_MIDDLE、IPPI_END、または IPPI_WHOLE でないエラーを示す。
ippStsContextMatchErr	呼び出された関数のパラメータと、ippiMorphologyInitAlloc に渡されたパラメータが一致しないエラーを示す。

DilateStrip

任意の形状の構造要素を使用して、画像を膨張する。

```
IppStatus ippiDilateStrip_<mod>(const Ipp<DataType>* pSrc,
      int srcStep, Ipp<DataType>* pDst, int dstStep,
      IppiSize* pRoiSize, IppiMorphState* pState, int stage);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pRoiSize	ピクセル数で示された入出力の画像 ROI サイズを返す変数へのポインタ。入力パラメータとしては、ソース・イメージの処理部分のサイズを指定する。出力パラメータとしては、出力部分のサイズが返される。*stage* パラメータの説明も参照のこと。
pState	関数 ippiMorphologyInitAlloc に適切な *dataType* と *channels*、および十分な ROI 幅を指定して初期化したモルフォロジー状態構造体へのポインタ。*elShape* 値には制限はない。
stage	次のいずれかで示される処理段階
IPPI_WHOLE	関数は画像全体を処理する。
IPPI_START	関数はサイクリック・バッファを初期化し、大きい画像の上部を処理する。出力部分は、ソース部分より小さいか、構造要素の高さが 1 の場合は等しくなる。

IPPI_END	関数はサイクリック・バッファをフラッシュし、大きい画像の下部を処理する。出力部分は、ソース部分より大きいか、構造要素の高さが 1 の場合は等しくなる。
IPPI_MIDDLE	関数は大きい画像の中央部分を処理する。出力部分は、ソース部分と同じサイズになる。

説明

関数 ippiDilateStrip は、ippcv.h ファイルの中で宣言される。この関数は、任意の形状の構造要素を使用して、ソース・イメージの ROI を膨張（すなわち、最大値フィルタを適用）する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize.width* が正でないか、または *roiSize.height* が負であるエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* が *roiSize.width * <pixelSize>* より小さいエラーを示す。
ippStsNotEvenStepErr	浮動小数点画像のステップが 4 で割り切れない場合のエラーを示す。
ippStsOutOfRangeErr	*stage* が IPPI_START、IPPI_MIDDLE、IPPI_END、または IPPI_WHOLE でないエラーを示す。
ippStsContextMatchErr	呼び出された関数のパラメータと、ippiMorphologyInitAlloc に渡されたパラメータが一致しないエラーを示す。

DilateStrip_Rect

矩形の構造要素を使用して、画像を膨張する。

```
IppStatus ippiDilateStrip_Rect_<mod>(const Ipp<DataType>* pSrc,
      int srcStep, Ipp<DataType>* pDst, int dstStep,
      IppiSize* pRoiSize, IppiMorphState* pState, int stage);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pRoiSize	ピクセル数で示された入出力の画像 ROI サイズを返す変数のポインタ。入力パラメータとしては、ソース・イメージの処理部分のサイズを指定する。出力パラメータとしては、出力部分のサイズが返される。*stage* パラメータの説明も参照のこと。
pState	関数 ippiMorphologyInitAlloc で、*elShape* に IPPI_SHAPE_RECT を指定し、適切な値の *dataType* と *channels*、および十分な ROI 幅を指定して初期化したモルフォロジー状態構造体へのポインタ
stage	次のいずれかで示される処理段階
IPPI_WHOLE	関数は画像全体を処理する。
IPPI_START	関数はサイクリック・バッファを初期化し、大きい画像の上部を処理する。出力部分は、ソース部分より小さいか、構造要素の高さが 1 の場合は等しくなる。

IPPI_END	関数はサイクリック・バッファをフラッシュし、大きい画像の下部を処理する。出力部分は、ソース部分より大きいか、構造要素の高さが 1 の場合は等しくなる。
IPPI_MIDDLE	関数は大きい画像の中央部分を処理する。出力部分は、ソース部分と同じサイズになる。

説明

関数 ippiDilateStrip_Rect は、ippcv.h ファイルの中で宣言される。この関数は、矩形の構造要素を使用して、ソース・イメージの ROI を膨張（すなわち、最大値フィルタを適用）する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize.width* が正でないか、または *roiSize.height* が負であるエラーを示す。
ippStsStepErr	*srcStep* または *dstStep* が *roiSize.width * <pixelSize>* より小さいエラーを示す。
ippStsNotEvenStepErr	浮動小数点画像のステップが 4 で割り切れない場合のエラーを示す。
ippStsOutOfRangeErr	*stage* が IPPI_START、IPPI_MIDDLE、IPPI_END、または IPPI_WHOLE でないエラーを示す。
ippStsContextMatchErr	呼び出された関数のパラメータと、ippiMorphologyInitAlloc に渡されたパラメータが一致しないエラーを示す。

DilateStrip_Cross

クロスシェイプの構造要素を使用して、画像を膨張する。

```
IppStatus ippiDilateStrip_Cross_<mod>(const Ipp<DataType>* pSrc,
      int srcStep, Ipp<DataType>* pDst, int dstStep,
      IppiSize* pRoiSize, IppiMorphState* pState, int stage);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pRoiSize	ピクセル数で示さえれた入出力の画像 ROI サイズを返す変数へのポインタ。入力パラメータとしては、ソース・イメージの処理部分のサイズを指定する。出力パラメータとしては、出力部分のサイズが返される。*stage* パラメータの説明も参照のこと。
pState	関数 ippiMorphologyInitAlloc、*elShape*IPPI_SHAPE_CROSS を指定し、適切な値の *dataType* と *channels*、および十分な ROI 幅を指定して初期化したモルフォロジー状態構造体へのポインタ
stage	次のいずれかで示される処理段階
IPPI_WHOLE	関数は画像全体を処理する。
IPPI_START	関数はサイクリック・バッファを初期化し、大きい画像の上部を処理する。出力部分は、ソース部分より小さいか、構造要素の高さが 1 の場合は等しくなる。

IPPI_END	関数はサイクリック・バッファをフラッシュし、大きい画像の下部を処理する。出力部分は、ソース部分より大きいか、構造要素の高さが 1 の場合は等しくなる。
IPPI_MIDDLE	関数は大きい画像の中央部分を処理する。出力部分は、ソース部分と同じサイズになる。

説明

関数 ippiDilateStrip_Cross は、ippcv.h ファイルの中で宣言される。この関数は、クロスシェイプの構造要素を使用して、ソース・イメージ ROI で膨張（すなわち、最大値フィルタを適用）する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize*.width が正でないか、または *roiSize.height* が負であるエラーを示す。
ippStsStepErr	*srcStep* または *dstStep* が *roiSize.width* * *<pixelSize>* より小さいエラーを示す。
ippStsNotEvenStepErr	浮動小数点画像のステップが 4 で割り切れない場合のエラーを示す。
ippStsOutOfRangeErr	*stage* が IPPI_START、IPPI_MIDDLE、IPPI_END、または IPPI_WHOLE でないエラーを示す。
ippStsContextMatchErr	呼び出された関数のパラメータと、ippiMorphologyInitAlloc に渡されたパラメータが一致しないエラーを示す。

フィルタリング関数 9

本章では、画像に対して、線形および非線形のフィルタリング操作を実行するインテル® IPP 画像処理関数について説明する。
フィルタリングは、縁取り、ぼかし、ノイズ除去、特徴抽出など、さまざまな画像処理操作に使用できる。

表 9-1 に、本章で詳しく説明する関数の一覧を示す。

表 9-1　フィルタリング関数

関数の基本名	操作
FilterBox	ボックス・フィルタを使用して、画像にぼかしを入れる。
FilterMin	最小値フィルタを使用して、画像をフィルタリングする。
FilterMax	最大値フィルタを使用して、画像をフィルタリングする。
メディアン・フィルタ	
FilterMedian	画像にメディアン・フィルタを適用する。
FilterMedianHoriz	水平マスクを持つメディアン・フィルタを使用して、画像をフィルタリングする。
FilterMedianVert	垂直マスクを持つメディアン・フィルタを使用して、画像をフィルタリングする。
FilterMedianCross	クロス・メディアン・フィルタを使用して、画像をフィルタリングする。
FilterMedianColor	画像にカラー・メディアン・フィルタを適用する。
汎用線形フィルタ	
Filter	汎用の矩形たたみ込みカーネルを使用して、画像をフィルタリングする。
Filter32f	浮動小数点の矩形たたみ込みカーネルを使用して、整数データの画像をフィルタリングする。
分離可能フィルタ	
FilterColumn	整数の列たたみ込みカーネルを使用して、画像をフィルタリングする。
FilterColumn32f	浮動小数点の列たたみ込みカーネルを使用して、画像をフィルタリングする。
FilterRow	整数の行たたみ込みカーネルを使用して、画像をフィルタリングする。
FilterRow32f	浮動小数点の行たたみ込みカーネルを使用して、画像をフィルタリングする。

続く

表 9-1　フィルタリング関数（続き）

関数の基本名	操作
2D たたみ込み	
ConvFull	2 つの画像の完全たたみ込みを実行する。
ConvValid	2 つの画像の有効たたみ込みを実行する。
固定フィルタ	
FilterPrewittHoriz	水平 Prewitt 演算子を使用して、画像をフィルタリングする。
FilterPrewittVert	垂直 Prewitt 演算子を使用して、画像をフィルタリングする。
FilterSharrHoriz	水平 Scharr 演算子を使用して、画像をフィルタリングする。
FilterSharrVert	垂直 Scharr 演算子を使用して、画像をフィルタリングする。
FilterSobelHoriz、FilterSobelHorizMask	水平 Sobel 演算子を使用して、画像をフィルタリングする。
FilterSobelVert、FilterSobelVertMask	垂直 Sobel 演算子を使用して、画像をフィルタリングする。
FilterSobelHorizSecond	二次導関数の水平 Sobel 演算子を使用して、画像をフィルタリングする。
FilterSobelVertSecond	二次導関数の垂直 Sobel 演算子を使用して、画像をフィルタリングする。
FilterSobelCross	二次導関数のクロス Sobel 演算子を使用して、画像をフィルタリングする。
FilterRobertsDown	水平 Roberts クロスグラディエント演算子を使用して、画像をフィルタリングする。
FilterRobertsUp	垂直 Roberts クロスグラディエント演算子を使用して、画像をフィルタリングする。
FilterLaplace	ラプラシアン・カーネルを使用して、画像をフィルタリングする。
FilterGauss	ガウシアン・カーネルを使用して、画像をフィルタリングする。
FilterHipass	画像にハイパス・フィルタを適用する。
FilterLowpass	画像にローパス・フィルタを適用する。
FilterSharpen	画像に鮮明化フィルタを適用する。

ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。ほとんどのフィルタリング関数が、ソースとデスティネーションで異なるイメージ・バッファを使用する（すなわち、非インプレース操作を行う）。

境界

本章で説明するフィルタリング関数は、隣接領域の操作を実行する（第 2 章の「隣接操作」を参照）。これらの関数は、これらの関数は、処理される各ピクセルについて、その操作に必要な、すべての参照先の隣接ピクセルも利用可能である前提に基づいて動作する。

各ピクセルの隣接領域は、フィルタ・カーネル（またはマスク）のサイズとアンカ・セルの位置によって定義される。アンカ・セル（図 9-1 a を参照）は、、カーネル内の固定されたセルである。アンカ・セルは、ソース・イメージの現在処理中のピクセルを基準とする、カーネルの位置指定に使用される。具体的には、アンカ・セルと入力ピクセルの位置が一致するように、カーネルが画像上に配置される。

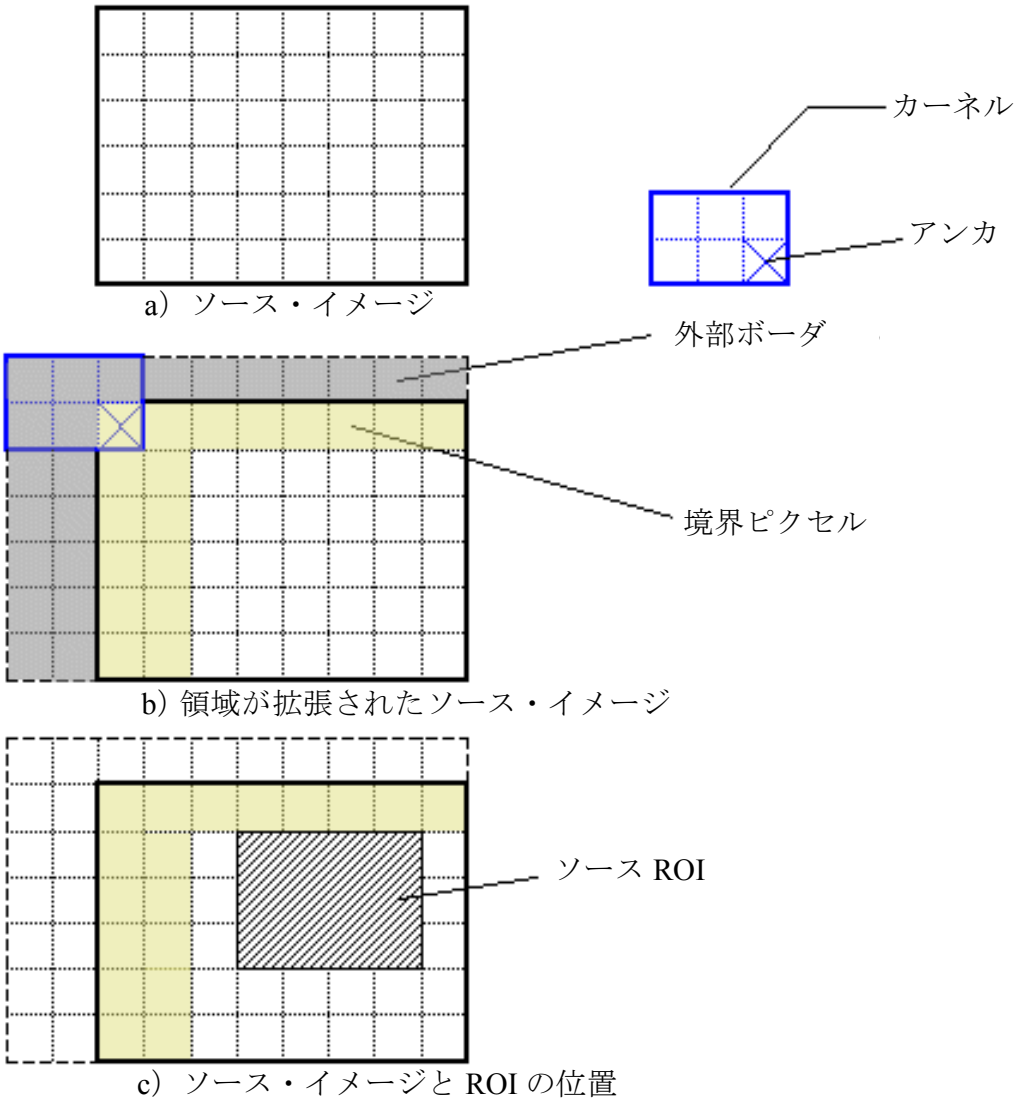
図 9-1 b に示すように、入力ピクセルが画像の水平または垂直の縁の近くにある場合、重ねられるカーネルは、ソース・イメージ内に存在しない（つまり、画像領域の外側にある）隣接ピクセルを参照することがある。
図 9-1 b では、指定されたカーネルとアンカで隣接領域の操作を実行する際に画像内に存在しないピクセルを必要とする、ソース・イメージのすべての境界ピクセルを黄色で示す。また、すべてのスキャンされる外部ピクセル（ボーダ・ピクセルと呼ばれる）の集合体をグレーで示す。

したがって、ソース・イメージ全体にフィルタリング関数を適用する場合は、最初にその操作に必要な追加のボーダ・ピクセルを計算し、次に何らかの方法でこれらの存在しないピクセルを定義する必要がある。これを行うには、インテル IPP 関数 `ippiCopyConstBorder` または `ippiCopyReplicateBorder` を使用できる。これらの関数は、拡張された画像の境界をユーザ定義のピクセル値で充填するか、ユーザ独自の拡張方法を適用する。
ただし、フィルタリング関数を呼び出す前に必要なボーダ・ピクセルが定義されていない場合は、メモリ違反エラーが発生する。

一方、ソース・イメージの一部または ROI（第 2 章の「インテル® IPP の処理対象領域」を参照）に対してフィルタリング操作を実行する場合、ボーダ・ピクセルを追加して画像領域を拡張する必要があるかどうかは、画像内の ROI のサイズと位置によって決まる。
図からただちにわかるように （図 9-1 c）、ROI と黄色の （内部の境界）ピクセルが重ならない場合は、外部のピクセルはスキャンされない。この場合は、境界の拡張は不要である。

境界ピクセルが ROI の一部である場合は、ソース・イメージの領域の拡張が必要である。

図 9-1　　隣接領域の操作に使用される境界



結論として、フィルタリング操作で有効な結果を得るには、アプリケーションは、フィルタリング関数に渡される ROI パラメータの値をチェックして、すべての必要な隣接ピクセルが実際に画像内に存在するかどうかを確認し、必要に応じて、存在しないピクセルを定義しなければならない。

FilterBox

単純ボックス・フィルタを使用して、画像にぼかしを入れる。

事例 1：非インプレース操作

```
IppStatus ippiFilterBox_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiSize maskSize, IppiPoint anchor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

事例 2：インプレース操作

```
IppStatus ippiFilterBox_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16s_AC4IR	32f_AC4IR

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	デスティネーション ROI のサイズ（ピクセル単位）
roiSize	（インプレース操作の場合）ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
pSrcDst	（インプレース操作の場合）ソース・バッファとデスティネーション・バッファへのポインタ

srcDstStep	（インプレース操作の場合）ソース・イメージ・バッファとデスティネーション・イメージ・バッファ内のステップ（バイト単位）
maskSize	マスクのサイズ（ピクセル単位）
anchor	入力ピクセル位置に対するマスク・アライメントを指定するアンカ・セル

説明

関数 ippiFilterBox は、ippi.h ファイルの中で宣言される。この関数は、出力画像内の各ピクセルに、入力画像における隣接矩形領域 （対象ピクセルをアンカ・セルとして、サイズを *maskSize* とする領域）内の全ピクセルの平均値を設定する。このフィルタは、入力画像にスムージングまたはぼかしの効果を与える。画像の境界ピクセルを処理するとき、有効な操作を保証するには、アプリケーションが追加のボーダ・ピクセルを正しく定義する必要がある（「境界」を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールド、または *roiSize* フィールド値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *SrcDstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*maskSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsAnchorErr	*anchor* がマスク・サイズの外側に位置する場合のエラー状態を示す。

FilterMin

画像に「最小値」フィルタを適用する。

```
IppStatus ippiFilterMin_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiSize maskSize, IppiPoint anchor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
maskSize	マスクのサイズ（ピクセル単位）
anchor	入力ピクセル位置に対するマスク・アライメントを指定するアンカ・セル

説明

関数 ippiFilterMin は、ippi.h ファイルの中で宣言される。この関数は、出力画像内の各ピクセルに、入力画像における隣接領域（対象ピクセルをアンカ・セルとして、サイズを *maskSize* とする領域）内の全ピクセルの最小値を設定する。このフィルタは、入力画像のコントラストを低くする効果がある。画像の境界ピクセルを処理するとき、有効な操作を保証するには、アプリケーションが追加のボーダ・ピクセルを正しく定義する必要がある（「境界」を参照）。

次のコード例は、ippiFilterMin 関数の使用方法を示している。この関数は、ROI 外部の隣接ピクセルがすべて使用可能であることを前提にしているのに注意する。

例 9-1　最小値フィルタの適用

```
IppStatus filterMin( void ) {
   Ipp8u x[5*4], y[5*4]={0};
   IppiSize img={5,4}, roi={3,2}, mask={3,3};
   IppiPoint anchor = {1,1};
   ippiSet_8u_C1R( 3, x, 5, img );
   /// set a hole (1) at row 1 and column 1.
   /// The value will be extended on filter
   /// mask area, depending on anchor
   x[5+1] = 1;
   /// ROI is inside the image.
   /// Offset pointers to jump at the ROI start
   return ippiFilterMin_8u_C1R( x+6, 5, y+6, 5, roi,
mask, anchor );
}
```

デスティネーション・イメージ *y* には、次のデータが含まれる。

```
00 00 00 00 00
00 01 01 03 00
00 01 01 03 00
00 00 00 00 00
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pSrc* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*maskSize* フィールドの値が 0 または負の場合のエラー状態を示す。

ippStsAnchorErr	*anchor* がマスク・サイズの外側に位置する場合のエラー状態を示す。

FilterMax

画像に「最大値」フィルタを適用する。

```
IppStatus ippiFilterMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
      IppiSize maskSize, IppiPoint anchor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
maskSize	マスクのサイズ（ピクセル単位）
anchor	入力ピクセル位置に対するマスク・アライメントを指定するアンカ・セル

説明

関数 ippiFilterMax は、ippi.h ファイルの中で宣言される。この関数は、出力画像内の各ピクセルに、入力画像における隣接領域 （対象ピクセルをアンカ・セルとして、サイズを *maskSize* とする領域）内の全ピクセルの最大値を設定する。こ

のフィルタは、入力画像のコントラストを高くする効果がある。画像の境界ピクセルを処理するとき、有効な操作を保証するには、アプリケーションが追加のボーダ・ピクセルを正しく定義する必要がある（「境界」を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*maskSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsAnchorErr	*anchor* がマスク・サイズの外側に位置する場合のエラー状態を示す。

メディアン・フィルタ

メディアン・フィルタ関数は、ソース・イメージ・データに対して非線形フィルタリングを実行する。

メディアン・フィルタ関数は、任意の矩形マスクを使用するか、次の IppiMaskSize 型の定義済みマスクを使用して、画像をフィルタリングする。

ippMskSize3x1	長さが 3 の水平マスク
ippMskSize5x1	長さが 5 の水平マスク
ippMskSize1x3	長さが 3 の垂直マスク
ippMskSize3x3	サイズが 3 の正方形マスク
ippMskSize1x5	長さが 5 の垂直マスク
ippMskSize5x5	サイズが 5 の正方形マスク

隣接領域のサイズと、隣接領域におけるアンカ・セル座標は、*mask* タイプで決まる。表 9-2 に、マスク・タイプと、その隣接領域サイズおよびアンカ・セル座標の一覧を示す。マスク名の中にマスク・サイズが（*XY*）の順に示されていることと、座標は左上隅が [0, 0] であることに注意する。

表 9-2　メディアン・フィルタのマスク、隣接領域、アンカ・セル

マスク	隣接領域のサイズ		アンカ・セル
	列数	行数	
ippMskSize3x1	3	1	[1, 0]
ippMskSize5x1	5	1	[2, 0]
ippMskSize1x3	1	3	[0, 1]
ippMskSize3x3	3	3	[1, 1]
ippMskSize1x5	1	5	[0, 2]
ippMskSize5x5	5	5	[2, 2]

メディアン・フィルタは、孤立した突出点を取り除く効果を持ち、画像のノイズ除去に使用できる。

インテル IPP の中でメディアン･フィルタリングに使用されているアルゴリズムの詳細については、[APMF] を参照のこと。

FilterMedian

メディアン・フィルタを使用して、
画像をフィルタリングする。

```
IppStatus ippiFilterMedian_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize
      dstRoiSize, IppiSize maskSize, IppiPoint anchor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R
8u_C3R      16s_C3R
8u_C4R      16s_C4R
8u_AC4R     16s_AC4R
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
maskSize	マスクのサイズ（ピクセル単位）
anchor	入力ピクセル位置に対するマスク・アライメントを指定するアンカ・セル

説明

関数 ippiFilterMedian は、ippi.h ファイルの中で宣言される。この関数は、出力バッファ内の各ピクセルに、入力ピクセル隣接領域内の全ピクセルの中央値を設定する。画像の境界ピクセルを処理するとき、有効な操作を保証するには、アプリケーションが追加のボーダ・ピクセルを正しく定義する必要がある（「境界」を参照）ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。
次の例は、メディアン・フィルタの使用方法を示している。メディアン・フィルタはノイズを除去するが、アベレージング・フィルタのように信号輝度を低下させることはない点に注意する。

例 9-2　画像へのメディアン・フィルタの適用

```
IppStatus filterMedian( void ) {
   IppiPoint anchor = { 1,1 };
   Ipp8u x[5*4], y[5*4]={0};
   IppiSize img={3,4}, roi={3,2}, mask={3,3};
   ippiSet_8u_C1R( 0x10, x, 5, img );
   /// raise the level of the signal,
   /// The edge will not be destroyed by the filter
   img.width = 5-3;
   ippiSet_8u_C1R( 0x40, x+3, 5, img );
   /// a spike, will be filtered
   x[5+1] = 0;
   /// roi is inside the image.
   /// Offset pointers to jump at the roi's beginning
  return ippiFilterMedian_8u_C1R( x+6, 5, y+6, 5, roi,
mask, anchor );
```

} デスティネーション・イメージ y には、次のデータが含まれる。

```
00 00 00 00 00
00 10 10 40 00
00 10 10 40 00
00 00 00 00 00
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

ippStsMaskSizeErr	*maskSize* フィールドの値が 0、負、または偶数の場合のエラー状態を示す。
ippStsAnchorErr	*anchor* がマスク・サイズの外側に位置する場合のエラー状態を示す。

FilterMedianHoriz

水平メディアン・フィルタを使用して、画像をフィルタリングする。

```
IppStatus ippiFilterMedianHoriz_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R
8u_AC4R	16s_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
mask	IppiMaskSize 型の定義済みマスク

説明

関数 ippiFilterMediaNhoriz は、ippi.h ファイルの中で宣言される。この関数は、出力バッファ内の各ピクセルに、入力ピクセル隣接領域内の全ピクセルの中央値を設定する。隣接領域の水平方向のサイズとアンカ・セル座標は、mask タイプ（ippMskSize3x1 または ppMskSize5x1）によって決まる（表 9-2 を参照）。この関数は、ソース・イメージ ROI の外部でも、マスク・サイズの半分までの距離にはピクセルが存在することを前提としている。したがって、アプリケーション・プログラムでは、*pSrc* と *dstRoiSize* の 2 つの引数に適切な値を設定するか、追加の境界ピクセルを設定する必要がある（「境界」を参照）ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*mask* の値が不当である場合のエラー状態を示す。

FilterMedianVert

垂直メディアン・フィルタを使用して、画像をフィルタリングする。

```
IppStatus ippiFilterMedianVert_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R
8u_AC4R	16s_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
mask	IppiMaskSize 型の定義済みマスク

説明

関数 ippiFilterMedianVert は、ippi.h ファイルの中で宣言される。この関数は、出力バッファ内の各ピクセルに、入力ピクセル隣接領域内の全ピクセルの中央値を設定する。隣接領域の垂直方向のサイズとアンカ・セル座標は、mask タイプ（ippMskSize1x3 または ippMskSize1x5）によって決まる（表 9-2 を参照）。この関数は、ソース・イメージ ROI の外部でも、マスク・サイズの半分までの距離にはピクセルが存在することを前提としている。したがって、アプリケーション・プログラムでは、*pSrc* と *dstRoiSize* の 2 つの引数に適切な値を設定するか、追加の境界ピクセルを設定する必要がある（「境界」を参照）ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*mask* の値が不当である場合のエラー状態を示す。

FilterMedianCross

クロス・メディアン・フィルタを
使用して、画像をフィルタリングする。

```
IppStatus ippiFilterMedianCross_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_AC4R	16s_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
mask	IppiMaskSize 型の定義済みマスク

説明

関数 ippiFilterMedianCross は、ippi.h ファイルの中で宣言される。この関数は、出力バッファ内の各ピクセルに、入力ピクセル隣接領域内の全ピクセルの中央値を設定する。隣接領域は、定義済みの正方形マスク （ippMskSize3x3 または ippMskSize5x5）によって決まる（表 9-2 を参照）。この関数は、ソース・イメージ ROI の外部でも、マスク・サイズの半分までの距離にはピクセルが存在することを前提としている。したがって、アプリケーション・プログラムでは、*pSrc* と *dstRoiSize* の2つの引数に適切な値を設定するか、追加の境界ピクセルを設定する必要がある（「境界」を参照）ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*mask* の値が不当である場合のエラー状態を示す。

FilterMedianColor

カラー・メディアン・フィルタを
使用して、画像をフィルタリングする。

```
IppStatus ippiFilterMedianColor_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
mask	IppiMaskSize 型の定義済みマスク

説明

関数 ippiFilterMedianColor は、ippi.h ファイルの中で宣言される。これまでに説明したフィルタリング関数をカラー・イメージに適用すると、画像の各カラー・プレーンが個別に処理され、カラー・コンポーネント間の相関が失われる。相関情報を保持したい場合は、ippiFilterMedianColor 関数を使用する。この関数は、*mask* で指定される隣接領域内の各ピクセルと入力ピクセルの差を、赤 (R)、緑 (G)、青 (B) のカラー・コンポーネントごとに計算する。入力ピクセル *i* と隣接ピクセル *j* の差は、次の式に示す絶対差の総和で求められる。

```
abs (R(i)-R(j)) + abs (G(i)-G(j)) + abs (B(i)-B(j))
```

隣接領域内をすべてスキャンし、*i* との差が最小の隣接ピクセルの値を、ピクセル *i* の出力値とする。

関数 ippiFilterMedianColor では、ippMskSize3x3 または ippMskSize5x5 の定義済み正方形マスクを使用でき、カラー・イメージのみを処理できる。画像の境界ピクセルを処理するとき、有効な操作を保証するには、アプリケーションが追加のボーダ・ピクセルを正しく定義する必要がある (「境界」を参照)。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*mask* の値が不当である場合のエラー状態を示す。

汎用線形フィルタ

ここに示す関数は、汎用矩形カーネルを使用して、画像をフィルタリングする。カーネルは、符号付き整数値または単精度実数値のマトリックスである。各入力ピクセルに対して、カーネル内の固定アンカ・セルが入力ピクセルと一致するように、カーネルを配置する。アンカ・セルは一般的にカーネルの幾何学的な中心であるが、中心からずらすこともできる。

カーネル値の配列へのポインタが、フィルタリング関数に渡される。カーネル値は、左上隅から行優先順序で読み込まれる。この配列には、*kernelSize.width * kernelSize.height* 個のエントリが入っている。アンカ・セルは、カーネルの左上隅を原点とする座標系を使用して、*anchor.x* および *anchor.y* の座標で指定する。

出力値は、カーネル・マトリックスを重み係数とした隣接ピクセル値の総和で計算される。この計算式は、たたみ込み演算を行うものなので、カーネル係数が逆順に使用されることに注意する。オプションで、出力ピクセルをスケーリングすることもできる。画像の境界ピクセルを処理するとき、有効な操作を保証するには、アプリケーションが追加のボーダ・ピクセルを正しく定義する必要がある（「境界」を参照）。

Filter

汎用矩形フィルタを使用して、
画像をフィルタリングする。

事例 1：整数データの操作

```
IppStatus ippiFilter_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32s* pKernel, IppiSize kernelSize, IppiPoint anchor,
    int divisor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R
8u_AC4R	16s_AC4R

事例 2：浮動小数点データの操作

```
IppStatus ippiFilter_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32f* pKernel, IppiSize kernelSize, IppiPoint anchor);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
pKernel	カーネル値へのポインタ
kernelSize	矩形カーネルのサイズ（ピクセル単位）
anchor	入力ピクセル位置に対する矩形カーネル・アライメントを指定するアンカ・セル
divisor	（整数データを処理する場合のみ）計算結果を割る整数値

説明

関数 ippiFilter は、ippi.h ファイルの中で宣言される。この関数は、*kernelSize* で指定されたサイズの汎用矩形カーネルを使用して、画像をフィルタリングする。この関数は、*kernelSize* と *anchor* で決まるソース・ピクセル隣接領域内の全ピクセルに対して、カーネル係数 *pKernel* とピクセル値の積を合計する。
このとき、カーネル係数が逆順に使用されることに注意する。求められた合計値がデスティネーション・ピクセルに書き込まれる。整数データの場合は、*divisor* で指定された固定のスケール係数で、計算結果を割る。
画像の境界ピクセルを処理するとき、有効な操作を保証するには、アプリケーションが追加のボーダ・ピクセルを正しく定義する必要がある（「境界」を参照）。

整数データを処理する場合、画像 ROI 内のピクセル $X_{i,j}$ に対する出力ピクセル $Y_{i,j}$ は、次の式で計算される。

$$Y_{i,j} = \frac{1}{divisor} \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} X_{i+n-Q,\, j+m-P} \times K_{W-n-1,\, H-m-1}$$

ここで、

$K_{n,m}$ はカーネル値である。

H = *kernelSize.height* はカーネルの垂直方向のサイズである。

W = *kernelSize.width* はカーネルの水平方向のサイズである。

P = H – *anchor.y* – 1

Q = W – *anchor.x* – 1

Ipp32f 型の入力データを処理する関数も同様の計算式を使用するが、合計値のスケーリングは行わない。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pKernel* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* または *kernelSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsDivisorErr	*divisor* 値が 0 である場合のエラー状態を示す。

Filter32f

浮動小数点型の矩形カーネルを使用して、画像をフィルタリングする。

```
IppStatus ippiFilter32f_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32f* pKernel, IppiSize kernelSize, IppiPoint anchor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R
8u_AC4R	16s_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
pKernel	カーネル値へのポインタ
kernelSize	矩形カーネルのサイズ（ピクセル単位）
anchor	入力ピクセル位置に対する矩形カーネル・アライメントを指定するアンカ・セル

説明

関数 ippiFilter32f は、ippi.h ファイルの中で宣言される。この関数は、浮動小数点値を持つ矩形カーネルを使用して、整数データの画像をフィルタリングする。この関数は、*kernelSize* と *anchor* で決まるソース・ピクセル隣接領域内の全ピクセルに対して、カーネル係数 *pKernel* とピクセル値の積を合計する。このとき、カーネル係数が逆順に使用されることに注意する。求められた合計値がデスティネーション・ピクセルに書き込まれる。合計値を求める式は、ippiFilter 関数で使用する式に似ているが、合計値のスケーリングは行わない。画像の境界ピクセルを処理するとき、有効な操作を保証するには、アプリケーションが追加のボーダ・ピクセルを正しく定義する必要がある（「境界」を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pKernel* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* または *kernelSize* フィールドの値が 0 または負の場合のエラー状態を示す。

分離可能フィルタ

分離可能フィルタは、1 列 （FilterColumn 関数など）または 1 行 （FilterRow 関数など）の空間カーネルを使用して、ソース・イメージをフィルタリングする。

FilterColumn

1 列の空間カーネルを使用して、
画像をフィルタリングする。

事例 1：整数データの操作

```
IppStatus ippiFilterColumn_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32s* pKernel, int kernelSize, int yAnchor,
    int divisor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R
8u_AC4R	16s_AC4R

事例 2：浮動小数点データの操作

```
IppStatus ippiFilterColumn_<mod>(const Ipp32f* pSrc, int
      srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize,
      const Ipp32f* pKernel, int kernelSize, int yAnchor);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ

dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
pKernel	カーネル値へのポインタ
kernelSize	カーネルのサイズ（ピクセル単位）
yAnchor	入力ピクセル位置に対する垂直カーネル・アライメントを指定するアンカ・セル
divisor	（整数データを処理する場合のみ）計算結果を割る整数値

説明

関数 ippiFilterColumn は、ippi.h ファイルの中で宣言される。この関数は、垂直の列カーネル（サイズは *kernelSize*）を使用して、画像をフィルタリングする。この関数は、*kernelSize* と *yAnchor* で決まるソース・ピクセル隣接領域内の全ピクセルに対して、カーネル係数 *pKernel* とピクセル値の積を合計する。このとき、カーネル係数が逆順に使用されることに注意する。整数データの場合は、*divisor* で指定された固定のスケール係数で、計算結果を割る。画像の境界ピクセルを処理するとき、有効な操作を保証するには、アプリケーションが追加のボーダ・ピクセルを正しく定義する必要がある（「境界」を参照）。

整数データを処理する場合、画像 ROI 内のピクセル $X_{i,j}$ に対する出力ピクセル $Y_{i,j}$ は、次の式で計算される。

$$Y_{i,j} = \frac{1}{divisor} \sum_{m=0}^{H-1} X_{i,j+m-P} \times K_{H-m-1}$$

ここで、

K_m はカーネル値である。
H = *kernelSize* は垂直の列カーネルのサイズである。
P = H – *yAnchor* – 1

Ipp32f 型の入力データを処理する関数も同様の計算式を使用するが、合計値のスケーリングは行わない。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pKernel* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsDivisorErr	*divisor* 値が 0 である場合のエラー状態を示す。

FilterColumn32f

浮動小数点型の列カーネルを使用して、整数データの画像をフィルタリングする。

```
IppStatus ippiFilterColumn32f_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32f* pKernel, int kernelSize, int yAnchor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R
8u_AC4R	16s_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
pKernel	列カーネル値へのポインタ
kernelSize	カーネルのサイズ（ピクセル単位）

yAnchor	入力ピクセル位置に対する垂直カーネル・アライメントを指定するアンカ・セル

説明

関数 ippiFilterColumn32f は、ippi.h ファイルの中で宣言される。この関数は、浮動小数点値を持つ垂直の列カーネルを使用して、整数データの画像をフィルタリングする。この関数は、*kernelSize* と *yAnchor* で決まるソース・ピクセル隣接領域内の全ピクセルに対して、カーネル係数 *pKernel* とピクセル値の積を合計する。このとき、カーネル係数が逆順に使用されることに注意する。合計値を求める式は、ippiFilterColumn 関数で使用する式に似ているが、合計値のスケーリングは行わない。画像の境界ピクセルを処理するとき、有効な操作を保証するには、アプリケーションが追加のボーダ・ピクセルを正しく定義する必要がある（「境界」を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pKernel* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。

FilterRow

1 行の空間カーネルを使用して、画像をフィルタリングする。

事例 1：整数データの操作

```
IppStatus ippiFilterRow_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32s* pKernel, int kernelSize, int xAnchor,
    int divisor);
```

サポートされる *mod* の値は、次のとおりである。

事例 2：浮動小数点データの操作

```
IppStatus ippiFilterRow_<mod>(const Ipp32f* pSrc, int
      srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize,
      const Ipp32f* pKernel, int kernelSize, int xAnchor);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
pKernel	カーネル値へのポインタ
kernelSize	カーネルのサイズ（ピクセル単位）
xAnchor	入力ピクセル位置に対する水平カーネル・アライメントを指定するアンカ・セル
divisor	（整数データを処理する場合のみ）計算結果を割る整数値

説明

関数 ippiFilterRow は、ippi.h ファイルの中で宣言される。この関数は、水平の行カーネル（サイズは *kernelSize*）を使用して、画像をフィルタリングする。この関数は、*kernelSize* と *xAnchor* で決まるソース・ピクセル隣接領域内の全ピクセルに対して、カーネル係数 *pKernel* とピクセル値の積を合計する。このとき、カーネル係数が逆順に使用されることに注意する。

整数データの場合は、*divisor* で指定された固定のスケール係数で、計算結果を割る。画像の境界ピクセルを処理するとき、有効な操作を保証するには、アプリケーションが追加のボーダ・ピクセルを正しく定義する必要がある（「境界」を参照）。

整数データを処理する場合、画像 ROI 内のピクセル $X_{i,j}$ に対する出力ピクセル $Y_{i,j}$ は、次の式で計算される。

$$Y_{i,j} = \frac{1}{divisor} \sum_{n=0}^{W-1} X_{i+n-Q,j} \times K_{W-n-1}$$

ここで、

K_n はカーネル値である。

W = *kernelSize* は水平の行カーネルのサイズである。

Q = W – *xAnchor* – 1

Ipp32f 型の入力データを処理する関数も同様の計算式を使用するが、合計値のスケーリングは行わない。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pKernel* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsDivisorErr	*divisor* 値が 0 である場合のエラー状態を示す。

FilterRow32f

浮動小数点型の行カーネルを使用して、整数データの画像をフィルタリングする。

```
IppStatus ippiFilterRow32f_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32f* pKernel, int kernelSize, int xAnchor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R
8u_AC4R	16s_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
pKernel	行カーネル値へのポインタ
kernelSize	カーネルのサイズ（ピクセル単位）
xAnchor	入力ピクセル位置に対する水平カーネル・アライメントを指定するアンカ・セル

説明

関数 ippiFilterRow32f は、ippi.h ファイルの中で宣言される。この関数は、浮動小数点値を持つ水平の行カーネルを使用して、整数データの画像をフィルタリングする。この関数は、*kernelSize* と *xAnchor* で決まるソース・ピクセル隣接領域内の全ピクセルに対して、カーネル係数 *pKernel* とピクセル値の積を合計する。このとき、カーネル係数が逆順に使用されることに注意する。合計値を求める式は、ippiFilterRow 関数で使用する式に似ているが、合計値のスケーリングは行わない。画像の境界ピクセルを処理するとき、有効な操作を保証するには、アプリケーションが追加のボーダ・ピクセルを正しく定義する必要がある（「境界」を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pKernel* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。

2D たたみ込み

本節で説明するインテル IPP 関数は、2 つのソース・イメージに対して二次元の有限線形たたみ込み演算を行い、その結果をデスティネーション・イメージに書き込む。たたみ込みは、鮮明化、ぼかし、ノイズ除去、エンボス、エッジ強調など、よく行われる画像処理に使用されている。

本節では、便宜上、任意のデジタル・イメージ f を、M_f 列、N_f 行のマトリックスで表す。個々のピクセル値は、$f[i,j]$、$0 \le i < M_f$、$0 \le j < N_f$ と表される。

ConvFull

2 つの画像の完全たたみ込みを実行する。

事例 1：整数データの操作

```
IppStatus ippiConvFull_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize src2Size, Ipp<datatype>* pDst, int dstStep,
    int divisor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_AC4R	16s_AC4R

事例 2：浮動小数点データの操作

```
IppStatus ippiConvFull_<mod>(const Ipp32f* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp32f* pSrc2, int src2Step, IppiSize
    src2Size, Ipp32f* pDst, int dstStep);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R
32f_C3R
32f_AC4R
```

引数

pSrc1, pSrc2	ソース・バッファへのポインタ
src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
src1Size, src2Size	ソース・イメージのサイズ（ピクセル単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
divisor	（整数データを処理する場合のみ）計算結果を割る整数値

説明

関数 ippiConvFull は、ippi.h ファイルの中で宣言される。この関数は、*pSrc1* と *pSrc2* が指す 2 つのソース・イメージに対して、二次元の完全有限線形たたみ込み演算を実行する。最初のソース・イメージをマトリックス f（サイズは $M_f \times N_f$）で表し、2 番目のソース・イメージをマトリックス g（サイズは $M_g \times N_g$）で表すと、このたたみ込み関数で得られるデスティネーション・イメージ h のサイズは、$M_h \times N_h$ となる。ただし、$M_h = M_f + M_g - 1$、$N_h = N_f + N_g - 1$ である。

関数 ippiConvFull は次の式を使用して、デスティネーション・イメージの値 $h[i,j]$ を計算する。

$$h[i,j] = \frac{1}{divisor} \sum_{l=0}^{N_h-1} \sum_{k=0}^{M_h-1} f[k,l] \times g[i-k, j-l] \quad ,$$

ここで $0 \le i < M_h$, $0 \le j < N_h$ および

$$f[k,l] = \begin{cases} f[k,l], & 0 \le k < M_f; \quad 0 \le l < N_f \\ 0 \quad , otherwise \end{cases}$$

$$g[i-k, j-l] = \begin{cases} g[i-k, j-l], & 0 \le i-k < M_g; \quad 0 \le j-l < N_g \\ 0 \quad , otherwise \end{cases}$$

Ipp32f 型の入力データを処理する関数も同様の計算式を使用するが、合計値のスケーリングは行わない（$divisor = 1$ となる）。

このたたみ込み関数の機能を説明するために、次のようなソース・イメージ f, g（サイズは 3 × 5）があるとする。

$$f = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad , \quad g = f$$

たたみ込みを実行して得られた画像 *h* のサイズは 5 × 9 となり、次のようなデータが含まれる。

$$h = \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 2 & 2 & 0 & 0 \\ 3 & 4 & 6 & 4 & 2 \\ 2 & 2 & 4 & 2 & 2 \\ 3 & 6 & 11 & 6 & 3 \\ 2 & 2 & 4 & 2 & 2 \\ 2 & 4 & 6 & 4 & 3 \\ 0 & 0 & 2 & 2 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc1*、*pSrc2*、または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*Src1Size* または *Src2Size* の値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*src1Step*、*src2Step*、または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsDivisorErr	*divisor* の値が 0 の場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

ConvValid

2 つの画像の有効たたみ込みを実行する。

事例 1：整数データの操作

```
IppStatus ippiConvValid_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, IppiSize src1Size, const Ipp<datatype>* pSrc2,
    int src2Step, IppiSize src2Size, Ipp<datatype>* pDst,
    int dstStep, int divisor);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_AC4R	16s_AC4R

事例 2：浮動小数点データの操作

```
IppStatus ippiConvValid_<mod>(const Ipp32f* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp32f* pSrc2, int src2Step,  IppiSize
    src2Size, Ipp32f* pDst, int dstStep);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1R
32f_C3R
32f_AC4R

引数

pSrc1, pSrc2	ソース・バッファへのポインタ
src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
src1Size, src2Size	ソース・イメージのサイズ（バイト数で指定）。
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
divisor	（整数データを処理する場合のみ）計算結果を割る整数値

説明

関数 ippiConvValid は、ippi.h ファイルの中で宣言される。この関数は、*pSrc1* と *pSrc2* が指す 2 つのソース・イメージに対して、二次元の有効有限線形たたみ込み演算を実行する。
最初のソース・イメージをマトリックス *f*（サイズは *Mf* × *Nf*）で表し、2 番目のソース・イメージをマトリックス *g*（サイズは *Mg* × *Ng*）で表すと、このたたみ込み関数で得られるデスティネーション・イメージ *h* のサイズは、$M_h \times N_h$ となる。ただし、$M_h = |M_f - M_g| + 1$、$N_h = |N_f - N_g| + 1$ である。

関数 ippiConvFull は次の式を使用して、デスティネーション・イメージの値 *h* [*i*,*j*] を計算する。

$$h[i,j] = \frac{1}{divisor} \sum_{l=0}^{N_g-1} \sum_{k=0}^{M_g-1} f[i+k, j+l] \times g[M_g - k - 1, N_g - l - 1] \quad ,$$

ただし、$0 \le i < M_h$, $0 \le j < N_h$

ここで、$M_f \ge M_g$ かつ $N_f \ge N_g$ とする。$M_f < M_g$ かつ $N_f < N_g$ の場合は、この式の添え字 *g* と *f* を入れ替える必要がある。これ以外のソース・イメージ・サイズの組み合わせの場合は、関数 ippiConvValid は何の処理も行わない。

上記の式は、ippiConvFull 関数と同じ結果になるが、ゼロ・パディングを使用せずに計算されたたたみ込み画像部分のみ出力される点が異なることに注意する。

Ipp32f 型の入力データを処理する関数も同様の計算式を使用するが、合計値のスケーリングは行わない（*divisor* = 1 となる）。

このたたみ込み関数の機能を説明するために、次のようなソース・イメージ *f*, *g*（サイズは 3 × 5）があるとする。

$$f = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad , \quad g = f$$

たたみ込みを実行して得られた画像のサイズは 1 × 1 となり、次のようなデータが含まれる。

$$h = \begin{bmatrix} 11 \end{bmatrix}$$

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc1*、*pSrc2*、または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*Src1Size* または *Src2Size* の値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*src1Step*、*src2Step*、または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsDivisorErr	*divisor* の値が 0 の場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

固定フィルタ

固定フィルタ関数は、あらかじめ定義されているたたみ込みカーネルの 1 つを使用して、ソース・イメージの線形フィルタリングを実行する。次の表に、サポートされている固定フィルタとそのカーネル・サイズの一覧を示す。

表 9-3　固定フィルタ関数のタイプ

固定フィルタ・タイプ	カーネルのサイズ
水平 Prewitt 演算子	3 × 3
垂直 Prewitt 演算子	3 × 3
水平 Scharr 演算子	3 × 3
垂直 Scharr 演算子	3 × 3
水平 Sobel 演算子	3 × 3 または 5 × 5
垂直 Sobel 演算子	3 × 3 または 5 × 5
二次導関数の水平 Sobel 演算子	3 × 3 または 5 × 5
二次導関数の垂直 Sobel 演算子	3 × 3 または 5 × 5
二次導関数のクロス Sobel 演算子	3 × 3 または 5 × 5
水平 Roberts 演算子	3 × 3
垂直 Roberts 演算子	3 × 3
ラプラシアン・ハイパス・フィルタ	3 × 3 または 5 × 5
ガウシアン・ローパス・フィルタ	3 × 3 または 5 × 5
ハイパス・フィルタ	3 × 3 または 5 × 5
ローパス・フィルタ	3 × 3 または 5 × 5
鮮明化フィルタ	3 × 3

あらかじめ定義されたカーネルを使用する固定フィルタ関数を利用すると、アプリケーション・プログラムでたたみ込みカーネルを作成する必要がないので便利である。

注： すべての固定フィルタ関数で、画像の境界ピクセルを処理するとき、有効な操作を保証するには、アプリケーションが追加のボーダ・ピクセルを正しく定義する必要がある（「境界」を参照）。

コンピュータ・ビジョン・アプリケーションで使用する固定フィルタの詳細については、第 14 章の「フィルタ」の節を参照のこと。

FilterPrewittHoriz

水平 Prewitt カーネルを使用して、画像をフィルタリングする。

```
IppStatus ippiFilterPrewittHoriz_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiFilterPrewittHoriz は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI に水平 Prewitt 演算子を適用する。これに対応するカーネルは、3 × 3 のサイズのマトリックスで、次の値が含まれる

```
 1   1   1
 0   0   0
-1  -1  -1
```

このフィルタは、画像の水平エッジを強調する効果を持つ。
ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

FilterPrewittVert

垂直 Prewitt カーネルを使用して、画像をフィルタリングする。

```
IppStatus ippiFilterPrewittVert_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiFilterPrewittVert は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI に垂直 Prewitt 演算子を適用する。これに対応するカーネルは、3 × 3 のサイズのマトリックスで、次の値が含まれる。

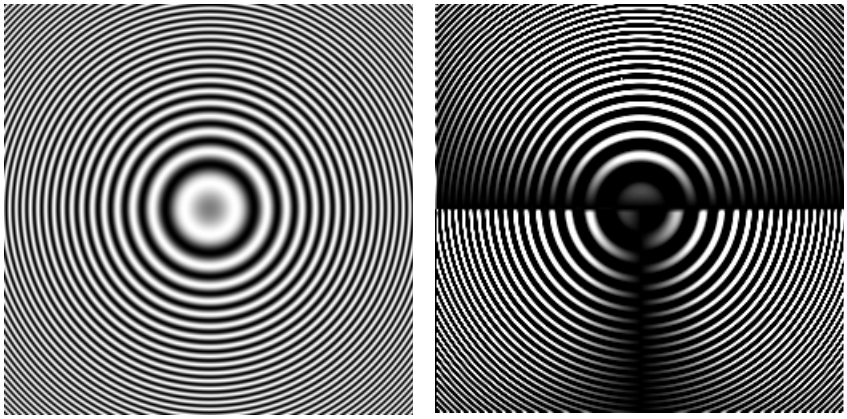
```
-1   0   1
-1   0   1
-1   0   1
```

このフィルタは、画像の垂直エッジを強調する効果を持つ。

ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

図 9-1 は、サンプル・イメージの上半分に水平 Prewitt 演算子、下半分に垂直 Prewitt 演算子を使用してフィルタリングした結果を示している。unsigned char 値に対する関数を使用したため、負の値を持つピクセルはすべて 0 になっている。

図 9-2　Prewitt 演算子を使用したイメージ・フィルタリング



サンプル・イメージ　　フィルタリングした画像

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	pSrc または pDst ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	dstRoiSize フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	srcStep または dstStep の値が 0 または負の場合のエラー状態を示す。

FilterSharrHoriz

水平 Scharr カーネルを使用して、画像をフィルタリングする。

```
IppStatus ippiFilterScharrHoriz_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep,
    IppiSize dstRoiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u16s_C1R    8s16s_C1R    32f_C1R
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiFilterScharrHoriz は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI に水平 Scharr 演算子を適用する。これに対応するカーネルは、3 × 3 のサイズのマトリックスで、次の値が含まれる。

```
 3   10   3
 0    0   0
-3  -10  -3
```

このフィルタは、画像の水平エッジを強調かつスムージングする効果を持つ。ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールド値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

FilterSharrVert

垂直 Scharr カーネルを使用して、画像をフィルタリングする。

```
IppStatus ippiFilterScharrVert_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep,
    IppiSize dstRoiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u16s_C1R    8s16s_C1R    32f_C1R
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiFilterScharrVert は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI に垂直 Scharr 演算子を適用する。これに対応するカーネルは、3 × 3 のサイズのマトリックスで、次の値が含まれる。

```
 3   0   -3
10   0  -10
 3   0   -3
```

このフィルタは、画像の垂直エッジを強調かつスムージングする効果を持つ。ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールド値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	srcStep または dstStep の値が 0 または負の場合のエラー状態を示す。

FilterSobelHoriz, FilterSobelHorizMask

水平 Sobel カーネルを使用して、画像をフィルタリングする。

```
IppStatus ippiFilterSobelHoriz_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiFilterSobelHoriz_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep,
    IppiSize dstRoiSize, IppiMaskSize mask);
```

サポートされる *mod* の値は、次のとおりである。

8u16s_C1R	8s16s_C1R

```
IppStatus ippiFilterSobelHorizMask_32f_C1R(const Ipp32f* pSrc,
    int srcStep, Ipp32f* pDst, int dstStep,
    IppiSize dstRoiSize, IppiMaskSize mask);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
mask	IppiMaskSize 型の定義済みマスク

説明

関数 ippiFilterSobelHoriz と関数 ippiFilterSobelHorizMask は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI に水平 Sobel 演算子を適用する。これに対応する適切なカーネルは、3 × 3 のサイズのマトリックス、または対応する関数の種類の *mask* パラメータに従って 3 × 3 または 5 × 5 のサイズのマトリックスである。カーネルには、次の値が含まれる。

```
 1   2   1                    1   4   6   4   1
 0   0   0        または      2   8  12   8   2
-1  -2  -1                    0   0   0   0   0
                              -2  -8 -12  -8  -4
                              -1  -4  -6  -4  -1
```

このフィルタは、画像の水平エッジを強調かつスムージングする効果を持つ。

ソース・イメージ ROI のサイズは、dstRoiSize（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

ippStsMaskSizeErr　　*mask* の値が無効な場合のエラー状態を示す。

FilterSobelVert, FilterSobelVertMask

垂直 Sobel カーネルを使用して、画像をフィルタリングする。

```
IppStatus ippiFilterSobelVert_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiFilterSobelVert_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep,
    IppiSize dstRoiSize, IppiMaskSize mask);
```

サポートされる *mod* の値は、次のとおりである。

8u16s_C1R	8s16s_C1R

```
IppStatus ippiFilterSobelMaskVert_32f_C1R(const Ipp32f* pSrc,
    int srcStep, Ipp32f* pDst, int dstStep,
    IppiSize dstRoiSize, IppiMaskSize mask);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
mask	IppiMaskSize の型の定義済みマスク

説明

関数 ippiFilterSobelVert と関数 *ippiFilterSovelVertMask* は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI に垂直 Sobel 演算子を適用する。これに対応する適切なカーネルは、3 × 3 のサイズのマトリックス、または対応する関数の種類の *mask* パラメータに従って 3 × 3 または 5 × 5 のサイズのマトリックスである。カーネルには、次の値が含まれる。

```
                                    1   4   6   4   1
   1   2   1                        2   8  12   8   2
   0   0   0        または          0   0   0   0   0
  -1  -2  -1                       -2  -8 -12  -8  -4
                                   -1  -4  -6  -4  -1
```

このフィルタは、画像の垂直エッジを強調かつスムージングする効果を持つ。
ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*mask* の値が無効な場合のエラー状態を示す。

FilterSobelHorizSecond

二次導関数の水平 Sobel 演算子を使用して、
画像をフィルタリングする。

```
IppStatus ippiFilterSobelHorizSecond_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

サポートされる *mod* の値は、次のとおりである。

8u16s_C1R　　8s16s_C1R　　32f_C1R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
mask	*IppiMaskSize* 型の定義済みマスク

説明

関数 ippiFilterSobelHorizSecond は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI に二次導関数の水平 Sobel 演算子を適用する。これに対応するカーネルは、3 × 3 または 5 × 5 のサイズのマトリックスで、次の値が含まれる。

```
                                    1   4   6   4   1
 1   2   1                          0   0   0   0   0
-2  -4  -2          または          -2  -8 -12  -8  -2
 1   2   1                          0   0   0   0   0
                                    1   4   6   4   1
```

このフィルタは、画像の水平エッジを強調かつスムージングする効果を持つ。
ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールド値が 0 または負の場合のエラー状態を示す。

ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*mask* の値が無効な場合のエラー状態を示す。

FilterSobelVertSecond

二次導関数の垂直 Sobel 演算子を使用して、画像をフィルタリングする。

```
IppStatus ippiFilterSobelVertSecond_<mod>(const Ipp<srcDatatype>* pSrc,
      int srcStep, Ipp<dstDatatype>* pDst, int dstStep,
IppiSize dstRoiSize,
      IppiMaskSize mask);
```

サポートされる *mod* の値は、次のとおりである。

```
8u16s_C1R     8s16s_C1R     32f_C1R
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
mask	IppiMaskSize 型の定義済みマスク

説明

関数 ippiFilterSobelVertSecond は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI に二次導関数の垂直 Sobel 演算子を適用する。これに対応するカーネルは、3 × 3 または 5 × 5 のサイズのマトリックスで、次の値が含まれる。

```
                                    1   0   -2   0   1
 1  -2   1                         4   0   -8   0   4
 2  -4   2          または         6   0  -12   0   6
 1  -2   1                         4   0   -8   0   4
                                    1   0   -2   0   1
```

このフィルタは、画像の垂直エッジを強調かつスムージングする効果を持つ。ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールド値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*mask* の値が無効な場合のエラー状態を示す。

FilterSobelCross

二次導関数のクロス Sobel 演算子を使用して、画像をフィルタリングする。

```
IppStatus ippiFilterSobelCross_<mod>(const Ipp<srcDatatype>* pSrc,
      int srcStep, Ipp<dstDatatype>* pDst, int dstStep,
IppiSize dstRoiSize,
      IppiMaskSize mask);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u16s_C1R     8s16s_C1R     32f_C1R
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
mask	IppiMaskSize 型の定義済みマスク

説明

関数 ippiFilterSobelCross は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI に二次導関数のクロス Sobel 演算子を適用する。これに対応するカーネルは、3 × 3 または 5 × 5 のサイズのマトリックスで、次の値が含まれる。

```
                                  -1  -2   0   2   1
 -1   0   1                       -2  -4   0   4   2
  0   0   0          または        0   0   0   0   0
  1   0  -1                        2   4   0  -4  -2
                                   1   2   0  -2  -1
```

このフィルタは、画像の水平エッジを強調かつスムージングする効果を持つ。
ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールド値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*mask* の値が無効な場合のエラー状態を示す。

FilterRobertsDown

水平 Roberts カーネルを使用して、画像をフィルタリングする。

```
IppStatus ippiFilterRobertsDown_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiFilterRobertsDown は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI に水平 Roberts 演算子を適用する。これに対応するカーネルは、2 × 2 のサイズのマトリックスで、次の値が含まれる。

```
0   0   0
0   1   0
0   0  -1
```

このフィルタは、ピクセル値の水平方向の傾きの近似値をもたらす。
ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

FilterRobertsUp

垂直 Roberts カーネルを使用して、画像をフィルタリングする。

```
IppStatus ippiFilterRobertsUp_<mod>(const Ipp<datatype>* pSrc, int
      srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiFilterRobertsUp は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI に垂直 Roberts 演算子を適用する。これに対応するカーネルは、3 × 3 のサイズのマトリックスで、次の値が含まれる。

```
 0   0   0
 0   1   0
-1   0   0
```

このフィルタは、ピクセル値の垂直方向の傾きの近似値をもたらす。
ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

FilterLaplace

ラプラシアン・カーネルを使用して、画像をフィルタリングする。

```
IppStatus ippiFilterLaplace_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize
    dstRoiSize, IppiMaskSize mask);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R	8u16s_C1R	8s16s_C1R
8u_C3R	16s_C3R	32f_C3R		
8u_C4R	16s_C4R	32f_C4R		
8u_AC4R	16s_AC4R	32f_AC4R		

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
mask	IppiMaskSize 型の定義済みマスク

説明

関数 ippiFilterLaplace は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI にハイパス・ラプラシアン・フィルタを適用する。これに対応するカーネルは、3 × 3 または 5 × 5 のサイズのマトリックスで、次の値が含まれる。

```
                                  -1  -3  -4  -3  -1
   -1  -1  1                      -3   0   6   0  -3
   -1   8  1          または      -4   6  20   6  -4
   -1  -1  1                      -3   0   6   0  -3
                                  -1  -3  -4  -3  -1
```

このフィルタは、画像内のゼロ交差位置を見つけ出すのに役立つ。
ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*mask* の値が無効な場合のエラー状態を示す。

FilterGauss

ガウシアン・カーネルを使用して、画像を
フィルタリングする。

```
IppStatus ippiFilterGauss_<mod>(const Ipp<datatype>* pSrc, int
      srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
       IppiMaskSize mask);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
mask	IppiMaskSize 型の定義済みマスク

説明

関数 ippiFilterGauss は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI にローパス・ガウシアン・フィルタを適用する。これに対応するカーネルは、3 × 3 または 5 × 5 のサイズのマトリックスである。

3 × 3 のフィルタは、つぎのカーネルを使用する。

```
1/16    2/16    1/16
2/16    4/16    2/16
1/16    2/16    1/16
```

このフィルタ係数は、標準偏差が 0.85 の二次元ガウス分布に対応している。

5 × 5 のフィルタは、次のカーネルを使用する。

```
 2/571   7/571   12/571   7/571   2/571
 7/571  31/571   52/571  31/571   7/571
12/571  52/571  127/571  52/571  12/571
 7/571  31/571   52/571  31/571   7/571
 2/571   7/571   12/571   7/571   2/571
```

このフィルタ係数は、標準偏差が 1.0 の二次元ガウス分布に対応している。
ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*mask* の値が無効な場合のエラー状態を示す。

FilterHipass

ハイパス・フィルタを使用して、画像をフィルタリングする。

```
IppStatus ippiFilterHipass_<mod>(const Ipp<datatype>* pSrc, int
      srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
       IppiMaskSize mask);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R      32f_C1R
8u_C3R      16s_C3R      32f_C3R
8u_C4R      16s_C4R      32f_C4R
8u_AC4R     16s_AC4R     32f_AC4R
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
mask	IppiMaskSize 型の定義済みマスク

説明

関数 ippiFilterHipass は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI にハイパス・フィルタを適用する。これに対応するカーネルは、3 × 3 または 5 × 5 のサイズのマトリックスであり、次の値が含まれる。

```
                                  -1  -1  -1  -1  -1
   -1  -1  -1                     -1  -1  -1  -1  -1
   -1   8  -1          または     -1  -1  24  -1  -1
   -1  -1  -1                     -1  -1  -1  -1  -1
                                  -1  -1  -1  -1  -1
```

このフィルタは、低周波成分を減衰させることにより、画像を鮮明化する。
ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	*mask* の値が無効な場合のエラー状態を示す。

FilterLowpass

ローパス・フィルタを使用して、画像を
フィルタリングする。

```
IppStatus ippiFilterLowpass_<mod>(const Ipp<datatype>* pSrc, int
      srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
       IppiMaskSize mask);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
mask	IppiMaskSize 型の定義済みマスク

説明

関数 ippiFilterLowpass は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI にローパス・フィルタを適用する。これに対応するカーネルは、3 × 3 または 5 × 5 のサイズのマトリックスである。

3 × 3 のフィルタは、次のカーネルを使用する。

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

5 × 5 のフィルタは、つぎのカーネルを使用する。ここで、

```
1/25   1/25   1/25   1/25   1/25
1/25   1/25   1/25   1/25   1/25
1/25   1/25   1/25   1/25   1/25
1/25   1/25   1/25   1/25   1/25
1/25   1/25   1/25   1/25   1/25
```

このフィルタは、ピクセル値を隣接領域に平均化することにより、画像をぼかす。ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMaskSizeErr	mask の値が無効な場合のエラー状態を示す。

FilterSharpen

鮮明化フィルタを使用して、画像をフィルタリングする。

```
IppStatus ippiFilterSharpen_<mod>(const Ipp<datatype>* pSrc, int
      srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R      32f_C1R
8u_C3R      16s_C3R      32f_C3R
8u_C4R      16s_C4R      32f_C4R
8u_AC4R     16s_AC4R     32f_AC4R
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiFilterSharpen は、ippi.h ファイルの中で宣言される。この関数は、画像 ROI にローパス・フィルタを適用する。これに対応するカーネルは、3 × 3 のサイズのマトリックス A/8 である。ここで、

```
-1/8  -1/8  -1/8
-1/8  16/8  -1/8
-1/8  -1/8  -1/8
```

このフィルタは、高周波成分を増幅することにより、画像を鮮明化する。
ソース・イメージ ROI のサイズは、*dstRoiSize*（デスティネーション・イメージ ROI のサイズ）と同じである。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

画像の線形変換 10

本章では、イメージ・バッファでの線形変換を実行する インテル® IPP 関数について説明する。

線形変換には、高速フーリエ変換（FFT）、離散フーリエ変換（DFT）、離散コサイン変換（DCT）がある。

表 10-1 に、インテル IPP 線形変換関数の一覧を示す。

表 10-1　画像の線形変換関数

関数の基本名	操作
フーリエ変換	
FFTInitAlloc	メモリを割り当て、画像 FFT 関数が必要とするコンテキスト・データを入れる。
FFTFree	FFT コンテキスト構造体が使用していたメモリを解放する。
FFTGetBufSize	FFT 関数が使用する外部ワーク・バッファのサイズを求める。
FFTFwd	画像に順方向高速フーリエ変換を適用する。
FFTInv	複素数ソース・データに逆方向高速フーリエ変換を適用し、その結果をデスティネーション・イメージに格納する。
MulPack	パックド形式のデータを持つ 2 つのソース・イメージを掛け合わせ、その結果をパックド形式でデスティネーション・イメージに格納する。
MulPackConj	パックド形式のデータを持つ 2 つのソース・イメージを掛け合わせ、その結果をデスティネーション・イメージに格納する。この画像は、`ippiMulPack` 関数で得られる画像に対して複素共役になる。
DFTInitAlloc	メモリを割り当て、画像 DFT 関数が必要とするコンテキスト・データを入れる。
DFTFree	DFT コンテキスト構造体が使用していたメモリを解放する。

続く

表 10-1　画像の線形変換関数 （続き）

関数の基本名	操作
DFTGetBufSize	DFT 関数が使用する外部ワーク・バッファのサイズを求める。
DFTFwd	画像に順方向離散フーリエ変換を適用する。
DFTInv	複素数ソース・データに逆方向離散フーリエ変換を適用し、その結果をデスティネーション・イメージに格納する。
PackToCplxExtend	パック ド形式の画像を複素データ画像に変換する。
離散コサイン変換	
DCTFwdInitAlloc	メモリを割り当て、順方向 DCT 関数が必要とするコンテキスト・データを入れる。
DCTInvInitAlloc	メモリを割り当て、逆方向 DCT 関数が必要とするコンテキスト・データを入れる。
DCTFwdFree	順方向 DCT コンテキスト構造体が使用していたメモリを解放する。
DCTInvFree	逆方向 DCT コンテキスト構造体が使用していたメモリを解放する。
DCTFwdGetBufSize	順方向 DCT 関数が使用する外部ワーク・バッファのサイズを求める。
DCTInvGetBufSize	逆方向 DCT 関数が使用する外部ワーク・バッファのサイズを求める。
DCTFwd	画像に順方向 DCT を実行する。
DCTInv	画像に逆方向 DCT を実行する。
DCT8x8Fwd	8 × 8 のサイズのバッファに順方向 DCT を実行する。
DCT8x8Inv	8 × 8 のサイズのバッファに順方向 DCT を実行する。
DCT8x8FwdLS	レベル・シフトを適用した 8 × 8 のサイズの 2D バッファに、順方向 DCT を実行する。
DCT8x8InvLSClip	8 × 8 のサイズのバッファに逆方向 DCT を実行し、レベル・シフトを適用する。

線形変換関数では、パフォーマンスを上げるために、計算に必要な補助データ （例えば、FFT 関数の回転因子テーブル）をあらかじめ計算しておき、実行時にそれを利用する。補助データは、変換の種類に応じた初期化関数で計算し、それぞれ専用のコンテキスト構造体を使用して線形変換関数に引き渡す。

インテル IPP のほとんどの線形変換関数は、異なる計算アルゴリズムを実行するためのコード分岐を持つ。線形変換関数が使用するコードを選択するには、*hint* 引数に 表 10-2 に示す次のいずれかの値を設定して指定する。

表 10-2　リニア変換関数の hint 引数

値	説明
ippAlgHintNone	計算アルゴリズムは関数の内部ロジックで選択される。
ippAlgHintFast	高速アルゴリズムを使用する。出力結果の精度は低下する。
ippAlgHintAccurate	高精度アルゴリズムを使用する。関数の実行時間は長くなる。

インテル IPP 線形変換関数は、データや中間結果を格納するのに外部ワーク・バッファを使用できる。外部ワーク・バッファを使用すると、内部メモリ・バッファを割り当てる必要がなくなるので、関数のパフォーマンスを向上できる。必要なワーク・バッファのサイズを調べるには、線形変換の種類に応じたサポート関数を使用する。外部バッファが指定されていない場合は、線形変換関数が内部でメモリ割り当てを行う。

8 × 8 のサイズの DCT を除くすべてのインテル IPP 線形変換関数は、浮動小数点データの画像のみを対象とする。

フーリエ変換

FFT と DFT を実行するインテル IPP 関数は、実数と複素数のどちらの画像も処理できる。実数データを処理する関数タイプは関数名に R のサフィックスが付き、複素数データを処理する関数タイプは C のサフィックスが付く（第 2 章の「関数の命名」を参照）。

フーリエ変換結果を正規化するには、コンテキスト初期化関数の *flag* 引数に適切な値を指定する。このパラメータで、順変換と逆変換で使用される正規化因子のペアが、次の表のように設定される。

表 10-3　フーリエ変換結果の正規化因子

flag 引数の値	正規化因子	
	順変換	逆変換
IPP_FFT_DIV_FWD_BY_N	1/MN	1
IPP_FFT_DIV_INV_BY_N	1	1/MN
IPP_FFT_DIV_BY_SQRTN	1/sqrt(MN)	1/sqrt(MN)
IPP_FFT_NODIV_BY_ANY	1	1

この表の N と M は、フーリエ変換の x および y 方向の長さ （すなわち、変換される 2D 配列の列数と行数）である。
FFT の場合、これらの 2 つの長さは 2 の整数べき乗、すなわち $N=2^{orderX}$、$M=2^{orderY}$ でなければならない。この指数部分を FFT の次数と呼ぶ。
DFT の場合は、N と M は負でない任意の整数値を取ることができる。

実数 - 複素数パック ド形式（RCPack2D 形式）

実数の二次元画像データを順方向にフーリエ変換して得られた複素数マトリックスは、共役対称性を持つ。インテル IPP 関数は、このタイプのデータを格納するのにパック ド形式 RCPack2D を使用する。それに対して、実数を対象とする逆フーリエ変換関数は、対称性を持つパック ド形式の共役複素数データから、元の実数データを復元する。
RCPack2D 形式は、変換後の複素数データの共役対称性を利用して、求められたフーリエ係数の半分だけを格納する。すなわち、N × M の変換の場合、FFT および DFT の各関数は、複素フーリエ係数 A(i,j)、ただし i = 0,...,M-1; j = 0,... N/2 の実数部と虚数部を、次数 （N, M）の実数配列に格納する。RCPack2D 格納形式では、次の表に示すように、M が奇数か偶数かによって、データの配置が少し異なる。

表 10-4　行数が奇数の場合の RCPack2D 格納形式

Re A(0,0)	Re A(0,1)	Im A(0,1)	...	Re A(0,(N-1)/2)	Im A(0,(N-1)/2)	Re A(0,N/2)
Re A(1,0)	Re A(1,1)	Im A(1,1)	...	Re A(1,(N-1)/2)	Im A(1,(N-1)/2)	Re A(1,N/2)
Im A(1,0)	Re A(2,1)	Im A(2,1)	...	Re A(2,(N-1)/2)	Im A(2,(N-1)/2)	Im A(1,N/2)
...	...	...	...	...	...	...
Re A(M/2,0)	Re A(M-2,1)	Im A(M-2,1)	...	Re A(M-2,(N-1)/2)	Im A(M-2,(N-1)/2)	Re A(M/2,N/2)
Im A(M/2,0)	Re A(M-1,1)	Im A(M-1,1)	...	Re A(M-1,(N-1)/2)	Im A(M-1,(N-1)/2)	Im A(M/2,N/2)

表 10-5　行数が偶数の場合の RCPack2D 格納形式

Re A(0,0)	Re A(0,1)	Im A(0,1)	...	Re A(0,(N-1)/2)	Im A(0,(N-1)/2)	Re A(0,N/2)
Re A(1,0)	Re A(1,1)	Im A(1,1)	...	Re A(1,(N-1)/2)	Im A(1,(N-1)/2)	Re A(1,N/2)
Im A(1,0)	Re A(2,1)	Im A(2,1)	...	Re A(2,(N-1)/2)	Im A(2,(N-1)/2)	Im A(1,N/2)
...	...	...	...	...	...	...
Re A(M/2-1,0)	Re A(M-3,1)	Im A(M-3,1)	...	Re A(M-3,(N-1)/2)	Im A(M-3,(N-1)/2)	Re A(M/2-1,N/2)
Im A(M/2-1,0)	Re A(M-2,1)	Im A(M-2,1)	...	Re A(M-2,(N-1)/2)	Im A(M-2,(N-1)/2)	Im A(M/2-1,N/2)
Re A(M/2,0)	Re A(M-1,1)	Im A(M-1,1)	...	Re A(M-1,(N-1)/2)	Im A(M-1,(N-1)/2)	Re A(M/2,N/2)

上の 2 つの表で、網掛けした最後の列は、N が偶数のときのみ使用される。

残りのフーリエ係数は、共役対称性から、次の関係式で求められる。

```
A(i,j) = conj(A(M-i,N-j))     i = 1,..., M-1;   j = 1,..., N-1
A(0,j) = conj(A(0,N-j))       j = 1,..., N-1
A(i,0) = conj(A(M-i,0))       i = 1,..., M-1
```

FFTInitAlloc

メモリを割り当て、画像 FFT 関数が必要とするコンテキスト・データを入れる。

```
IppStatus ippiFFTInitAlloc_R_32s (IppiFFTSpec_R_32s** pFFTSpec,
     int orderX, int orderY, int flag, IppHintAlgorithm hint);
IppStatus ippiFFTInitAlloc_R_32f (IppiFFTSpec_R_32f** pFFTSpec,
     int orderX, int orderY, int flag, IppHintAlgorithm hint);
IppStatus ippiFFTInitAlloc_C_32fc (IppiFFTSpec_C_32fc** pFFTSpec,
     int orderX, int orderY, int flag, IppHintAlgorithm hint);
```

引数

pFFTSpec	初期化する FFT コンテキスト構造体へのポインタのポインタ
orderX, orderY	x 方向と y 方向の FFT の次数
flag	計算結果の正規化オプションを選択するフラグ
hint	変換関数の計算アルゴリズムを選択するオプション

説明

関数 ippiFFTInitAlloc は、ippi.h ファイルの中で宣言される。この関数は、メモリを割り当て、二次元画像データの順方向および逆方向 FFT で必要なコンテキスト構造体 *pFFTSpec* を初期化する。
ippiFFTFwd と ippiFFTInv の 2 種類の関数は、初期化された *pFFTSpec* 構造体を引数として呼び出され、次のような高速フーリエ変換を行う。

- x 方向の長さは $N=2^{orderX}$、y 方向の長さは $M=2^{orderY}$
- 計算結果の正規化モードは、*flag* で指定 （表 10-3 を参照）
- 計算アルゴリズムは *hint* で指定 （詳細は、10-2 を参照）

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pFFTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsFftOrderErr	FFT の次数値が無効である場合のエラー状態を示す。
ippStsFFTFlagErr	*flag* の値が無効である場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

FFTFree

FFT コンテキスト構造体が使用していたメモリを解放する。

```
IppStatus ippiFFTFree_R_32s (IppiFFTSpec_R_32s* pFFTSpec);
IppStatus ippiFFTFree_R_32f (IppiFFTSpec_R_32f* pFFTSpec);
IppStatus ippiFFTFree_C_32fc (IppiFFTSpec_C_32fc* pFFTSpec);
```

引数

pFFTSpec	解放する FFT コンテキスト構造体へのポインタ

説明

関数 ippiFFTFree は、ippi.h ファイルの中で宣言される。この関数は、前に初期化した FFT コンテキスト構造体 *pFFTSpec* が使用していたメモリを解放する。アプリケーション・プログラムは、高速フーリエ変換の計算が終了してから、この関数を呼び出す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pFFTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	*pFFTSpec* 構造体への無効なポインタが渡された場合のエラー状態を示す。

FFTGetBufSize

FFT 関数が使用する外部ワーク・バッファのサイズを求める。

```
IppStatus ippiFFTGetBufSize_R_32s (IppiFFTSpec_R_32s* pFFTSpec,
      int* size);
IppStatus ippiFFTGetBufSize_R_32f (IppiFFTSpec_R_32f* pFFTSpec,
      int* size);
IppStatus ippiFFTGetBufSize_C_32fc (IppiFFTSpec_C_32fc* pFFTSpec,
      int* size);
```

引数

pFFTSpec	前に初期化した FFT コンテキスト構造体へのポインタ
size	FFT 関数が使用する外部ワーク・バッファのサイズへのポインタ

説明

関数 ippiFFTGetBufSize は、ippi.h ファイルの中で宣言される。この関数は、コンテキスト *pFFTSpec* を使用する FFT 関数が、計算中にワークスペースとして使用する外部メモリ・バッファのサイズ （バイト数）を求める。
外部バッファを使用する場合は、*pFFTSpec* を初期化してから、この関数を呼び出す。
求めた *size* 値をバイト数として、外部バッファの割り当てを行うが、それには ippsMalloc のような関数を使用する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pFFTSpec* または *size* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	*pFFTSpec* 構造体への無効なポインタが渡された場合のエラー状態を示す。

FFTFwd

画像に順方向高速フーリエ変換を適用する。

事例 1：整数データの画像の操作

```
IppStatus ippiFFTFwd_RToPack_<mod> (const Ipp8u* pSrc, int srcStep,
      Ipp32s* pDst, int dstStep, const IppiFFTSpec_R_32s* pFFTSpec,
      int scaleFactor, Ipp8u* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
8u32s_C1RSfs
8u32s_C3RSfs
8u32s_C4RSfs
8u32s_AC4RSfs
```

事例 2：浮動小数点データの画像の操作

```
IppStatus ippiFFTFwd_RToPack_<mod> (const Ipp32f* pSrc, int srcStep,
      Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,
      Ipp8u* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

事例 3：複素数データの画像の操作

```
IppStatus ippiFFTFwd_CToC_32fc_C1R (const Ipp32fc* pSrc, int srcStep,
      Ipp32fc* pDst, int dstStep, const IppiFFTSpec_C_32fc* pFFTSpec,
      Ipp8u* pBuffer);
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・バッファ内のステップ（バイト単位）
pFFTSpec	前に初期化した FFT コンテキスト構造体へのポインタ

scaleFactor	整数結果のスケーリングに使用される係数
pBuffer	FFT 関数が使用する外部バッファへのポインタ

説明

関数 ippiFFTFwd は、ippi.h ファイルの中で宣言される。この関数は、入力画像 *pSrc* の各チャネルに対して順方向 FFT を実行し、そのフーリエ係数を出力バッファ *pDst* の対応するチャネルに書き込む。
実数データの画像を処理する関数 ippiFFTFwd_RToPack は、対称性を利用して出力データを RCPack2D 形式 で書き込む。この関数は、1、3、4 チャネルの画像を処理できる。AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

複素数データの画像を処理する関数 ippiFFTFwd_CToC は、周波数領域に関する対称性が見られないため、変換結果をパックしない。複素数データの画像のメモリ・レイアウトは、各ピクセル値が実数部と虚数部の 2 つで構成される点を除けば、実数の画像と同じである。

順方向の FFT 関数は、事前に初期化しておいた *pFFTSpec* コンテキスト構造体を使用して、計算モードを設定し、サポート・データを取得する。
また、アプリケーションから、外部ワーク・バッファ *pBuffer* へのポインタを渡すこともできる。その場合は、FFT 関数を呼び出す前に ippiFFTGetBufSize 関数を呼び出して、必要なバッファ・サイズを求める必要がある。NULL ポインタが渡された場合は、*ippiFFTFwd* 関数が内部でメモリを割り当てる。

下のコード例は、順方向 FFT 関数を使用した実数データの処理を示している。

例 10-1　実数画像の高速フーリエ変換

```
IppStatus fft( void ) {
   Ipp32f src[64] = {0}, dst[64];
   IppiFFTSpec_R_32f *spec;
   IppStatus status;
   src[0] = -3; src[9] = 1;
   ippiFFTInitAlloc_R_32f( &spec, 3, 3, IPP_FFT_DIV_INV_BY_N,
ippAlgHintAccurate );
   status = ippiFFTFwd_RToPack_32f_C1R( src, 8*4, dst, 8*4,
spec, 0 );
   ippiFFTFree_R_32f( spec );
   return status;
}
```

ippiFFTFwd_RToPack 関数が終了すると、デスティネーション・バッファに次のデータが RCPack2D 形式で入っている。

```
-2.00  -2.29  -0.71  -3.00  -1.00  -3.71  -0.71  -4.00
-2.29  -3.00  -1.00  -3.71  -0.71  -4.00  +0.00  -3.71
-0.71  -3.71  -0.71  -4.00  +0.00  -3.71  +0.71  +0.71
-3.00  -4.00  +0.00  -3.71  +0.71  -3.00  +1.00  -3.00
-1.00  -3.71  +0.71  -3.00  +1.00  -2.29  +0.71  +1.00
-3.71  -3.00  +1.00  -2.29  +0.71  -2.00  +0.00  -2.29
-0.71  -2.29  +0.71  -2.00  +0.00  -2.29  -0.71  +0.71
-4.00  -2.00  +0.00  -2.29  -0.71  -3.00  -1.00  -2.00
```

次のコード例は、複素数データの FFT 処理を示している。入力データは上の例と同じだが、複素数領域で指定していることに注意する。

例 10-2　複素数画像の高速フーリエ変換

```
IppStatus fft_cplx( void ) {
   Ipp32fc src[64] = {0}, dst[64], m3 = {-3,0}, one = {1,0};
   IppiFFTSpec_C_32fc *spec;
   IppStatus status;
   src[0] = m3; src[9] = one;
   ippiFFTInitAlloc_C_32fc( &spec, 3, 3, IPP_FFT_DIV_INV_BY_N,
   ippAlgHintAccurate);
   status = ippiFFTFwd_CToC_32fc_C1R( src, 8*4*2, dst, 8*4*2, spec, 0 );
    ippiFFTFree_C_32fc( spec );
   return status;
}
```

ippiFFTFwd_CToC 関数が終了すると、デスティネーション・バッファに次のデータが入っている （複素数の実数部と虚数部をコンマで区切って示す)。

```
-2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7
-2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0,-0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0, 0.0
-3.0,-1.0 -3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0,-0.0 -2.3,-0.7
-3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0,-0.0 -2.3,-0.7 -3.0,-1.0
-4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7
-3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0,-0.0
-3.0, 1.0 -2.3, 0.7 -2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0,-0.0 -3.7, 0.7
-2.3, 0.7 -2.0,-0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pFFTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

ippStsContextMatchErr　*pFFTSpec* 構造体への無効なポインタが渡された場合のエラー状態を示す。

ippStsMemAllocErr　メモリ割り当てに失敗した場合のエラー状態を示す。

FFTInv

複素数ソース・データに逆方向 FFT を適用し、その結果をデスティネーション・イメージに格納する。

事例 1：整数データの画像の操作

```
IppStatus ippiFFTInv_PackToR_<mod> (const Ipp32s* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, const IppiFFTSpec_R_32s* pFFTSpec,
      int scaleFactor, Ipp8u* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
32s8u_C1RSfs
32s8u_C3RSfs
32s8u_C4RSfs
32s8u_AC4RSfs
```

事例 2：浮動小数点データの画像の操作

```
IppStatus ippiFFTInv_PackToR_<mod> (const Ipp32f* pSrc, int srcStep,
      Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,
      Ipp8u* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

事例 3：複素数データの画像の操作

```
IppStatus ippiFFTInv_CToC_32fc_C1R (const Ipp32fc* pSrc, int srcStep,
      Ipp32fc* pDst, int dstStep, const IppiFFTSpec_C_32fc* pFFTSpec,
      Ipp8u* pBuffer);
```

引数

pSrc　ソース・バッファへのポインタ

srcStep	ソース・バッファ内のステップ（バイト単位）
pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pFFTSpec	前に初期化した FFT コンテキスト構造体へのポインタ
scaleFactor	整数結果のスケーリングに使用される係数
pBuffer	FFT 関数が使用する外部バッファへのポインタ

説明

関数 ippiFFTInv は、ippi.h ファイルの中で宣言される。この関数は、入力バッファ *pSrc* の各チャネルに対して逆方向 FFT を実行し、復元した画像データを、出力イメージ・バッファ *pDst* の対応するチャネルに書き込む。
ippiFFTInv_PackToR で始まる関数を使用する場合は、入力バッファにデータが RCPack2D 形式で入っている必要がある。

逆方向 FFT 関数は、事前に初期化しておいた *pFFTSpec* コンテキスト構造体を使用して、計算モードを設定し、サポート・データを取得する。
また、アプリケーションから、外部ワーク・バッファ *pBuffer* へのポインタを渡すこともできる。その場合は、FFT 関数を呼び出す前に ippiFFTGetBufSize 関数を呼び出して、必要なバッファ・サイズを求める必要がある。NULL ポインタが渡された場合は、ippiFFTInv 関数が内部でメモリを割り当てる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pFFTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsContextMatchErr	*pFFTSpec* 構造体への無効なポインタが渡された場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

MulPack

パックド形式のデータを持つ２つのソース・イメージを掛け合わせ、その結果をパックド形式でデスティネーション・イメージに格納する。

事例１：整数データに対する非インプレース操作

```
IppStatus ippiMulPack_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
      const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
      int dstStep, IppiSize roiSize, int sFactor);
```

サポートされる *mod* の値は、次のとおりである。

16s_C1RSfs	32s_C1RSfs
16s_C3RSfs	32s_C3RSfs
16s_C4RSfs	32s_C4RSfs
16s_AC4RSfs	32s_AC4RSfs

事例２：浮動小数点データに対する非インプレース操作

```
IppStatus ippiMulPack_<mod>(const Ipp32f* pSrc1, int src1Step,
      const Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

事例３：整数データに対するインプレース操作

```
IppStatus ippiMulPack_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pSrcDst, int dstSrcStep, IppiSize roiSize,
      int sFactor);
```

サポートされる *mod* の値は、次のとおりである。

16s_C1IRSfs	32s_C1IRSfs
16s_C3IRSfs	32s_C3IRSfs
16s_C4IRSfs	32s_C4IRSfs
16s_AC4IRSfs	32s_AC4IRSfs

事例 4：浮動小数点データに対するインプレース操作

```
IppStatus ippiMulPack_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f*
      pSrcDst, int dstSrcStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

引数

pSrc, pSrc1, pSrc2	ソース・バッファへのポインタ
srcStep, src1Step, src2Step	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ
dstSrcStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
sFactor	整数結果のスケーリングに使用される係数

説明

関数 ippiMulPack は、ippi.h ファイルの中で宣言される。この関数は、RCPack 形式で表される 2 つのソース・イメージ、*A* および *B* で、対応するピクセル値を掛け合わせ、その結果をパックド形式でデスティネーション・バッファ *C* に書き込む。乗算は、次の公式に従って実行される。

```
ReC = ReA*ReB - ImA*ImB;
ImC = ImA*ReB + ImB*ReA
```

整数データの演算の場合、得られた値は *sFactor* によってスケーリングされる。AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

この関数は、FFT 変換を含む画像のフィルタリング処理で使用できる。次のコード例は、画像の水平方向のエッジを検出する方法を示している。まず、ソース・イメージとフィルタの両方に順方向 FFT 関数を適用して、周波数領域に変換し、結果をパックド形式で出力する。

次に、関数 ippiMulPack を使用して、この 2 つのパックド・データをポイントごとに掛け合わせて、実際のフィルタリングを実行する。最後に、フィルタリング済みデータに逆方向 FFT 関数を適用して、時間領域に戻す。

例 10-3 ippiMulPack 関数を使用した画像のフィルタリング

```
IppStatus mulpack( void ) {
   Ipp32f tsrc[64], tflt[64], tdst[64];
   Ipp32f fsrc[64], fflt[64], fdst[64];
   IppiFFTSpec_R_32f *spec;
   const IppiSize roiSize8x8 = { 8, 8 }, roiSize3x3 = { 3, 3};
   const Ipp32f filter[3*3] = {-1,-1,-1, 0,0,0, 1,1,1};
   ippiSet_32f_C1R( 0, tsrc, 8*4, roiSize8x8 );
   ippiAddC_32f_C1IR( 1, tsrc+8+1, 8*4, roiSize3x3 );
   ippiSet_32f_C1R( 0, tflt, 8*4, roiSize8x8 );
   ippiCopy_32f_C1R( filter, 3*4, tflt, 8*4,  roiSize3x3);
   ippiFFTInitAlloc_R_32f( &spec, 3, 3,  IPP_FFT_DIV_INV_BY_N,
ippAlgHintAccurate );
   ippiFFTFwd_RToPack_32f_C1R( tsrc, 8*4, fsrc, 8*4, spec, 0 );
   ippiFFTFwd_RToPack_32f_C1R( tflt, 8*4, fflt, 8*4, spec, 0 );
   ippiMulPack_32f_C1R( fsrc, 8*4, fflt, 8*4, fdst, 8*4, roiSize8x8);
   ippiFFTInv_PackToR_32f_C1R( fdst, 8*4, tdst, 8*4, spec, 0 );
   ippiFFTFree_R_32f( spec );
   return 0;
}
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。

ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負である場合のエラーを示す。

MulPackConj

ソース・イメージにパックド形式のデータを持つ共役複素画像を掛けて、その結果をパックド形式でデスティネーション・バッファに格納する。

```
IppStatus ippiMulPackConj_<mod>(const Ipp32f* pSrc, int srcStep,
      Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

引数

pSrc	第 1 のソース・バッファへのポインタ
srcStep	第 1 のソース・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	第 2 のソース・バッファとデスティネーション・バッファへのポインタ
srcDstStep	第 2 のソース・イメージ・バッファとデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiMulPackConj は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ *A* のピクセル値に、RCPack 形式で表現される共役複素画像 *A** の対応するピクセル値を掛ける。この操作の結果は、パックド形式でデスティネーション・バッファに格納される。

この関数は、インプレース操作だけを実行する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

DFTInitAlloc

メモリを割り当て、画像 DFT 関数が必要とするコンテキスト・データを入れる。

```
IppStatus ippiDFTInitAlloc_R_32s (IppiDFTSpec_R_32s** pDFTSpec,
     IppiSize roiSize, int flag, IppHintAlgorithm hint);
IppStatus ippiDFTInitAlloc_R_32f (IppiDFTSpec_R_32f** pDFTSpec,
     IppiSize roiSize, int flag, IppHintAlgorithm hint);
IppStatus ippiDFTInitAlloc_C_32fc (IppiDFTSpec_C_32fc** pDFTSpec,
     IppiSize roiSize, int flag, IppHintAlgorithm hint);
```

引数

pDFTSpec	初期化する DFT コンテキスト構造体へのポインタのポインタ
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
flag	計算結果の正規化オプションを選択するフラグ
hint	変換関数の計算アルゴリズムを選択するオプション

説明

関数 ippiDFTInitAlloc は、ippi.h ファイルの中で宣言される。この関数は、メモリを割り当て、二次元画像データの順方向および逆方向 DFT で必要なコンテキスト構造体 *pDFTSpec* を初期化する。ippiDFTFwd と ippiDFTInv の 2 種類の関数は、初期化された *pDFTSpec* 構造体を引数として呼び出され、サイズが *roiSize* の ROI 内のポイントに対して離散フーリエ変換を実行する。結果の正規化モードは *flag*（表 10-3 を参照）で指定し、計算アルゴリズムは *hint*（表 10-2 を参照）で指定する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pDFTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsFFTFlagErr	*flag* の値が無効である場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

DFTFree

DFT コンテキスト構造体が使用していたメモリを解放する。

```
IppStatus ippiDFTFree_R_32s (IppiDFTSpec_R_32s* pDFTSpec);
IppStatus ippiDFTFree_R_32f (IppiDFTSpec_R_32f* pDFTSpec);
IppStatus ippiDFTFree_C_32fc (IppiDFTSpec_C_32fc* pDFTSpec);
```

引数

pDFTSpec	解放する DFT コンテキスト構造体へのポインタ

説明

関数 ippiDFTFree は、ippi.h ファイルの中で宣言される。この関数は、前に初期化した DFT コンテキスト構造体 *pDFTSpec* が使用していたメモリを解放する。アプリケーション・プログラムは、離散フーリエ変換の計算が終了してから、この関数を呼び出す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pDFTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	*pDFTSpec* 構造体への無効なポインタが渡された場合のエラー状態を示す。

DFTGetBufSize

DFT 関数が使用する外部ワーク・バッファのサイズを求める。

```
IppStatus ippiDFTGetBufSize_R_32s (IppiDFTSpec_R_32s* pDFTSpec,
      int* size);
IppStatus ippiDFTGetBufSize_R_32f (IppiDFTSpec_R_32f* pDFTSpec,
      int* size);
IppStatus ippiDFTGetBufSize_C_32fc (IppiDFTSpec_C_32fc* pDFTSpec,
      int* size);
```

引数

pDFTSpec	前に初期化したコンテキスト構造体へのポインタ
size	DFT 関数が使用する外部ワーク・バッファのサイズへのポインタ

説明

関数 ippiDFTGetBufSize は、ippi.h ファイルの中で宣言される。この関数は、コンテキスト *pDFTSpec* を使用する DFT 関数が、計算中にワークスペースとして使用する外部メモリ・バッファのサイズ（バイト数）を求める。

外部バッファを使用する場合は、pDFTSpec を初期化してから、この関数を呼び出す。
求めた *size* 値をバイト数として、外部バッファの割り当てを行うが、それには ippsMalloc のような関数を使用する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pDFTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	*pDFTSpec* 構造体への無効なポインタが渡された場合のエラー状態を示す。

DFTFwd

画像に順方向離散フーリエ変換を適用する。

事例 1：整数データの画像の操作

```
IppStatus ippiDFTFwd_RToPack_<mod> (const Ipp8u* pSrc, int srcStep,
      Ipp32s* pDst, int dstStep, const IppiDFTSpec_R_32s* pDFTSpec,
      int scaleFactor, Ipp8u* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
8u32s_C1RSfs
8u32s_C3RSfs
8u32s_C4RSfs
8u32s_AC4RSfs
```

事例 2：浮動小数点データの画像の操作

```
IppStatus ippiDFTFwd_RToPack_<mod> (const Ipp32f* pSrc, int srcStep,
      Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,
      Ipp8u* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

事例 3：複素数データの画像の操作

```
IppStatus ippiDFTFwd_CToC_32fc_C1R (const Ipp32fc* pSrc, int srcStep,
     Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc*
     pDFTSpec,          Ipp8u* pBuffer);
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・バッファ内のステップ（バイト単位）
pDFTSpec	前に初期化した DFT コンテキスト構造体へのポインタ
scaleFactor	整数結果のスケーリングに使用される係数
pBuffer	DFT 関数が使用する外部バッファへのポインタ

説明

関数 ippiDFTFwd は、ippi.h ファイルの中で宣言される。この関数は、入力画像 *pSrc* の各チャネルに対して順方向 DFT を実行し、そのフーリエ係数を出力バッファ *pDst* の対応するチャネルに書き込む。

実数データの画像を処理する関数 ippiDFTFwd_RToPack は、対称性を利用して出力データを RCPack2D 形式 で書き込む。この関数は、1、3、4 チャネルの画像を処理できる。AC4 記述子を持つ関数は、アルファ・チャネルを処理しないことに注意する。

複素数データの画像を処理する関数 ippiDFTFwd_CToC は、周波数領域に関する対称性が見られないため、変換結果をパックしない。複素数データの画像のメモリ・レイアウトは、各ピクセル値が実数部と虚数部の 2 つで構成される点を除けば、実数の画像と同じである。

順方向の DFT 関数は、事前に初期化しておいた *pDFTSpec* コンテキスト構造体を使用して、計算モードを設定し、サポート・データを取得する。
また、アプリケーションから、外部ワーク・バッファ *pBuffer* へのポインタを渡すこともできる。その場合は、DFT 関数を呼び出す前に ippiDFTGetBufSize 関数を呼び出して、必要なバッファ・サイズを求める必要がある。NULL ポインタが渡された場合は、ippiDFTFwd 関数が内部でメモリを割り当てる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pDFTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsContextMatchErr	*pDFTSpec* 構造体への無効なポインタが渡された場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

DFTInv

複素数ソース・データに逆方向 DFT を適用し、その結果をデスティネーション・イメージに格納する。

事例 1：整数データの画像の操作

```
IppStatus ippiDFTInv_PackToR_<mod> (const Ipp32s* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, const IppiDFTSpec_R_32s* pDFTSpec,
      int scaleFactor, Ipp8u* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
32s8u_C1RSfs
32s8u_C3RSfs
32s8u_C4RSfs
32s8u_AC4RSfs
```

事例 2：浮動小数点データの画像の操作

```
IppStatus ippiDFTInv_PackToR_<mod> (const Ipp32f* pSrc, int srcStep,
      Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,
      Ipp8u* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

事例 3：複素数データの画像の操作

```
IppStatus ippiDFTInv_CToC_32fc_C1R (const Ipp32fc* pSrc, int srcStep,
      Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc* pDFTSpec,
      Ipp8u* pBuffer);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・バッファ内のステップ（バイト単位）
pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pDFTSpec	前に初期化した DFT コンテキスト構造体へのポインタ
scaleFactor	整数結果のスケーリングに使用される係数
pBuffer	DFT 関数が使用する外部バッファへのポインタ

説明

関数 ippiDFTInv は、ippi.h ファイルの中で宣言される。この関数は、入力バッファ *pSrc* の各チャネルに対して逆方向 DFT を実行し、復元した画像データを、出力イメージ・バッファ *pDst* の対応するチャネルに書き込む。
ippiDFTInv_PackToR で始まる関数を使用する場合は、入力バッファにデータが RCPack2D 形式で入っている必要がある。

逆方向 DFT 関数は、事前に初期化しておいた pDFTSpec コンテキスト構造体を使用して、計算モードを設定し、サポート・データを取得する。
また、アプリケーションから、外部ワーク・バッファ *pBuffer* へのポインタを渡すこともできる。その場合は、DFT 関数を呼び出す前に ippiDFTGetBufSize 関数を呼び出して、必要なバッファ・サイズを求める必要がある。NULL ポインタが渡された場合は、ippiDFTInv 関数が内部でメモリを割り当てる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pDFTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsContextMatchErr	*pDFTSpec* 構造体への無効なポインタが渡された場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

PackToCplxExtend

パックド形式の画像を複素データ画像に変換する。

```
IppStatus ippiPackToCplxExtend_32s32sc_C1R(const Ipp32s* pSrc,
     IppiSize srcSize, int srcStep, Ipp32sc* pDst, int dstStep);
IppStatus ippiPackToCplxExtend_32f32fc_C1R(const Ipp32f* pSrc,
     IppiSize srcSize, int srcStep, Ipp32fc* pDst, int dstStep);
```

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcSize	ソース・イメージのサイズ（ピクセル単位）
srcStep	ソース・バッファ内のステップ（バイト単位）
pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）

説明

関数は ippiPackToCplxExtend は、ippi.h ファイルの中で宣言される。この関数は、RCPack2D 形式 のソース・イメージ *pSrc* 複素データ形式に変換し、その結果を *pDst* に格納する。*pDst* は、一連のフーリエ係数を含む行列である。ただし、

RCPack2D 形式の *pSrc* が次元（NxM）の実数配列である場合、*pDst* は次元（2xNxM）の実数配列になる。この関数の実行のためにメモリを割り当てる際は、このことを考慮に入れる必要がある。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsSizeErr	*srcSize* フィールドの値が 0 または負の場合のエラー状態を示す。

離散コサイン変換

二次元の実数画像に離散コサイン変換（DCT）を適用すると、実数の出力結果が得られるので、変換後のデータをパックド形式で格納する必要はない。ただし、順方向と逆方向の DCT 関数（ippDCTFwd と ippDCTInv）を使用する前に初期化してデータを入れておくコンテキスト・データ構造体は、順方向と逆方向で異なる。同様に、必要なワーク・バッファ・サイズも異なるので、外部バッファを使用する場合は、それぞれに対応するサポート関数を呼び出してそのサイズを求める必要がある。コンテキスト構造体を使用する DCT 関数には、[Rao90] で提案された修正計算アルゴリズムが導入されている。

DCT 関数の ippiDCT8x8Fwd および ippiDCT8x8Inv は、固定の 8 × 8 イメージ・バッファを使用し、コンテキスト・データ・バッファも外部ワーク・バッファも必要としない。関数 ippiDCT8x8Inv は、IEEE-1180 規格の必要条件に適合する（[IEEE] を参照）。

固定の 8 × 8 イメージ・バッファを処理するインテル IPP 離散コサイン変換関数は、SIMD 命令向けに修正された Feig と Winograd のアルゴリズム（[Feig92]）を使用する。DCT 変換に使用されるアルゴリズムの詳細と参考資料については、[AP922] を参照のこと。

DCTFwdInitAlloc

メモリを割り当て、順方向 DCT 関数が必要とするコンテキスト・データを入れる。

```
IppStatus ippiDCTFwdInitAlloc_32f (IppiDCTFwdSpec_32f** pDCTSpec,
     IppiSize roiSize, IppHintAlgorithm hint);
```

引数

pDFTSpec	初期化する順方向 DCT コンテキスト構造体へのポインタ
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
hint	変換関数の計算アルゴリズムを選択するオプション

説明

関数 ippiDCTFwdInitAlloc は、ippi.h ファイルの中で宣言される。この関数は、メモリを割り当て、二次元画像データの順方向 DCT で必要なコンテキスト構造体 *pDCTSpec* を初期化する。ippiDCTFwd 関数は、初期化された *pDCTSpec* 構造体を引数として呼び出され、サイズが *roiSize* の ROI 内のポイントに対して順方向の離散コサイン変換を実行する。計算アルゴリズムは *hint*（表 10-2 を参照）で指定する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pDCTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

DCTInvInitAlloc

メモリを割り当て、逆方向 DCT 関数が必要とするコンテキスト・データを入れる。

```
IppStatus ippiDCTInvInitAlloc_32f (IppiDCTInvSpec_32f** pDCTSpec,
     IppiSize roiSize, IppHintAlgorithm hint);
```

引数

pDFTSpec	初期化する逆方向 DCT コンテキスト構造体へのポインタ
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
hint	変換関数の計算アルゴリズムを選択するオプション

説明

関数 ippiDCTInvInitAlloc は、ippi.h ファイルの中で宣言される。この関数は、メモリを割り当て、二次元画像データの逆方向 DCT で必要なコンテキスト構造体 *pDCTSpec* を初期化する。ippiDCTInv 関数は、初期化された *pDCTSpec* 構造体を引数として呼び出され、サイズが *roiSize* の ROI 内のポイントに対して逆方向の離散コサイン変換を実行する。計算アルゴリズムは *hint*（表 10-2 を参照）で指定する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pDCTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

DCTFwdFree

順方向 DCT コンテキスト構造体が使用して
いたメモリを解放する。

```
IppStatus ippiDCTFwdFree_32f (IppiDCTFwdSpec_32f** pDCTSpec);
```

引数

pDCTSpec	解放する順方向 DCT コンテキスト構造体へのポインタ

説明

関数 ippiDCTFwdFree は、ippi.h ファイルの中で宣言される。この関数は、前に初期化した順方向 DCT コンテキスト構造体 *pDCTSpec* が使用していたメモリを解放する。アプリケーション・プログラムは、順方向離散コサイン変換の計算が終了してから、この関数を呼び出す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pDCTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	*pDCTSpec* 構造体への無効なポインタが渡された場合のエラー状態を示す。

DCTInvFree

逆方向 DCT コンテキスト構造体が使用して
いたメモリを解放する。

```
IppStatus ippiDCTInvFree_32f (IppiDCTInvSpec_32f** pDCTSpec);
```

引数

pDCTSpec	解放する逆方向 DCT コンテキスト構造体へのポインタ

説明

関数 ippiDCTInvFree は、ippi.h ファイルの中で宣言される。この関数は、前に初期化した逆方向 DCT コンテキスト構造体 *pDCTSpec* が使用していたメモリを解放する。アプリケーション・プログラムは、逆方向離散コサイン変換の計算が終了してから、この関数を呼び出す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pDCTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	*pDCTSpec* 構造体への無効なポインタが渡された場合のエラー状態を示す。

DCTFwdGetBufSize

順方向 DCT 関数が使用するワーク・バッファのサイズを求める。

```
IppStatus ippiDCTFwdGetBufSize_32f (IppiDCTFwdSpec_32f** pDCTSpec,
      int* size);
```

引数

pDCTSpec	前に初期化した順方向 DCT コンテキスト構造体へのポインタ
size	順方向 DCT 関数が使用する外部ワーク・バッファのサイズへのポインタ

説明

関数 ippiDCTFwdGetBufSize は、ippi.h ファイルの中で宣言される。この関数は、コンテキスト *pDCTSpec* を使用する順方向 DCT 関数が、計算中にワークスペースとして使用する外部メモリ・バッファのサイズ（バイト数）を求める。
外部バッファを使用する場合は、*pDCTSpec* を初期化してから、この関数を呼び出す。
求めた *size* 値をバイト数として、外部バッファの割り当てを行うが、それには ippsMalloc のような関数を使用する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pDCTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	*pDCTSpec* 構造体への無効なポインタが渡された場合のエラー状態を示す。

DCTInvGetBufSize

逆方向 DCT 関数が使用するワーク・バッファのサイズを求める。

```
IppStatus ippiDCTInvGetBufSize_32f (IppiDCTInvSpec_32f** pDCTSpec,
     int* size);
```

引数

pDCTSpec	前に初期化した逆方向 DCT コンテキスト構造体へのポインタ
size	逆方向 DCT 関数が使用する外部ワーク・バッファのサイズへのポインタ

説明

関数 ippiDCTInvGetBufSize は、ippi.h ファイルの中で宣言される。この関数は、コンテキスト *pDCTSpec* を使用する逆方向 DCT 関数が、計算中にワークスペースとして使用する外部メモリ・バッファのサイズ（バイト数）を求める。

外部バッファを使用する場合は、*pDCTSpec* を初期化してから、この関数を呼び出す。
求めた *size* 値をバイト数として、外部バッファの割り当てを行うが、それには ippsMalloc のような関数を使用する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

ippStsNullPtrErr	*pDCTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	*pDCTSpec* 構造体への無効なポインタが渡された場合のエラー状態を示す。

DCTFwd

画像に順方向離散コサイン変換を適用する。

```
IppStatus ippiDCTFwd_<mod> (const Ipp32f* pSrc, int srcStep,
      Ipp32f* pDst, int dstStep, const IppiDCTFwdSpec_32f* pDCTSpec,
      Ipp32f* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・バッファ内のステップ（バイト単位）
pDCTSpec	前に初期化した順方向 DCT コンテキスト構造体へのポインタ
pBuffer	順方向 DCT 関数が使用する外部バッファへのポインタ

説明

関数 ippiDCTFwd は、ippi.h ファイルの中で宣言される。この関数は、入力画像 *pSrc* の各チャネルに対して順方向 DCT を実行し、その結果を出力画像バッファ *pDst* の対応するチャネルに書き込む。AC4 記述子を持つ関数の種類は、アルファ・チャネルを処理しないことに注意する。
この関数は、事前に初期化しておいた *pDCTSpec* コンテキスト構造体を使用して、計算モードを設定し、サポート・データを取得する。

また、アプリケーションから、外部ワーク・バッファ *pBuffer* へのポインタを渡すこともできる。
その場合は、順方向 DCT 関数を呼び出す前に ippiDCTFwdGetBufSize 関数を呼び出して、必要なバッファ・サイズを求める必要がある。NULL ポインタが渡された場合は、ippiDCTFwd 関数が内部でメモリを割り当てる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pDCTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsContextMatchErr	*pDCTSpec* 構造体への無効なポインタが渡された場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

DCTInv

画像に逆方向離散コサイン変換を適用する。

```
IppStatus ippiDCTInv_<mod> (const Ipp32f* pSrc, int srcStep,
      Ipp32f* pDst, int dstStep, const IppiDCTInvSpec_32f* pDCTSpec,
      Ipp32f* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ

dstStep	デスティネーション・バッファ内のステップ（バイト単位）
pDCTSpec	前に初期化した逆方向 DCT コンテキスト構造体へのポインタ
pBuffer	逆方向 DCT 関数が使用する外部バッファへのポインタ

説明

関数 ippiDCTInv は、ippi.h ファイルの中で宣言される。この関数は、入力画像 *pSrc* の各チャネルに対して逆方向 DCT を実行し、その結果を出力画像バッファ *pDst* の対応するチャネルに書き込む。AC4 記述子を持つ関数の種類は、アルファ・チャネルを処理しないことに注意する。
この関数は、事前に初期化しておいた *pDCTSpec* コンテキスト構造体を使用して、計算モードを設定し、サポート・データを取得する。
また、アプリケーションから、外部ワーク・バッファ *pBuffer* へのポインタを渡すこともできる。その場合は、逆方向 DCT 関数を呼び出す前に ippiDCTInvGetBufSize 関数を呼び出して、必要なバッファ・サイズを求める必要がある。NULL ポインタが渡された場合は、ippiDCTInv 関数が内部でメモリを割り当てる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pDCTSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsContextMatchErr	*pDCTSpe* 構造体への無効なポインタが渡された場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

DCT8x8Fwd

8 × 8 のサイズの 2D バッファに順方向
DCT を実行する。

事例 1：非インプレース操作

```
IppStatus ippiDCT8x8Fwd_<mod>(const Ipp<datatype>* pSrc,
      Ipp<datatype>* pDst);
```

サポートされる *mod* の値は、次のとおりである。

```
16s_C1      32f_C1
```

事例 2：ROI を使用する非インプレース操作

```
IppStatus ippiDCT8x8Fwd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst);
```

サポートされる *mod* の値は、次のとおりである。

```
16s_C1R         8u16s_C1R
```

事例 3：インプレース操作

```
IppStatus ippiDCT8x8Fwd_<mod>(Ipp<datatype>* pSrcDst );
```

サポートされる *mod* の値は、次のとおりである。

```
16s_C1I         32f_C1I
```

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	（ROI を使用する操作の場合）ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ

説明

関数 ippiDCT8x8Fwd は、ippi.h ファイルの中で宣言される。この関数は、8 × 8 のサイズの 2D バッファに入っているショート整数または浮動小数点データに対して、順方向離散コサイン変換を実行する。この変換関数を使用するときは、事前に必要な作業はない。
次のコード例は、ippiDCT8x8Fwd 関数の使用方法を示している。

例 10-4　8 × 8 のサイズの順方向 DCT

```
IppStatus dct16s( void ) {
   Ipp16s x[64] = {0};
   IppiSize roi = {8,8};
   int i;
   for( i=0; i<8; ++i ) {
      ippiSet_16s_C1R( (Ipp16s)i, x+8*i+i, 8*2, roi );
      --roi.width;
      --roi.height;
   }
   return ippiDCT8x8Fwd_16s_C1I( x );
}
```

デスティネーション・イメージ x には、次のデータが含まれる。

```
18 -9 -2 -1  0  0  0  0
-9  7  0  0  0  0  0  0
-2  0  2  0  0  0  0  0
-1  0  0  1  0  0  0  0
 0  0  0  0  1  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

DCT8x8Inv

8 × 8 のサイズの 2D バッファに逆方向 DCT を実行する。

事例 1：非インプレース操作

```
IppStatus ippiDCT8x8Inv_<mod>(const Ipp<datatype>* pSrc,
      Ipp<datatype>* pDst);
```

サポートされる *mod* の値は、次のとおりである。

16s_C1　　32f_C1

事例 2：ROI を使用する非インプレース操作

```
IppStatus ippiDCT8x8Inv_<mod>(const Ipp<srcDatatype>* pSrc,
     Ipp<dstDatatype>* pDst, int dstStep);
```

サポートされる *mod* の値は、次のとおりである。

16s_C1R　　16s8u_C1R

事例 3：インプレース操作

```
IppStatus ippiDCT8x8Inv_<mod>(Ipp<datatype>* pSrcDst);
```

サポートされる *mod* の値は、次のとおりである。

16s_C1I　　32f_C1I

引数

pSrc	ソース・イメージへのポインタ
pDst	デスティネーション・バッファへのポインタ
dstStep	（ROI を使用する操作の場合）デスティネーション・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ

説明

関数 ippiDCT8x8Inv は、ippi.h ファイルの中で宣言される。この関数は、8 × 8 のサイズの 2D バッファに入っているショート整数または浮動小数点データに対して、逆方向離散コサイン変換を実行する。この変換関数を使用するときは、事前に必要な作業はない。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pDst*、または *pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

DCT8x8FwdLS

事前にデータ変換とレベル・シフトを適用した 8 × 8 のサイズの 2D バッファに、順方向 DCT を実行する。

```
IppStatus ippiDCT8x8FwdLS_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, Ipp16s addVal);
```

引数

pSrc	ソース・イメージ・バッファへのポインタ
pDst	デスティネーション・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のストライドまたはステップ（バイト単位）
addVal	レベル・シフトの値

説明

関数 ippiDCT8x8FwdLS は、ippi.h ファイルの中で宣言される。この関数は、最初に、8 × 8 のサイズの 2D バッファ *pSrc* 内のデータを符号なし Ipp8u 型から符号付き Ipp16s 型に変換し、次に、各サンプルに一定の値 *addVal* を加算してレベル・シフト操作を実行する。この処理の後、この関数は、変換されたデータに順方向の離散コサイン変換を実行する。結果は *pDst* に格納される。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

DCT8x8InvLSClip

8 × 8 のサイズの 2D バッファに逆方向 DCT を実行し、データ変換とレベル・シフトを適用する。

```
IppStatus ippiDCT8x8InvLSClip_16s8u_C1R(const Ipp16s* pSrc, Ipp8u*
    pDst, int dstStep, Ipp16s addVal, Ipp8u clipDown, Ipp8u clipUp);
```

引数

pSrc	ソース・イメージ・バッファへのポインタ
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
addVal	レベル・シフトの値
clipDown	出力値の範囲の下限
clipUp	出力値の範囲の上限

説明

関数 ippiDCT8x8InvLSClip は、ippi.h ファイルの中で宣言される。この関数は、8 × 8 のサイズの 2D バッファ *pSrc* に逆方向の離散コサイン変換を実行する。DCT の完了後、この関数は、各サンプルに一定の値 *addVal* を加算してレベル・シフト操作を実行する。最後に、この関数は、符号付き Ipp16s 型から符号なし Ipp8u 型へのデータ変換を実行する。出力データは、[*clipDown .. clipUp*] の範囲でクリッピングされる。結果は *pDst* に格納される。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

画像の統計関数

11

本章では、画像の次の統計パラメータを計算するインテル® インテグレーテッド・パフォーマンス・プリミティブ関数について説明する。

- ピクセル値の和と平均
- ピクセル値の輝度ヒストグラム
- ピクセル値の最小値と最大値
- 次数 0 ～ 3 の空間モーメントと中心モーメント
- 画像のピクセル値、および 2 つの画像のピクセル値の差に対する無限大、*L1*、*L2* ノルム
- 2 つの画像のピクセル値の差に対する無限大、*L1*、*L2* ノルムの相対誤差
- 画像とテンプレート（他の画像）の近接性尺度

表 11-1 に、画像の統計関数の一覧を示す。

表 11-1　画像の統計関数

関数の基本名	操作
Sum	画像のピクセル値の和を計算する。
Mean	画像のピクセル値の平均を計算する。
Mean_StdDev	画像のピクセル値の平均と標準偏差を計算する。
HistogramRange	画像の輝度ヒストグラムを計算する。
HistogramEven	等しいビンを使用して、画像の輝度ヒストグラムを計算する。
CountInRange	指定された輝度範囲内にあるピクセル数をカウントする。
最大値と最小値の演算	
Min	画像のピクセル値の最小値を計算する。
MinIndx	画像のピクセル値の最小値を計算し、最小輝度を持つピクセルの座標を取り出す。
Max	画像のピクセル値の最大値を計算する。
MaxIndx	画像のピクセル値の最大値を計算し、最大輝度を持つピクセルの座標を取り出す。
MinMax	画像のピクセル値の最小値と最大値を計算する。
MinMaxIndx	画像のピクセル値の最小値と最大値を計算し、それぞれのインデックスを取り出す。

表 11-1　画像の統計関数（続き）

関数の基本名	操作
モーメント	
MomentInitAlloc	ippiMoments 関数のためのコンテキストを初期化する。
MomentFree	画像のモーメントを保存するために前に初期化した構造体を解放する。
Moments	画像の 0 ～ 3 次のすべてのモーメントと Hu モーメント不変量を計算する。
GetSpatialMoment	ippiMoments で計算した画像の空間モーメントを取り出す。
GetCentralMoment	ippiMoments で計算した画像の中心モーメントを取り出す。
GetNormalizedSpatialMoment	ippiMoments で計算した画像の空間モーメントを正規化して取り出す。
GetNormalizedCentralMoment	ippiMoments で計算した画像の中心モーメントを正規化して取り出す。
GetHuMoments	ippiMoments で計算した画像の Hu モーメント不変量を取り出す。
ノルム	
Norm_Inf	画像のピクセル値の無限大ノルムを計算する。
Norm_L1	画像のピクセル値の L1 ノルムを計算する。
Norm_L2	画像のピクセル値の L2 ノルムを計算する。
NormDiff_Inf	2 つの画像のピクセル値の差から無限大ノルムを計算する。
NormDiff_L1	2 つの画像のピクセル値の差から L1 ノルムを計算する。
NormDiff_L2	2 つの画像のピクセル値の差から L2 ノルムを計算する。
NormRel_Inf	2 つの画像のピクセル値の差から求めた無限大ノルムの相対誤差を計算する。
NormRel_L1	2 つの画像のピクセル値の差から求めた L1 ノルムの相対誤差を計算する。
NormRel_L2	2 つの画像のピクセル値の差から求めた L2 ノルムの相対誤差を計算する。
近接性尺度	
SqrDistanceFull_Norm	画像とテンプレートの間の正規化された完全ユークリッド距離を計算する。
SqrDistanceSame_Norm	画像とテンプレートの間の正規化されたユークリッド距離を計算する。
SqrDistanceValid_Norm	画像とテンプレートの間の正規化された有効ユークリッド距離を計算する。
CrossCorrFull_Norm	画像とテンプレートの間の正規化された完全相互相関を計算する。
CrossCorrSame_Norm	画像とテンプレートの間の正規化された相互相関を計算する。
CrossCorrValid_Norm	画像とテンプレートの間の正規化された有効相互相関を計算する。
CrossCorrFull_NormLevel	画像とテンプレートの間の正規化された完全相関係数を計算する。
CrossCorrSame_NormLevel	画像とテンプレートの間の正規化された相関係数を計算する。

表 11-1　画像の統計関数 （続き）

関数の基本名	操作
CrossCorrValid_NormLevel	画像とテンプレートの間の正規化された有効相関係数を計算する。

Sum

画像のピクセル値の和を計算する。

事例 1：1 チャネル整数データの操作

```
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f* pSum);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R
```

事例 2：1 チャネル浮動小数点データの操作

```
IppStatus ippiSum_32f_C1R(const Ipp32f* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f* pSum, IppHintAlgorithm hint);
```

事例 3：マルチチャネル整数データの操作

```
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f sum[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R
8u_AC4R     16s_AC4R
```

```
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f sum[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R
```

事例 4：マルチチャネル浮動小数点データの操作

```
IppStatus ippiSum_<mod>(const Ipp32f* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f sum[3], IppHintAlgorithm hint);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiSum_32f_C4R(const Ipp32f* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f sum[4], IppHintAlgorithm hint);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
pSum	計算したピクセル値の和へのポインタ
sum	ソース・バッファ内のピクセルの各カラー・チャネル値の和が入る配列
hint	関数に導入されるアルゴリズムを選択するオプション

説明

関数 ippiSum は、ippi.h ファイルの中で宣言される。この関数は、*hint* 引数（表 11-2を参照）によって指定されるアルゴリズムを使用してソース・イメージ *pSrc* のピクセル値の和 *pSum* を計算する。マルチチャネル・イメージの場合、各チャネルで個別に和を計算し、配列 *sum* に格納する。

次のコード例は、ippiSum 関数の使用方法を示している。

例 11-1　画像のピクセル値の和

```
IppStatus sum( void ) {
   Ipp64f sum;
   Ipp8u x[5*4];
   IppiSize roi = {5,4};
   ippiSet_8u_C1R( 1, x, 5, roi );
   return ippiSum_8u_C1R( x, 5, roi, &sum);
}
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pSum* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

Mean

画像のピクセル値の平均を計算する。

事例 1：1 チャネル整数データの操作

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f* pMean);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1R       16s_C1R
```

事例 2：1 チャネル浮動小数点データの操作

```
IppStatus ippiMean_32f_C1R(const Ipp32f* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f* pMean, IppHintAlgorithm hint);
```

事例 3：1 チャネル・データのマスクを使用した操作

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize,
      Ipp64f* pMean);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1MR      8s_C1MR      32f_C1MR
```

事例 4：マルチチャネル整数データの操作

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f mean[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C3R       16s_C3R
      8u_AC4R      16s_AC4R
```

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f mean[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C4R       16s_C4R
```

事例 5：マルチチャネル浮動小数点データの操作

```
IppStatus ippiMean_<mod>(const Ipp32f* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f mean[3], IppHintAlgorithm hint);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiMean_32f_C4R(const Ipp32f* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f mean[4], IppHintAlgorithm hint);
```

事例 6：マルチチャネル・データのマスクを使用した操作

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
      Ipp64f* pMean);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3CMR     8s_C3CMR     32f_C3CMR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
coi	（カラー・イメージの場合）対象チャネル。1、2、または 3 のいずれか
pMean	計算した平均ピクセル値へのポインタ
mean	マルチチャネル・イメージの各カラー・チャネルの平均値が入る配列
hint	関数に導入されるアルゴリズムを選択するオプション

説明

マスクを使用した操作を実行する関数 ippiMean の種類は、ippcv.h ファイルの中で宣言される。その他のすべての種類の関数は、ippi.h ファイルソース・イメージ *pSrc* のピクセル値の平均 *pMean* を計算する。計算アルゴリズムは、*hint* 引数で指定する（表 11-2 を参照）。マルチチャネル・イメージのマスクを使用しない操作の場

合（**事例 4、5**）、各チャネルについて平均が計算され、配列 *mean* に格納される。マルチチャネル・イメージのマスクを使用した操作の場合（**事例 6**）、*coi* で指定された 1 つの対象チャネルに対して平均を計算する。
次のコード例は、ippiMean 関数の使用方法を示している。

例 11-2　ピクセル値の平均の計算

```
IppStatus mean( void ) {
   Ipp64f mean;
   Ipp8u x[5*4];
   IppiSize roi = {5,4};
   ippiSet_8u_C1R( 3, x, 5, roi );
   return ippiMean_8u_C1R( x, 5, roi, &mean );
}
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	次のような場合のエラー状態を示す。 *srcStep* の値が 0 または負である場合。 マスクを使用した操作で *srcStep* または *maskStep* が *roiSize.width* * <pixelSize> より小さい場合。
ippStsNotEvenStepErr	マスクを使用した操作で、浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsCOIErr	*coi* が 1、2、3 以外である場合のエラー状態を示す。

Mean_StdDev

画像のピクセル値の平均と標準偏差を計算する。

事例 1：1 チャネル・データの操作

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, IppiSize roiSize, Ipp64f* pMean, Ipp64f* pStdDev);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      8s_C1R      32f_C1R
```

事例 2：1 チャネル・データのマスクを使用した操作

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
      Ipp64f* pMean, Ipp64f* pStdDev);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1MR     8s_C1MR     32f_C1MR
```

事例 3：マルチチャネル・データの操作

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, IppiSize roiSize, int coi, Ipp64f* pMean,
      Ipp64f* pStdDev);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3CR     8s_C3CR     32f_C3CR
```

事例 4：マルチチャネル・データのマスクを使用した操作

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc,
     int srcStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
     int coi, Ipp64f* pMean, Ipp64f* pStdDev);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3CMR     8s_C3CMR     32f_C3CMR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
coi	（カラー・イメージの場合）対象チャネル。1、2、または 3 のいずれか
pMean	求められた、ピクセル値の平均へのポインタ
pStdDev	求められた、画像のピクセル値の標準偏差へのポインタ

説明

関数 ippiMean_StdDev は、ippcv.h ファイルの中で宣言される。この関数は、ソース・イメージ *pSrc* の ROI 内のピクセル値に対して、平均と標準偏差を計算する。マスクを使用した操作の場合は、マスク値が 0 でないピクセルだけが計算の対象となる。
マルチチャネル・モードの場合、*coi* で指定された 1 つの対象チャネルに対して平均を計算する。
pMean または *pStdDev* のいずれかが不要な場合は、そのパラメータにゼロ・ポインタを渡す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* ポインタまたは *pMask* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *maskStep* が *roiSize.width* * <pixelSize> より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が4で割り切れない場合のエラー状態を示す。
ippStsCOIErr	*coi* が 1、2、3 以外である場合のエラー状態を示す。

HistogramRange

画像の輝度ヒストグラムを計算する。

事例 1：1 チャネル整数データの操作

```
IppStatus ippiHistogramRange_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist, const Ipp32s* pLevels,
    int nLevels);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R
```

事例 2：マルチチャネル整数データの操作

```
IppStatus ippiHistogramRange_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist[3],
    const Ipp32s* pLevels[3], int nLevels[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R
8u_AC4R     16s_AC4R
```

```
IppStatus ippiHistogramRange_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist[4],
    const Ipp32s* pLevels[4], int nLevels[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R
```

事例 3：1 チャネル浮動小数点データの操作

```
IppStatus ippiHistogramRange_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist, const Ipp32f* pLevels, int
    nLevels);
```

事例 4：マルチチャネル浮動小数点データの操作

```
IppStatus ippiHistogramRange_<mod>(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[3],  const Ipp32f* pLevels[3],
    int nLevels[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiHistogramRange_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[4], const Ipp32f* pLevels[4],
    int nLevels[4]);
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
pHist	計算されたヒストグラムへのポインタ。マルチチャネル・データの場合、*pHist* は、各チャネルのヒストグラムへのポインタの配列になる。
pLevels	レベル値の配列へのポインタ。マルチチャネル・データの場合、*pLevels* は、各チャネルのレベル値の配列へのポインタの配列になる。
nLevels	レベルの数 （各チャネル別）

説明

関数 ippiHistogramRange は、ippi.h ファイルの中で宣言される。この関数は、配列（マルチチャネル・データの場合は複数の配列）*pLevels* によって指定される範囲内の画像の輝度ヒストグラムを計算する。ヒストグラムは、各チャネル別に計算され、配列 *pHist* に格納される。この関数を呼び出す前に、配列 *pLevels* を初期化して、ヒストグラム計算用のビンの上限と下限を指定しておく必要がある。
配列 *pLevels* と *pHist* の長さは、*nLevels* パラメータで定義される。*nLevels* はレベルの数であるため、ヒストグラム・ビンの数は *nLevels* - 1 になる。同様に、配列 *pHist* 内の値の数も *nLevels* - 1 になる。
pHist と *pLevels* の値の意味は、次の例で示される。*pHist*[*k*] は、以下の条件を満たすソース・イメージ・ピクセル *pSrc*(*x*,*y*) の数である。

$$pLevels[k] \le pSrc(x,y) < pLevels[k+1].$$

次のコード例は、ippiHistogramRange 関数の使用方法を示している。

例 11-3　画像のヒストグラムの計算

```
// Compute histogram of an image for 4 bins in the range [28,
127]
{
Ipp8u img[WIDTH*HEIGHT];
IppiSize imgSize = {WIDTH, HEIGHT};
const Ipp32s levels[5] = {28, 50, 70, 90, 128};
Ipp32s histo[5];

ippiHistogramRange_8u_C1R(img, WIDTH, imgSize, levels, histo,
5);

// after executing of the function the array histo
// will contain a histogram in specified range.
// Actually, four result values will be stored

// compute cumulative histogram
for (i = 1; i < 4; i ++)
    histo[i] += histo[i-1];
}
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsMemAllocErr	内部ヒストグラム用のメモリが足りない場合のエラーを示す。
ippStsHistoNofLevelsErr	*nLevels* が 2 より小さい場合のエラーを示す。

HistogramEven

等しいビンを使用して、画像の輝度ヒストグラムを計算する。

事例 1：1 チャネル整数データの操作

```
IppStatus ippiHistogramEven_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist, Ipp32s* pLevels,
    int nLevels, Ipp32s lowerLevel, Ipp32s upperLevel);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R

事例 2：マルチチャネル整数データの操作

```
IppStatus ippiHistogramEven_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist[3], Ipp32s* pLevels[3],
    int nLevels[3], Ipp32s lowerLevel[3], Ipp32s upperLevel[3]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16s_C3R
8u_AC4R	16s_AC4R

```
IppStatus ippiHistogramEven_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist[4], Ipp32s* pLevels[4],
    int nLevels[4], Ipp32s lowerLevel[4], Ipp32s upperLevel[4]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4R	16s_C4R

引数

pSrc	ソース・バッファへのポインタ。
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
pHist	計算されたヒストグラムへのポインタ。マルチチャネル・データの場合、*pHist* は、各チャネルのヒストグラムへのポインタの配列になる。

pLevels	レベル値の配列へのポインタ。マルチチャネル・データの場合、*pLevels* は、各チャネルのレベル値の配列へのポインタの配列になる。
nLevels	レベルの数 （各チャネル別）
lowerLevel	レベルの下限（各チャネル別）
upperLevel	レベルの上限 （各チャネル別）

説明

関数 ippiHistogramEven は、ippi.h ファイルの中で宣言される。この関数は、*lowerLevel*（この値を含む）、*upperLevel*（この値を含まない）、*nLevels* の値によって指定される範囲内の画像の輝度ヒストグラムを計算する。この関数は、すべてのヒストグラム・ビンが同じ幅を持ち、ビン （レベル）の上下限値が等しいことを前提として動作する。
この関数は、各チャネルについて個別に計算したヒストグラムを配列 *pHist* に格納する。計算されたレベル値は、配列 *pLevels* に格納される。

配列 *pLevels* と *pHist* の長さは、*nLevels* パラメータで定義される。*nLevels* はレベルの数であるため、ヒストグラム・ビンの数は *nLevels* - 1 になる。同様に、配列 *pHist* 内の値の数も *nLevels* - 1 になる。
pHist と *pLevels* の値の意味は、次の例で示される。*pHist*[*k*] は、以下の条件を満たすソース・イメージ・ピクセル *pSrc*(*x*,*y*) の数である。

$$pLevels[k] \le pSrc(x,y) < pLevels[k+1].$$

次のコード例は、ippiHistogramEven 関数の使用方法を示している。

例 11-4　画像の均等ヒストグラムの計算

```
// Compute histogram of an image for 4 bins in the range [28, 127];
// compute level values for the bins.
{
    Ipp8u img[WIDTH*HEIGHT];
    IppiSize imgSize = {WIDTH, HEIGHT};
    Ipp32s levels[5], histo[5];

    ippiHistogramEven_8u_C1R(img, WIDTH, imgSize, levels,
                                              histo, 5, 28, 128);

    // When the function completes operation the array histo will
    // contain a histogram in specified range, and the array
    // levels will contain the level values {28, 53, 78, 103, 128}
}
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsMemAllocErr	内部ヒストグラム用のメモリが足りない場合のエラーを示す。
ippStsHistoNofLevelsErr	*nLevels* が 2 より小さい場合のエラーを示す。

CountInRange

指定された輝度範囲内にある
ピクセル数を カウントする。

事例 1：1 チャネル・データの操作

```
IppStatus ippiCountInRange_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, IppiSize roiSize, int* counts, Ipp<datatype>
      lowerBound, Ipp<datatype> upperBound);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      32f_C1R
```

事例 2：マルチチャネル・データの操作

```
IppStatus ippiCountInRange_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, IppiSize roiSize, int counts[3], Ipp<datatype>
      lowerBound[3], Ipp<datatype> upperBound[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      32f_C3R
8u_AC4R     32f_AC4R
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
counts	指定された輝度範囲内にあるピクセルの個数。マルチチャネル・データの場合は、3 つの値を持つ配列
lowerBound	輝度範囲の下限値
upperBound	輝度範囲の上限値

説明

関数 ippiCountInRange は、ippi.h ファイルの中で宣言される。この関数は、輝度が *lowerBound* と *upperBound* の範囲内（上下限値を含む）にあるピクセル数をカウントする。

マルチチャネル・イメージの場合、各カラー・チャネルで個別に輝度範囲内のピクセル数をカウントし、配列 *counts* に 3 つのカウント結果を返す。アルファ・チャネル値は、存在しても処理されない。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。
ippStsRangeErr	*lowerBound* が *upperBound* より大きい場合のエラー状態を示す。

Min

画像のピクセル値の最小値を計算する。

事例 1：1 チャネル・データの操作

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype>* pMin);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R      32f_C1R
```

事例 2：マルチチャネル・データの操作

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype> min[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C3R       16s_C3R      32f_C3R
      8u_AC4R      16s_AC4R     32f_AC4R
```

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype> min[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C4R       16s_C4R      32f_C4R
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
pMin	（1 チャネル・データの場合）最小ピクセル値へのポインタ
min	（マルチチャネル・データの場合）ソース・バッファ内のピクセルのチャネルごとの最小値が入る配列

説明

関数 ippiMin は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ *pSrc* の最小ピクセル値 *pMin* を計算する。マルチチャネル・イメージの場合は、各チャネルで個別に最小値を計算し、配列 *min* に格納する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pMin* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

MinIndx

画像のピクセル値の最小値を計算し、最小輝度を持つピクセルの x 座標と y 座標を取り出す。

事例 1：1 チャネル・データの操作

```
IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype>* pMin, int* pIndexX, int*
      pIndexY);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R     32f_C1R
```

事例 2：マルチチャネル・データの操作

```
IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype> min[3], int indexX[3],
      int indexY[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R     32f_C3R
8u_AC4R     16s_AC4R    32f_AC4R
```

```
IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype> min[4], int indexX[4],
      int indexY[4]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4R　　16s_C4R　　32f_C4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
pMin	（1 チャネル・データの場合）最小ピクセル値へのポインタ
min	（マルチチャネル・データの場合）ソース・バッファ内のピクセルのチャネルごとの最小値が入る配列
pIndexX, pIndexY	最小値を持つピクセルの x 座標と y 座標へのポインタ
indexX, indexY	最小のチャネル値を持つピクセルの x 座標と y 座標が入る配列

説明

関数 ippiMinIndx は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ *pSrc* の最小ピクセル値 *pMin* を計算する。マルチチャネル・イメージの場合は、各チャネルで個別に最小値を計算し、配列 *min* に格納する。この関数は、最小値を持つピクセルの *x* 座標と *y* 座標も返す。同じ最小値を持つピクセルが複数存在する場合は、ソース・バッファの先頭に最も近いピクセルの座標が返される。マルチチャネル・データの場合は、*indexX*[k] と *indexY*[k] が、k 番目 （k = 1,2,3,4）のチャネルで最小輝度を持つピクセルの *x* 座標と *y* 座標を表す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pMin*、*pIndexX*、または *pIndexY* ポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

Max

画像のピクセル値の最大値を計算する。

事例 1：1 チャネル・データの操作

```
IppStatus ippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype>* pMax);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R      32f_C1R
```

事例 2：マルチチャネル・データの操作

```
IppStatus ippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype> max[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R      32f_C3R
8u_AC4R     16s_AC4R     32f_AC4R
```

```
IppStatus ippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype> max[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R      32f_C4
```

引数

pSrc	ソース・バッファへのポインタ

srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
pMax	（1 チャネル・データの場合）最大ピクセル値へのポインタ
max	（マルチチャネル・データの場合）ソース・バッファ内のピクセルのチャネルごとの最大値が入る配列

説明

関数 ippiMax は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ *pSrc* の最大ピクセル値 *pMax* を計算する。マルチチャネル・イメージの場合、各チャネルで個別に最大値を計算し、配列 *max* に格納する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pMax* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

MaxIndx

画像のピクセル値の最大値を計算し、最大輝度を持つピクセルの x 座標と y 座標を取り出す。

事例 1：1 チャネル・データの操作

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype>* pMax, int* pIndexX, int* pIndexY);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16s_C1R	32f_C1R

事例 2：マルチチャネル・データの操作

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype> max[3], int indexX[3],
      int indexY[3]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype> max[4], int indexX[4],
      int indexY[4]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4R	16s_C4R	32f_C4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
pMax	（1 チャネル・データの場合）最大ピクセル値へのポインタ
max	（マルチチャネル・データの場合）ソース・バッファ内のピクセルのチャネルごとの最大値が入る配列

pIndexX, *pIndexY*	最大値を持つピクセルの x 座標と y 座標へのポインタ
indexX, *indexY*	最大のチャネル値を持つピクセルの x 座標と y 座標が入る配列

説明

関数 ippiMaxIndx は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ *pSrc* の最大ピクセル値 *pMax* を計算する。マルチチャネル・イメージの場合は、各チャネルで個別に最大値を計算し、配列 *max* に格納する。この関数は、最大値を持つピクセルの *x* 座標と *y* 座標も返す。同じ最大値を持つピクセルが複数存在するときは、ソース・バッファの先頭に最も近いピクセルの座標が返される。マルチチャネル・データの場合は、*indexX*[k] と *indexY*[k] が、k 番目 （k = 1,2,3,4）のチャネルで最大輝度を持つピクセルの *x* 座標と *y* 座標を表す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pMax*、*pIndexX*、または *pIndexY* ポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

MinMax

画像のピクセル値の最小値と最大値を計算する。

事例 1：1 チャネル・データの操作

```
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype>* pMin, Ipp<datatype>* pMax);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1R      16s_C1R      32f_C1R
```

事例 2：マルチチャネル・データの操作

```
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype> min[3], Ipp<datatype> max[3]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp<datatype> min[4], Ipp<datatype> max[4]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4R	16s_C4R	32f_C4R

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
pMin, *pMax*	（1 チャネル・データの場合）最小および最大のピクセル値へのポインタ
min, *max*	（マルチチャネル・データの場合）ソース・バッファ内のピクセルのカラー・チャネルごとの最小値と最大値が入る配列

説明

関数 ippiMinMax は、ippi.h ファイルの中で宣言される。この関数は、最小ピクセル値 *pMin* と最大ピクセル値 *pMax* を計算する。マルチチャネル・イメージの場合、チャネルごとに最小値と最大値を計算し、配列 *min* および *max* に格納する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc*、*pMin*、または *pMax* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

MinMaxIndx

選択した矩形画像内で最小および最大のピクセル値を計算し、そのインデックスを求める。

事例 1：1 チャネル・データの操作

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc,
int srcStep, IppiSize roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal,
IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R　　8s_C1R　　32f_C1R

事例 2：1 チャネル・データのマスクを使用した操作

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
   const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f* pMinVal,
   Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1MR　　8s_C1MR　　32f_C1MR

事例 3：マルチチャネル・データの操作

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc,
   int srcStep, IppiSize roiSize, int coi, Ipp32f* pMinVal,
   Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3CR　　8s_C3CR　　32f_C3CR

事例 4：マルチチャネル・データのマスクを使用した操作

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
   const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
   Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex,
   IppiPoint* pMaxIndex);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3CMR　　8s_C3CMR　　32f_C3CMR

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
roiSize	イメージ ROI のサイズ（ピクセル単位）
coi	（カラー・イメージの場合）対象チャネル。1、2、または 3 のいずれか
pMinVal	最小ピクセル値を返す変数へのポインタ
pMaxVal	最大ピクセル値を返す変数へのポインタ
pMinIndex	検出された最小値のインデックスを返す変数へのポインタ
pMaxIndex	検出された最大値のインデックスを返す変数へのポインタ

説明

関数 ippiMinMaxIndx は、ippcv.h ファイルの中で宣言される。この関数は、イメージ ROI 内、または 0 以外のマスク値で決まる任意の画像領域内で、最小および最大のピクセル値とそのインデックスを見つけ出す。
指定された領域内に最小値または最大値を持つピクセルが複数存在する場合は、最も左に位置するピクセルのインデックスを返す。
マスクで指定された領域が空のとき、すなわちマスク・イメージ内がすべて 0 のときは、{*minIndex*, *maxIndex*} = {0, 0}、*minVal* = *maxVal* = 0 を返す。
pMinVal、*pMaxVal*、*pMinIndex*、または *pMaxIndex* のうち、不要なパラメータがあれば、そのパラメータにゼロ・ポインタを渡す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pMask* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *maskStep* が *roiSize.width* * <*pixelSize*> より小さい場合のエラー状態を示す。

ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラーを示す。
ippStsCOIErr	*coi* が 1、2、3 以外である場合のエラー状態を示す。

画像のモーメント

空間モーメントと中心モーメントは、画像の重要な統計特性である。次数 (*m*,*n*) の空間モーメント $M_U(m,n)$ は次のように定義される。

$$M_U(m,n) = \sum_j \sum_k x_k^m y_j^n P_{j,k}$$

この式では、画像のすべての行と列に渡って和が計算される。$P_{j,k}$ は、ピクセル値である。x_k と y_j は、ピクセルの座標である。*m* と *n* は、モーメントの次数を決定する、整数の指数である。

中心モーメント $U_U(m,n)$ は、「重心」(x_0, y_0) に対して計算された空間モーメントである。

$$U_U(m,n) = \sum_j \sum_k (x_k - x_0)^m (y_j - y_0)^n P_{j,k}$$

$x_0 = M_U(1,0)/M_U(0,0)$、$y_0 = M_U(0,1)/M_U(0,0)$ である。

正規化した空間モーメント $M(m,n)$ と中心モーメント $U(m,n)$ は次のように定義される。

$$M(m,n) = \frac{M_U(m,n)}{M_U(0,0)^{\frac{m+n+2}{2}}}$$

$$U(m,n) = \frac{U_U(m,n)}{U_U(0,0)^{\frac{m+n+2}{2}}}$$

インテル IPP 関数は、次数 (*m*, *n*) のモーメント（ただし、$0 \le m + n \le 3$）をサポートしている。また、2 次と 3 次のモーメントから導出される 7 つの不変 Hu モーメントの計算もサポートしている。計算されたモーメントは、タイプが IppiMomentState_64s （整数バージョン）または IppiMomentState_64f （浮動小数点バージョン）のコンテキスト構造体に保存される。

画像のモーメントを計算するほとんどのインテル IPP 関数は、異なる計算アルゴリズムを実行するためのコード分岐を持つ。関数が使用するコードを選択するには、*hint* 引数に表 11-2 に示す次のいずれかの値を設定して指定する。

表 11-2　画像のモーメント関数の hint 引数

値	説明
ippAlgHintNone	計算アルゴリズムは関数の内部ロジックで選択される。
ippAlgHintFast	高速アルゴリズムを使用する。出力結果の精度は低下する。
ippAlgHintAccurate	高精度アルゴリズムを使用する。関数の実行時間は長くなる。

MomentInitAlloc

ippiMoments 関数のためのコンテキストを初期化する。

```
IppStatus ippiMomentInitAlloc_64f(IppiMomentState_64f** pState,
      IppHintAlgorithm hint);
IppStatus ippiMomentInitAlloc_64s(IppiMomentState_64s** pState,
      IppHintAlgorithm hint);
```

引数

pState	モーメント値を格納する構造体へのポインタ
hint	関数の計算アルゴリズムを選択するオプション

説明

関数 ippiMomentInitAlloc は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiMoments が画像のモーメントを計算して格納するのに必要な構造体を初期化する。データ・タイプが整数か浮動小数点かによって、構造体を初期化する関数が異なる。計算アルゴリズムは、*hint* 引数で指定する（表 11-2 を参照）。

ippiMomentInitAlloc 関数は、初期化に成功すると、割り当てた IppiMomentState 型の構造体へのポインタを *pState* に設定する。

メモリ割り当てに失敗した場合は、ippStsMemAllocErr エラー・ステータス・コードを返す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

MomentFree

前に初期化した *pState* 構造体を解放する。

```
IppStatus ippiMomentFree_64f(IppiMomentState_64f* pState);

IppStatus ippiMomentFree_64s(IppiMomentState_64s* pState);
```

引数

pState	画像のモーメントを格納する構造体へのポインタ

説明

関数 ippiMomentFree は、ippi.h ファイルの中で宣言される。この関数を使用して、画像のモーメント・データを格納するために前に初期化しておいた *pState* 構造体を解放する。データ・タイプが整数か浮動小数点かによって、構造体を解放する関数が異なる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pState* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	無効な構造体へのポインタが渡された場合のエラー状態を示す。

Moments

画像の 0 ～ 3 次のすべてのモーメントと、
Hu モーメント不変量を計算する。

事例 1：浮動小数点結果の計算

```
IppStatus ippiMoments64f_<mod>(const Ipp<datatype>* pSrc,
      int srcStep, IppiSize roiSize, IppiMomentState_64f* pState);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      32f_C1R
8u_C3R      32f_C3R
8u_AC4R     32f_AC4R
```

事例 2：整数結果の計算

```
IppStatus ippiMoments64s_<mod>(const Ipp8u* pSrc, int srcStep,
      IppiSize roiSize, IppiMomentState_64s* pState);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R
8u_C3R
8u_AC4R
```

引数

pState	画像のモーメントを格納する構造体へのポインタ
pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）

説明

関数 ippiMoments は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ *pSrc* について、0 ～ 3 次の空間モーメントと中心モーメントをすべて計算する。また、7 つの Hu モーメント不変量も計算する。
画像のモーメントを整数形式で計算するか浮動小数点形式で計算するかによって、ippiMoments64s と ippiMoments64f の 2 つの関数を使い分ける。

ippiMoments 関数は、*pSrc* が指すイメージ・ポイントを基準とした空間モーメントを計算する。このポイントは、ROI の原点であり、画像全体の原点とは限らないことに注意する。実際の画像の原点を基準とした空間モーメントを計算するには、ippiGetSpatialMoment 関数を使用して、再計算する必要がある。

計算したモーメント値は、*pState* 構造体に保存される。特定のモーメント値を取り出すには、以降に説明する関数の適切なものを使用する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pState* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。
ippStsContextMatchErr	無効な構造体へのポインタが渡された場合のエラー状態を示す。

GetSpatialMoment

ippiMoments で計算した画像のモーメント値から、指定された次数の空間モーメントを取り出す。

```
IppStatus ippiGetSpatialMoment_64f(const IppiMomentState_64f*
      pState, int mOrd, int nOrd, int nChannel, IppiPoint
      roiOffset, Ipp64f* pValue);
IppStatus ippiGetSpatialMoment_64s(const IppiMomentState_64s*
      pState, int mOrd, int nOrd, int nChannel, IppiPoint
      roiOffset, Ipp64s* pValue, int scaleFactor);
```

引数

pState	画像のモーメントが格納されている構造体へのポインタ

mOrd, *nOrd*	モーメントの次数を決める整数の指数。これらの引数は、次の条件を満たす必要がある。 $0 \leq$ *mOrd* + *nOrd* ≤ 3.
nChannel	モーメントを返すチャネル
roiOffset	画像の原点から見た ROI 原点（左上隅）のオフセット（ピクセル単位）
pValue	取り出したモーメント値へのポインタ
scaleFactor	（整数データの場合）モメント値のスケーリングに使用される係数

説明

関数 ippiGetSpatialMoment は、ippi.h ファイルの中で宣言される。この関数は、前に ippiMoments 関数で計算した空間モーメントへのポインタ *pValue* を返す。ippiMoments 関数により、画像の ROI 原点を基準とした空間モーメントがすべて計算される。
roiOffset を適切に設定すれば、画像内の別のポイントを基準とした空間モーメントを取り出せる。

モーメントの次数は、整数の指数 *mOrd*、*nOrd* で指定する。

空間モーメントを整数形式で取り出すか浮動小数点形式で取り出すかによって、異なる関数を使用する。整数データの場合は、*scaleFactor* 引数で指定した係数で、結果をスケーリングできる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pState* または *pValue* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	無効な構造体へのポインタが渡された場合のエラー状態を示す。
ippStsSizeErr	*mOrd* + *nOrd* が 3 より大きいか、*nChannel* が無効な値である場合のエラー状態を示す。

GetNormalizedSpatialMoment

ippiMoments で計算した画像の空間モーメント値を正規化して取り出す。

```
IppStatus ippiGetNormalizedSpatialMoment_64f(IppiMomentState_64f*
      pState, int mOrd, int nOrd, int nChannel, IppiPoint
      roiOffset, Ipp64f* pValue);
IppStatus ippiGetNormalizedSpatialMoment_64s(IppiMomentState_64s*
      pState, int mOrd, int nOrd, int nChannel, IppiPoint
      roiOffset, Ipp64s* pValue, int scaleFactor);
```

引数

pState	画像のモーメントが格納される構造体へのポインタ
mOrd, *nOrd*	モーメントの次数を決める整数の指数。この引数は、次の条件を満たす必要がある。 $0 \le mOrd + nOrd \le 3$.
nChannel	モーメントを返すチャネル
roiOffset	画像の原点から見た ROI 原点（左上隅）のオフセット（ピクセル単位）
pValue	正規化したモーメント値へのポインタ
scaleFactor	（整数データの場合）モーメント値のスケーリングに使用する係数

説明

関数 ippiGetNormalizedSpatialMoment は、ippi.h ファイルの中で宣言される。この関数は、前に ippiMoments 関数で計算した空間モーメント値を正規化し、正規化済みモーメントへのポインタ *pValue* を返す。モーメントの正規化の詳細については、「画像のモーメント」を参照のこと。モーメントの次数（*mOrd*、*nOrd*）は、整数の指数で指定する。
ippiMoments 関数により、画像の ROI 原点を基準とした空間モーメントがすべて計算される。*roiOffset* を適切に設定すれば、画像内の別のポイントを基準とした正規化済み空間モーメントを取り出せる。

正規化済み空間モーメントを整数形式で取り出すか浮動小数点形式で取り出すかによって、異なる関数を使用する。整数データの場合は、*scaleFactor* 引数で指定した係数で、結果をスケーリングできる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pState* または *pValue* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	無効な構造体へのポインタが渡された場合のエラー状態を示す。
ippStsMoment00ZeroErr	$M(0,0)$ 値が 0 に近い場合のエラー状態を示す。
ippStsSizeErr	*mOrd* + *nOrd* が 3 より大きいか、あるいは *nChannel* が無効な値である場合のエラー状態を示す。

GetCentralMoment

ippiMoments で計算した、画像の中心モーメントを取り出す。

```
IppStatus ippiGetCentralMoment_64f(const IppiMomentState_64f*
      pState, int mOrd, int nOrd, int nChannel, Ipp64f* pValue);
IppStatus ippiGetCentralMoment_64s(const IppiMomentState_64s*
      pState, int mOrd, int nOrd, int nChannel, Ipp64s* pValue,
      int scaleFactor);
```

引数

pState	画像のモーメントが格納されている構造体へのポインタ
mOrd, *nOrd*	モーメントの次数を決める整数の指数。これらの引数は、次の条件を満たす必要がある。 $0 \le$ *mOrd* + *nOrd* ≤ 3.
nChannel	モーメントを返すチャネル
pValue	取り出したモーメント値へのポインタ
scaleFactor	（整数データの場合）モーメント値のスケーリングに使用する係数

説明

関数 ippiGetCentralMoment は、ippi.h ファイルの中で宣言される。この関数は、前に ippiMoments 関数で計算した中心モーメントへのポインタ *pValue* を返す。モーメントの次数は、整数の指数 *mOrd*、*nOrd* で指定する。

中心モーメントを整数形式で取り出すか浮動小数点形式で取り出すかによって、異なる関数を使用する。整数データの場合は、*scaleFactor* 引数で指定した係数で、結果をスケーリングできる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pState* または *pValue* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	無効な構造体へのポインタが渡された場合のエラー状態を示す。
ippStsSizeErr	*mOrd + nOrd* が 3 より大きいか、あるいは *nChannel* が無効な値である場合のエラー状態を示す。

GetNormalizedCentralMoment

ippiMoments で計算した画像の中心モーメント値を正規化して取り出す。

```
IppStatus ippiGetNormalizedCentralMoment_64f(IppiMomentState_64f*
      pState, int mOrd, int nOrd, int nChannel, Ipp64f* pValue);
IppStatus ippiGetNormalizedCentralMoment_64s(IppiMomentState_64s*
      pState, int mOrd, int nOrd, int nChannel, Ipp64s* pValue,
      int scaleFactor);
```

引数

pState	画像のモーメントが格納されている構造体へのポインタ
mOrd, *nOrd*	モーメントの次数を決める整数の指数。これらの引数は、次の条件を満たす必要がある。 $0 \le mOrd + nOrd \le 3$.

nChannel	モーメントを返すチャネル
pValue	取り出したモーメント値へのポインタ
scaleFactor	（整数データの場合）モーメント値のスケーリングに使用する係数

説明

関数 ippiGetNormalizedCentralMoment は、ippi.h ファイルの中で宣言される。この関数は、前に ippiMoments 関数で計算した中心モーメント値を正規化し、正規化済みモーメントへのポインタ *pValue* を返す。モーメントの次数 (*mOrd*、*nOrd*) は、整数の指数で指定する。モーメントの正規化の詳細については、「画像のモーメント」を参照のこと。
正規化済み中心モーメントを整数形式で取り出すか浮動小数点形式で取り出すかによって、異なる関数を使用する。整数データの場合は、*scaleFactor* 引数で指定した係数で、結果をスケーリングできる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pState* または *pValue* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	無効な構造体へのポインタが渡された場合のエラー状態を示す。
ippStsMoment00ZeroErr	$M(0,0)$ 値が 0 に近い場合のエラー状態を示す。

GetHuMoments

ippiMoments 関数で計算した、画像の Hu モーメント不変量を取り出す。

```
IppStatus ippiGetHuMoments_64f(IppiMomentState_64f* pState, int
      nChannel, IppiHuMoment_64f* pHm);
IppStatus ippiGetHuMoments_64s(IppiMomentState_64s* pState, int
      nChannel, IppiHuMoment_64s* pHm, int scaleFactor);
```

引数

pState	画像のモーメントが格納されている構造体へのポインタ
nChannel	モーメントを返すチャネル
pHm	Hu モーメント不変量が入った配列へのポインタ
scaleFactor	（整数データの場合）モーメント値のスケーリングに使用する係数

説明

関数 ippiGetHuMoments は、ippi.h ファイルの中で宣言される。この関数は、前に ippiMoments 関数で計算した 7 つの Hu モーメント不変量が入った配列へのポインタ *pHm* を返す。

Hu モーメントを整数形式で取り出すか浮動小数点形式で取り出すかによって、異なる関数を使用する。整数データの場合は、*scaleFactor* 引数で指定した係数で、結果をスケーリングできる。

次のコード例は、水平方向と垂直方向のラインを持つ画像の Hu モーメント不変量を計算する方法を示している。

例 11-5　Hu モーメントの計算

```
IppStatus mom8u( void ) {
   IppStatus stH, stV;
   Ipp8u x[64];
   IppiSize roiA={8,8}, roiH={8,1}, roiV={1,8};
   IppiHuMoment_64f hmH, hmV;
   IppiMomentState_64f* ctx;

   stH = ippiMomentInitAlloc_64f( &ctx, ippAlgHintNone );

   ippiSet_8u_C1R( 0, x, 8, roiA );
   ippiSet_8u_C1R( 3, x, 8, roiH );
   stH = ippiMoments64f_8u_C1R( x, 8, roiA, ctx );
   ippiGetHuMoments_64f( ctx, 0, hmH );

   ippiSet_8u_C1R( 0, x, 8, roiA );
   ippiSet_8u_C1R( 3, x, 8, roiV );
   stV = ippiMoments64f_8u_C1R( x, 8, roiA, ctx );
   ippiGetHuMoments_64f( ctx, 0, hmV );

   ippiMomentFree_64f( ctx );
   return ippStsNoErr==stH ? stV : stH;
}
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pState* または *pHm* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	無効な構造体へのポインタが渡された場合のエラー状態を示す。
ippStsMoment00ZeroErr	$M(0,0)$ 値が 0 に近い場合のエラー状態を示す。

画像のノルム

本節では、画像のピクセル値から次のノルムを計算する関数について説明する。

- 無限大ノルム（ピクセルの絶対値の最大値）
- L1 ノルム （ピクセルの絶対値の和）
- L2 ノルム （ピクセル値の二乗和の平方根）

このグループに属する関数は、2 つの入力画像からピクセル値の差のノルムを計算したり、2 つの入力画像の相対誤差を計算したりするのにも役立つ。

Norm_Inf

画像のピクセル値の無限大ノルムを計算する。

事例 1：1 チャネル・データの操作

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f* pValue);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R     32s_C1R     32f_C1R
```

事例 2：1 チャネル・データのマスクを使用した操作

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize,
      Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1MR     8s_C1MR     32f_C1MR
```

事例 3：マルチチャネル・データの操作

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f value[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R     32f_C3R
8u_AC4R     16s_AC4R    32f_AC4R
```

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f value[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R      32f_C4R
```

事例 4：マルチチャネル・データのマスクを使用した操作

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
      Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3CMR    8s_C3CMR    32f_C3CMR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
coi	（カラー・イメージの場合）対象チャネル。1、2、または 3 のいずれか
pValue	ピクセル値から計算した無限大ノルムへのポインタ。
value	マルチチャネル・データの場合は、各チャネル値から計算した無限大ノルムが入った配列
pNorm	（マスクを使用した操作の場合）計算したノルム値へのポインタ

説明

マスクを使用した操作を実行する関数 ippiNorm_Inf の種類は、ippcv.h ファイルの中で宣言される。その他のすべての種類の関数は、ippi.h ファイルの中で宣言される。関数 ippiNorm_Inf は、ソース・イメージ *pSrc* の無限大ノルム *pValue*（マスクを使用した操作では *pNorm*）を計算する。無限大ノルムとは、画像内のピクセルの絶対値の最大値で定義される。マスクを使用した操作の場合は、マスク値が 0 でないピクセルだけが計算の対象となる。

マルチチャネル・イメージのマスクなし操作の場合（**事例 3**）は、各チャネルで個別にノルムを計算し、配列 *value* に格納する。

マルチチャネル・イメージのマスクを使用した操作の場合（**事例 4**）、*coi* で指定された 1 つの対象チャネルに対してのみノルムを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	次のような場合のエラー状態を示す。 *srcStep* の値が 0 または負である場合。 マスクを使用した操作で *srcStep* または *maskStep* が *roiSize.width* * <pixelSize> より小さい場合。
ippStsNotEvenStepErr	マスクを使用した操作で、浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsCOIErr	*coi* が 1、2、3 以外である場合のエラー状態を示す。

Norm_L1

画像のピクセル値の L1 ノルムを計算する。

事例 1：1 チャネルの整数データの操作

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f* pValue);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R
```

事例 2：1 チャネルの浮動小数点データの操作

```
IppStatus ippiNorm_L1_32f_C1R(const Ipp32f* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

事例 3：1 チャネル・データのマスクを使用した操作

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     const Ipp8u* pMask, int maskStep, IppiSize roiSize,
     Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1MR     8s_C1MR     32f_C1MR
```

事例 4：マルチチャネルの整数データの操作

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     IppiSize roiSize, Ipp64f value[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R
8u_AC4R     16s_AC4R
```

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     IppiSize roiSize, Ipp64f value[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R
```

事例 5：マルチチャネルの浮動小数点データの操作

```
IppStatus ippiNorm_L1_<mod>(const Ipp32f* pSrc, int srcStep,
     IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiNorm_L1_32f_C4R(const Ipp32f* pSrc, int srcStep,
     IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

事例 6：マルチチャネル・データのマスクを使用した操作

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
     const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
     Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3CMR    8s_C3CMR    32f_C3CMR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
coi	（カラー・イメージの場合）対象チャネル。1、2、または 3 のいずれか
pValue	ピクセル値から計算した L1 ノルムへのポインタ。
value	マルチチャネル・データの場合は、各チャネル値から計算した L1 ノルムが入った配列
pNorm	（マスクを使用した操作の場合）計算したノルム値へのポインタ
hint	関数の計算アルゴリズムを選択するオプション

説明

マスクを使用した操作を実行する関数 ippiNorm_Inf の種類は、ippcv.h ファイルの中で宣言される。その他のすべての種類の関数は、ippi.h ファイルの中で宣言される。関数 ippiNorm_L1 は、ソース・イメージ *pSrc* の L1 ノルム *pValue*（マスクを使用した操作では *pNorm*）を計算する。L1 ノルムとは、画像内のピクセルの絶対値の和で定義される。計算アルゴリズムは、*hint* 引数で指定する（表 11-2 を参照）。マスクを使用した操作の場合は、マスク値が 0 でないピクセルだけが計算の対象となる。

マルチチャネル・イメージのマスクなし操作の場合（**事例 4、5**）は、各カラー・チャネルで個別にノルムを計算し、配列 *value* 格納する。

マルチチャネル・イメージのマスクを使用した操作の場合（**事例 6**）は、*coi* で指定された 1 つの対象チャネルに対してのみノルムを計算する。

例 11-6　画像のノルムの計算

```
IppStatus norm( void ) {
   Ipp64f sum, normL1;
   Ipp8u x[5*4];
   IppiSize roi = {5,4};
   ippiSet_8u_C1R( 1, x, 5, roi );
   ippiSum_8u_C1R( x, 5, roi, &sum);
   return ippiNorm_L1_8u_C1R( x, 5, roi, &normL1 );
}
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	次のような場合のエラー状態を示す。 *srcStep* の値が 0 または負である場合。 マスクを使用した操作で *srcStep* または *maskStep* が *roiSize.width* * <pixelSize> より小さい場合。
ippStsNotEvenStepErr	マスクを使用した操作で、浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsCOIErr	*coi* が 1、2、3 以外である場合のエラー状態を示す。

Norm_L2

画像のピクセル値の L2 ノルムを計算する。

事例 1：1 チャネルの整数データの操作

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f* pValue);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R
```

事例 2：1 チャネルの浮動小数点データの操作

```
IppStatus ippiNorm_L2_32f_C1R(const Ipp32f* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

事例 3：1 チャネル・データのマスクを使用した操作

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize,
      Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1MR     8s_C1MR     32f_C1MR
```

事例 4：マルチチャネルの整数データの操作

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f value[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R
8u_AC4R     16s_AC4R
```

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f value[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R
```

事例 5：マルチチャネルの浮動小数点データの操作

```
IppStatus ippiNorm_L2_<mod>(const Ipp32f* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiNorm_L2_32f_C4R(const Ipp32f* pSrc, int srcStep,
      IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

事例 6：マルチチャネル・データのマスクを使用した操作

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
      Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3CMR     8s_C3CMR     32f_C3CMR
```

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
coi	（カラー・イメージの場合）対象チャネル。1、2、または 3 のいずれか
pValue	ピクセル値から計算した L2 ノルムへのポインタ。
value	マルチチャネル・データの場合は、各チャネル値から計算した L2 ノルムが入った配列
pNorm	（マスクを使用した操作の場合）計算したノルム値へのポインタ
hint	関数の計算アルゴリズムを選択するオプション

説明

マスクを使用した操作を実行する関数 ippiNorm_L2 の種類は、ippcv.h ファイルの中で宣言される。その他のすべての種類の関数は、ippi.h ファイルの中で宣言される。関数 ippiNorm_L2 は、ソース・イメージ *pSrc* の L2 ノルム *pValue*（マスクを使用した操作では *pNorm*）を計算する。L2 ノルムとは、画像内のピクセル値の二乗和の平方根で定義される。計算アルゴリズムは、*hint* 引数で指定する（表 11-2 を参照）。マスクを使用した操作の場合は、マスク値が 0 でないピクセルだけが計算の対象となる。

マルチチャネル・イメージのマスクなし操作の場合（**事例 4、5**）は、各チャネルで個別にノルムを計算し、配列 *value* に格納する。

マルチチャネル・イメージのマスクを使用した操作の場合（**事例 6**）は、*coi* で指定された 1 つの対象チャネルに対してのみノルムを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	次のような場合のエラー状態を示す。 *srcStep* の値が 0 または負である場合。 マスクを使用した操作で *srcStep* または *maskStep* が *roiSize.width* * <pixelSize> より小さい場合。
ippStsNotEvenStepErr	マスクを使用した操作で、浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsCOIErr	*coi* が 1、2、3 以外である場合のエラー状態を示す。

NormDiff_Inf

2 つの画像のピクセル値の差から無限大ノルムを計算する。

事例 1：1 チャネル・データの操作

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f* pValue);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R     32f_C1R
```

事例 2：1 チャネル・データのマスクを使用した操作

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize,
      Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1MR     8s_C1MR     32f_C1MR
```

事例 3：マルチチャネル・データの操作

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R     32f_C3R
8u_AC4R     16s_AC4R    32f_AC4R
```

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R     32f_C4R
```

事例 4：マルチチャネル・データのマスクを使用した操作

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize,
      int coi, Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3CMR     8s_C3CMR     32f_C3CMR
```

引数

pSrc1, *pSrc2*	ソース・イメージ・バッファへのポインタ
src1Step, *src2Step*	ソース・イメージ・バッファ内のステップ（バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
pValue	ピクセル値の差から計算した無限大ノルムへのポインタ。
value	マルチチャネル・データの場合は、各チャネル値の差から計算した無限大ノルムが入った配列
coi	（カラー・イメージの場合）対象チャネル。1、2、または 3 のいずれか
pNorm	（マスクを使用した操作の場合）計算したノルム値へのポインタ

説明

マスクを使用した操作を実行する関数 ippiNorm_Inf の種類は、ippcv.h ファイルの中で宣言される。その他のすべての種類の関数は、ippi.h ファイルの中で宣言される。関数 ippiNormDiff_Inf は、2 つのソース・イメージ *pSrc1* および *pSrc2* のピクセル値の差から無限大ノルム *pValue*（マスクを使用した操作では *pNorm*）を計算する。無限大ノルムとは、差の絶対値の最大値で定義される。

$$norm = \max |pSrc1 - pSrc2|$$

マスクを使用した操作の場合は、マスク値が 0 でないピクセルだけが計算の対象となる。

マルチチャネル・イメージのマスクなし操作の場合（**事例 3**）は、対応する各チャネルのペアで個別にノルムを計算し、配列 *value* に格納する。

マルチチャネル・イメージのマスクを使用した操作の場合（**事例 4**）、*coi* で指定された 1 つの対象チャネルに対してのみノルムを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	次のような場合のエラー状態を示す。 *src1Step* または *src2Step* の値が 0 または負である場合。 マスクを使用した操作で *src1Step*、*src2Step*、または *maskStep* が *roiSize.width* * <pixelSize> より小さい場合。
ippStsNotEvenStepErr	マスクを使用した操作で、浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsCOIErr	*coi* が 1、2、3 以外である場合のエラー状態を示す。

NormDiff_L1

2 つの画像のピクセル値の差から L1 ノルムを計算する。

事例 1：1 チャネルの整数データの操作

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f* pValue);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1R      16s_C1R
```

事例 2：1 チャネルの浮動小数点データの操作

```
IppStatus ippiNormDiff_L1_32f_C1R(const Ipp32f* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

事例 3：1 チャネル・データのマスクを使用した操作

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize,
      Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1MR     8s_C1MR     32f_C1MR
```

事例 4：マルチチャネルの整数データの操作

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R
8u_AC4R     16s_AC4R
```

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R
```

事例 5：マルチチャネルの浮動小数点データの操作

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp32f* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiNormDiff_L1_32f_C4R(const Ipp32f* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

事例 6：マルチチャネル・データのマスクを使用した操作

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
      Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3CMR     8s_C3CMR     32f_C3CMR
```

引数

pSrc1, *pSrc2*	ソース・イメージ・バッファへのポインタ
src1Step, *src2Step*	ソース・イメージ・バッファ内のステップ（バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
coi	（カラー・イメージの場合）対象チャネル。1、2、または 3 のいずれか
pValue	ピクセル値の差から計算した L1 ノルムへのポインタ。
value	マルチチャネル・データの場合は、各チャネル値の差から計算した L1 ノルムが入った配列
pNorm	（マスクを使用した操作の場合）計算したノルム値へのポインタ
hint	関数の計算アルゴリズムを選択するオプション

説明

マスクを使用した操作を実行する関数 ippiNormDiff_L1 の種類は、ippcv.h ファイルの中で宣言される。その他のすべての種類の関数は、ippi.h ファイルの中で宣言される。関数 ippiNormDiff_L1 は、2 つのソース・イメージ・バッファ *pSrc1* および *pSrc2* のピクセル値の差から L1 ノルム *pValue*（マスクを使用した操作では *pNorm*）を計算する。
L1 ノルムとは、差の絶対値の和で定義される。

$$\text{norm} = \sum |pSrc1 - pSrc2|$$

計算アルゴリズムは、*hint* 引数で指定する（表 11-2 を参照）。マスクを使用した操作の場合は、マスク値が 0 でないピクセルだけが計算の対象となる。

マルチチャネル・イメージのマスクなし操作の場合（**事例 4、5**）は、対応する各チャネルのペアで個別にノルムを計算し、配列 *value* に格納する。

マルチチャネル・イメージのマスクを使用した操作の場合（**事例 6**）は、*coi* で指定された 1 つの対象チャネルに対してのみノルムを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	次のような場合のエラー状態を示す。 *src1Step* または *src2Step* の値が 0 または負である場合。 マスクを使用した操作で *src1Step*、*src2Step*、または *maskStep* が *roiSize.width* * <pixelSize> より小さい場合。
ippStsNotEvenStepErr	マスクを使用した操作で、浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsCOIErr	*coi* が 1、2、3 以外である場合のエラー状態を示す。

NormDiff_L2

2 つの画像のピクセル値の差から L2 ノルムを計算する。

事例 1：1 チャネルの整数データの操作

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f* pValue);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1R      16s_C1R
```

事例 2：1 チャネルの浮動小数点データの操作

```
IppStatus ippiNormDiff_L2_32f_C1R(const Ipp32f* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

事例 3：1 チャネル・データのマスクを使用した操作

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize,
      Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1MR     8s_C1MR     32f_C1MR
```

事例 4：マルチチャネルの整数データの操作

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R
8u_AC4R     16s_AC4R
```

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R
```

事例 5：マルチチャネルの浮動小数点データの操作

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp32f* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiNormDiff_L2_32f_C4R(const Ipp32f* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

事例 6：マルチチャネル・データのマスクを使用した操作

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
     const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
      Ipp64f* norm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3CMR     8s_C3CMR     32f_C3CMR
```

引数

pSrc1, *pSrc2*	ソース・イメージ・バッファへのポインタ
src1Step, *src2Step*	ソース・イメージ・バッファ内のステップ（バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
coi	（カラー・イメージの場合）対象チャネル。1、2、または 3 のいずれか
pValue	ピクセル値の差から計算した L2 ノルムへのポインタ。
value	マルチチャネル・データの場合は、各チャネル値の差から計算した L2 ノルムが入った配列
pNorm	（マスクを使用した操作の場合）計算したノルム値へのポインタ
hint	関数の計算アルゴリズムを選択するオプション

説明

マスクを使用した操作を実行する関数 ippiNormDiff_L2 の種類は、ippcv.h ファイルの中で宣言される。その他のすべての種類の関数は、ippi.h ファイルの中で宣言される。関数 ippiNormDiff_L2 は、2 つのソース・イメージ・バッファ *pSrc1* および *pSrc2* のピクセル値の差から L2 ノルム *pValue*（マスクを使用した操作では *pNorm*）を計算する。
L2 ノルムとは、差の二乗和の平方根で定義される。

$$\text{norm} = \sqrt{\sum |pSrc1 - pSrc2|^2} .$$

計算アルゴリズムは、*hint* 引数で指定する（表 11-2 を参照）。マスクを使用した操作の場合は、マスク値が 0 でないピクセルだけが計算の対象となる。

マルチチャネル・イメージのマスクなし操作の場合（**事例 4、5**）は、対応する各チャネルで個別にノルムを計算し、配列 *value* に格納する。

マルチチャネル・イメージのマスクを使用した操作の場合（**事例 6**）は、*coi* で指定された 1 つの対象チャネルに対してのみノルムを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	次のような場合のエラー状態を示す。 *src1Step* または *src2Step* の値が 0 または負である場合。 マスクを使用した操作で *src1Step*、*src2Step*、または *maskStep* が *roiSize.width* * <pixelSize> より小さい場合。
ippStsNotEvenStepErr	マスクを使用した操作で、浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsCOIErr	*coi* が 1、2、3 以外である場合のエラー状態を示す。

NormRel_Inf

2 つの画像のピクセル値の差から求めた無限大ノルムの相対誤差を計算する。

事例 1：1 チャネル・データの操作

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f* pValue);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R      32f_C1R
```

事例 2：1 チャネル・データのマスクを使用した操作

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize,
      Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1MR     8s_C1MR     32f_C1MR
```

事例 3：マルチチャネル・データの操作

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R     32f_C3R
8u_AC4R     16s_AC4R    32f_AC4R
```

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R     32f_C4R
```

事例 4：マルチチャネル・データのマスクを使用した操作

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize,
      int coi, Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3CMR    8s_C3CMR    32f_C3CMR
```

引数

pSrc1, *pSrc2*	ソース・イメージ・バッファへのポインタ
src1Step, *src2Step*	ソース・イメージ・バッファ内のステップ（バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）

pValue	計算した相対誤差へのポインタ。
value	マルチチャネル・データの場合は、各チャネルで個別に計算した相対誤差が入った配列
coi	（カラー・イメージの場合）対象チャネル。1、2、または 3 のいずれか
pNorm	（マスクを使用した操作の場合）計算した相対ノルムへのポインタ

説明

マスクを使用した操作を実行する関数 ippiNorm_Inf の種類は、ippcv.h ファイルの中で宣言される。その他のすべての種類の関数は、ippi.h ファイルの中で宣言される。関数 ippiNormRel_Inf は、まず 2 つのソース・バッファ *pSrc1* および *pSrc2* のピクセル値の差から無限大ノルムを計算する。無限大ノルムとは、画像内のピクセルの絶対値の最大値で定義される。
出力される相対誤差 *pValue*（マスクを使用した操作では *pNorm*）は、差から求めたノルムを、2 番目のソース・イメージ・バッファ *pSrc2* の無限大ノルムで割ることで求められる。
マスクを使用した操作の場合は、マスク値が 0 でないピクセルだけが計算の対象となる。

マルチチャネル・イメージのマスクなし操作の場合（**事例 3**）は、対応する各チャネルのペアで個別に相対ノルムを計算し、配列 *value* に格納する。

マルチチャネル・イメージのマスクを使用した操作の場合（**事例 4**）は、*coi* で指定された 1 つの対象チャネルに対してのみ相対ノルムを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	次のような場合のエラー状態を示す。 *src1Step* または *src2Step* の値が 0 または負である場合。 マスクを使用した操作で *src1Step*、*src2Step*、または *maskStep* が *roiSize.width* * <pixelSize> より小さい場合。

ippStsNotEvenStepErr	マスクを使用した操作で、浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsCOIErr	*coi* が 1、2、3 以外である場合のエラー状態を示す。
ippStsDivByZero	*pSrc2* の無限大ノルムが 0 の場合の警告を示す。

NormRel_L1

2 つの画像のピクセル値の差から求めた L1 ノルムの相対誤差を計算する。

事例 1：1 チャネルの整数データの操作

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f* pValue);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1R      16s_C1R
```

事例 2：1 チャネルの浮動小数点データの操作

```
IppStatus ippiNormRel_L1_32f_C1R(const Ipp32f* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

事例 3：1 チャネル・データのマスクを使用した操作

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize,
      Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_C1MR     8s_C1MR     32f_C1MR
```

事例 4：マルチチャネルの整数データの操作

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[3]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3R　　16s_C3R

8u_AC4R　　16s_AC4R

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[4]);
```

サポートされる *mod* の値は、次のとおりである。

8u_C4R　　16s_C4R

事例 5：マルチチャネルの浮動小数点データの操作

```
IppStatus ippiNormRel_L1_<mod>(const Ipp32f* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

サポートされる *mod* の値は、次のとおりである。

32f_C3R

32f_AC4R

```
IppStatus ippiNormRel_L1_32f_C4R(const Ipp32f* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

事例 6：マルチチャネル・データのマスクを使用した操作

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
      Ipp64f* norm);
```

サポートされる *mod* の値は、次のとおりである。

8u_C3CMR　　8s_C3CMR　　32f_C3CMR

引数

pSrc1, *pSrc2*	ソース・イメージ・バッファへのポインタ
src1Step, *src2Step*	ソース・イメージ・バッファ内のステップ（バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
coi	（カラー・イメージの場合）対象チャネル。1、2、または 3 のいずれか

pValue	計算した相対誤差へのポインタ。
value	マルチチャネル・データの場合は、各チャネルで個別に計算した相対誤差が入った配列
pNorm	（マスクを使用した操作の場合）計算した相対ノルムへのポインタ
hint	関数の計算アルゴリズムを選択するオプション

説明

マスクを使用した操作を実行する関数 ippiNormRel_L1 の種類は、ippcv.h ファイルの中で宣言される。その他のすべての種類の関数は、ippi.h ファイルの中で宣言される。関数 ippiNormRel_L1 は、まず 2 つのソース・バッファ *pSrc1* および *pSrc2* のピクセル値の差から L1 ノルムを計算する。L1 ノルムとは、画像内のピクセルの絶対値の和で定義される。出力される相対誤差 *pValue* （マスクを使用した操作では *pNorm*）は、差から求めたノルムを、2 番目のソース・イメージ・バッファ *pSrc2* の L1 ノルムで割ることで求められる。計算アルゴリズムは、*hint* 引数で指定する（表 11-2）。マスクを使用した操作の場合は、マスク値が 0 でないピクセルだけが計算の対象となる。

マルチチャネル・イメージのマスクなし操作の場合（**事例 4、5**）は、対応する各チャネルのペアで個別に相対ノルムを計算し、配列 *value* に格納する。

マルチチャネル・イメージのマスクを使用した操作の場合（**事例 6**）は、*coi* で指定された 1 つの対象チャネルに対してのみ相対ノルムを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	次のような場合のエラー状態を示す。 *src1Step* または *src2Step* の値が 0 または負である場合。 マスクを使用した操作で *src1Step*、*src2Step*、または *maskStep* が *roiSize.width* * <pixelSize> より小さい場合。
ippStsNotEvenStepErr	マスクを使用した操作で、浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。

ippStsCOIErr	*coi* が 1、2、3 以外である場合のエラー状態を示す。
ippStsDivByZero	*pSrc2* の L1 ノルムが 0 の場合の警告を示す。

NormRel_L2

2 つの画像のピクセル値の差から求めた L2 ノルムの相対誤差を計算する。

事例 1：1 チャネルの整数データの操作

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f* pValue);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1R      16s_C1R
```

事例 2：1 チャネルの浮動小数点データの操作

```
IppStatus ippiNormRel_L2_32f_C1R(const Ipp32f* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

事例 3：1 チャネル・データのマスクを使用した操作

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize,
      Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1MR     8s_C1MR     32f_C1MR
```

事例 4：マルチチャネルの整数データの操作

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[3]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3R      16s_C3R
8u_AC4R     16s_AC4R
```

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[4]);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C4R      16s_C4R
```

事例 5：マルチチャネルの浮動小数点データの操作

```
IppStatus ippiNormRel_L2_<mod>(const Ipp32f* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiNormRel_L2_32f)C4R(const Ipp32f* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

事例 6：マルチチャネル・データのマスクを使用した操作

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1,
      int src1Step, const Ipp32f* pSrc2, int src2Step,
      const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
      Ipp64f* pNorm);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C3CMR    8s_C3CMR    32f_C3CMR
```

引数

pSrc1, *pSrc2*	ソース・イメージ・バッファへのポインタ
src1Step, *src2Step*	ソース・イメージ・バッファ内のステップ（バイト単位）
pMask	マスク・イメージへのポインタ
maskStep	マスク・イメージ内のステップ（バイト単位）
roiSize	ソース ROI のサイズ（ピクセル単位）
coi	（カラー・イメージの場合）対象チャネル。1、2、または 3 のいずれか
pValue	計算した相対誤差へのポインタ。
value	マルチチャネル・データの場合は、各チャネルで個別に計算した相対誤差が入った配列

pNorm	（マスクを使用した操作の場合）計算した相対ノルムへのポインタ
hint	関数の計算アルゴリズムを選択するオプション

説明

マスクを使用した操作を実行する関数 ippiNormRel_L2 の種類は、ippcv.h ファイルの中で宣言される。その他のすべての種類の関数は、ippi.h ファイルの中で宣言される。関数 ippiNormRel_L2 は、まず 2 つのソース・バッファ *pSrc1* および *pSrc2* のピクセル値の差から L2 ノルムを計算する。L2 ノルムとは、画像内のピクセル値の二乗和の平方根で定義される。出力される相対誤差 *pValue*（マスクを使用した操作では *pNorm*）は、差から求めたノルムを、2 番目のソース・イメージ・バッファ *pSrc2* の L2 ノルムで割ることで求められる。計算アルゴリズムは、*hint* 引数で指定する（表 11-2 を参照）。マスクを使用した操作の場合は、マスク値が 0 でないピクセルだけが計算の対象となる。

マルチチャネル・イメージのマスクなし操作の場合（**事例 4、5**）は、対応する各チャネルのペアで個別に相対ノルムを計算し、配列 *value* に格納する。

マルチチャネル・イメージのマスクを使用した操作の場合（**事例 6**）は、*coi* で指定された 1 つの対象チャネルに対してのみ相対ノルムを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	次のような場合のエラー状態を示す。 *src1Step* または *src2Step* の値が 0 または負である場合。 マスクを使用した操作で *src1Step*、*src2Step*、または *maskStep* が *roiSize.width* * <pixelSize> より小さい場合。
ippStsNotEvenStepErr	マスクを使用した操作で、浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsCOIErr	*coi* が 1、2、3 以外である場合のエラー状態を示す。
ippStsDivByZero	*pSrc2* の L2 ノルムが 0 の場合の警告を示す。

画像の近接性尺度

この項で説明する関数は、画像とテンプレート（他の画像）の間の近接性（類似性）尺度を計算する。これらの関数は、フィーチャ検出関数として使用することも、より高度な手法の構成要素として使用することもできる。

2 つの画像の間の類似性尺度を計算するには、いくつかの方法がある。1 つの方法は、画像とテンプレートのユークリッド距離、すなわち、距離の 2 乗の和（SSD）を計算することである。特定のピクセルの SSD の値が小さいほど、そのピクセルの隣接領域の画像とテンプレートの間に高い類似性が存在する。

行 *r* と列 *c* のピクセルについて、テンプレートと画像の間のユークリッド距離の 2 乗 $S_{tx}(r,c)$ は、次の式から得られる。

$$S_{tx}(r,c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} [t(j,i) - x(r+j-tplRows/2,\ c+i-tplCols/2)]^2$$

$x(r,c)$ は行 *r* と列 *c* の画像ピクセル値、$t(j,i)$ は行 *j* と列 *i* のテンプレート・ピクセル値である。テンプレートのサイズは *tplCols* × *tplRows*、テンプレートの中心の位置は (r,c) である。

もう 1 つの類似性尺度として、相互相関関数がある。特定のピクセルの相互相関が高いほど、そのピクセルの隣接領域の画像とテンプレートの間に高い類似性が存在する。

行 *r* と列 *c* のピクセルについて、テンプレートと画像の間の相互相関 $R_{tx}(r,c)$ は、次の式で計算される。

$$R_{tx}(r,c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t(j,i) \cdot x(r+j-tplRows/2,\ c+i-tplCols/2)$$

この相互相関関数は、画像内の明るさの変化に依存する。この依存関係を避けるには、相互相関関数の代わりに相関係数関数を使用する。この関数は、次のように定義される。

$\bar{t}$ はテンプレートの平均値、$\bar{x}(r,c)$ はテンプレートのすぐ下の領域内の画像の平均値である。

すべてのインテル IPP 近接性関数は、SSD、相互相関、相関係数の**正規化された**値を計算する。これらの値は、次のように定義される。

$$G_{tx}(r,c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} [t(j,i) - \bar{t}\,][x(r+j-tplRows/2, c+i-tplCols/2) - \bar{x}(r,c)]$$

正規化された SSD: $\sigma_{tx}(r, c)$

$$\sigma_{tx}(r,c) = \frac{S_{tx}(r,c)}{\sqrt{R_{xx}(r,c)R_{tt}(tplRows/2,\ tplCols/2)}}$$

正規化された相互相関 : $\rho_{tx}(r, c)$:

$$\rho_{tx}(r,c) = \frac{R_{tx}(r,c)}{\sqrt{R_{xx}(r,c)R_{tt}(tplRows/2,\ tplCols/2)}}$$

R_{xx} は画像の自己相関、R_{tt} はテンプレートの自己相関を示す。

$$R_{xx}(r,c) = \sum_{j=r-(tplRows-1)/2}^{r+(tplRows-1)/2} \sum_{i=c-(tplCols-1)/2}^{c+(tplCols-1)/2} x_{j,i}x_{j,i}$$

$$R_{tt}(tplRows/2, tplCols/2) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t_{j,i}t_{j,i}$$

正規化された相関係数 $\gamma_{tx}(r, c)$:

$$\gamma_{tx}(r,c) = \frac{G_{tx}(r,c)}{\sqrt{G_{xx}(r,c)G_{tt}(tplRows/2,\ tplCols/2)}}$$

G_{xx} は画像の自己相関、G_{tt} はテンプレートの自己相関を示す。これらは一定の明るさ成分を含まない。

$$G_{xx}(r,c) = \sum_{j=r-(tplRows-1)/2}^{r+(tplRows-1)/2} \sum_{i=c-(tplCols-1)/2}^{c+(tplCols-1)/2} (x_{j,i} - \bar{x}(r,c))^2$$

$$G_{tt}(tplRows/2, tplCols/2) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} (t_{j,i} - \bar{t})^2$$

SqrDistanceFull_Norm

画像とテンプレートの間の正規化された
ユークリッド距離を計算する。

事例 1：整数出力を生成する操作

```
IppStatus ippiSqrDistanceFull_Norm_<mod>(const Ipp8u* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
      IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

事例 2：浮動小数点出力を生成するデータ操作

```
IppStatus ippiSqrDistanceFull_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
      int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R       8u32f_C1R     8s32f_C1R
32f_C3R       8u32f_C3R     8s32f_C3R
32f_C4R       8u32f_C4R     8s32f_C4R
32f_AC4R      8u32f_AC4R    8s32f_AC4R
```

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcRoiSize	ソース ROI のサイズ（ピクセル単位）
pTpl	テンプレート・イメージ・バッファへのポインタ
tplStep	テンプレート・イメージ・バッファ内のステップ（バイト単位）
tplRoiSize	テンプレート ROI のサイズ（ピクセル単位）
pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
scaleFactor	整数結果のスケーリング用の係数

説明

関数 ippiSqrDistanceFull_Norm は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージとテンプレート・イメージの間の距離の 2 乗の和 （SSD）を計算して返す。つまり、この関数は、ソース・バッファとテンプレート・バッファ内の各ピクセルについて正規化された SSD 値 $\sigma_{tx}(r, c)$ を計算し、算出した値を出力画像の対応するピクセルに格納する。正規化された SSD 係数を含む結果の行列のサイズは、次のとおりである。

$$(W_s + W_t - 1) * (H_s + H_t - 1),$$

W_s、H_s はソース・イメージの幅と高さを示し、W_t、H_t はテンプレートの幅と高さを示す。

画像ピクセルを一致させるためのテンプレート・アンカは、常にテンプレートの幾何学中心に置かれる （「画像の近接性尺度」の項の $\sigma_{tx}(r, c)$ の公式を参照）。ROI の外側のデータに関する必要条件は存在しない。この関数は、テンプレート・イメージとソース・イメージがゼロでパディングされていることを前提とする。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。

ippStsSizeErr	*srcRoiSize* または *tplRoiSize* のフィールドの値がゼロまたは負である場合、または *srcRoiSize* のフィールドの値が *tplRoiSize* の対応するフィールドの値より小さい場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*tplStep*、または *dstStep* のうち少なくとも 1 つの値がゼロまたは負である場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗したエラー状態を示す。

SqrDistanceSame_Norm

画像とテンプレートの間の正規化された
ユークリッド距離を計算する。

事例 1：整数出力を生成する操作

```
IppStatus ippiSqrDistanceSame_Norm_<mod>(const Ipp8u* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
      IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

事例 2：浮動小数点出力を生成するデータ操作

```
IppStatus ippiSqrDistanceSame_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
      int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

サポートされる *mod* の値は、次のとおりである。

```
32f_C1R      8u32f_C1R     8s32f_C1R
32f_C3R      8u32f_C3R     8s32f_C3R
32f_C4R      8u32f_C4R     8s32f_C4R
32f_AC4R     8u32f_AC4R    8s32f_AC4R
```

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcRoiSize	ソース ROI のサイズ（ピクセル単位）
pTpl	テンプレート・イメージ・バッファへのポインタ
tplStep	テンプレート・イメージ・バッファ内のステップ（バイト単位）
tplRoiSize	テンプレート ROI のサイズ（ピクセル単位）
pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
scaleFactor	整数結果のスケーリング用の係数

説明

関数 ippiSqrDistanceSame_Norm は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージだけに属するピクセルについて正規化された SSD 値 $\sigma_{tx}(r, c)$ を計算する。正規化された SSD 係数を含む結果の行列のサイズは、以下のソース・イメージのサイズと同じになる。

W_s * H_s,

W_s はソース・イメージの幅、H_s はソース・イメージの高さを示す。

画像ピクセルを一致させるためのテンプレート・アンカは、常にテンプレートの幾何学中心に置かれる （「画像の近接性尺度」の項の $\sigma_{tx}(r, c)$ の公式を参照）。ROI の外側のデータに関する必要条件は存在しない。この関数は、テンプレート・イメージとソース・イメージがゼロでパディングされていることを前提とする。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcRoiSize* または *tplRoiSize* のフィールドの値がゼロまたは負である場合、または *srcRoiSize* のフィールドの値が *tplRoiSize* の対応するフィールドの値より小さい場合のエラー状態を示す。

ippStsStepErr	*srcStep*、*tplStep*、または *dstStep* のうち少なくとも 1 つの値がゼロまたは負である場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗したエラー状態を示す。

SqrDistanceValid_Norm

画像とテンプレートの間の正規化された
ユークリッド距離を計算する。

事例 1：整数出力を生成する操作

```
IppStatus ippiSqrDistanceValid_Norm_<mod>(const Ipp8u* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
      IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

事例 2：浮動小数点出力を生成するデータ操作

```
IppStatus ippiSqrDistanceValid_Norm_<mod>(const Ipp<srcDatatype>*
      pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>*
      pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int
      dstStep);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）

srcRoiSize	ソース ROI のサイズ（ピクセル単位）
pTpl	テンプレート・イメージ・バッファへのポインタ
tplStep	テンプレート・イメージ・バッファ内のステップ（バイト単位）
tplRoiSize	テンプレート ROI のサイズ（ピクセル単位）
pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
scaleFactor	整数結果のスケーリング用の係数

説明

関数 ippiSqrDistanceValid_Norm は、ippi.h ファイルの中で宣言される。この関数は、ゼロでパディングされたエッジを持たないソース・イメージのピクセルについて正規化された SSD 値 $\sigma_{tx}(r, c)$ を計算する。正規化された SSD 値を含む結果の行列のサイズは、次のとおりである。

$(W_s - W_t + 1) * (H_s - H_t + 1)$,

W_s、H_s はソース・イメージの幅と高さを示し、W_t、H_t はテンプレートの幅と高さを示す。

画像ピクセルを一致させるためのテンプレート・アンカは、常にテンプレートの幾何学中心に置かれる （「画像の近接性尺度」の項の $\sigma_{tx}(r, c)$ の公式を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcRoiSize* または *tplRoiSize* のフィールドの値がゼロまたは負である場合、または *srcRoiSize* のフィールドの値が *tplRoiSize* の対応するフィールドの値より小さい場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*tplStep*、または *dstStep* のうち少なくとも 1 つの値がゼロまたは負である場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗したエラー状態を示す。

CrossCorrFull_Norm

画像とテンプレートの間の正規化された相
互相関を計算する。

事例 1：整数出力を生成する操作

```
IppStatus ippiCrossCorrFull_Norm_<mod>(const Ipp8u* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl,
      int tplStep, IppiSize tplRoiSize, Ipp8u* pDst,
      int dstStep, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

事例 2：浮動小数点出力を生成するデータ操作

```
IppStatus ippiCrossCorrFull_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
      int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcRoiSize	ソース ROI のサイズ（ピクセル単位）
pTpl	テンプレート・イメージ・バッファへのポインタ
tplStep	テンプレート・イメージ・バッファ内のステップ（バイト単位）
tplRoiSize	テンプレート ROI のサイズ（ピクセル単位）

pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
scaleFactor	整数結果のスケーリング用の係数

説明

関数 ippiCrossCorrFull_Norm は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージとテンプレート・イメージの間の相互相関を計算して返す。つまり、この関数は、ソース・バッファとテンプレート・バッファ内の各ピクセルについて正規化された相互相関値 $\rho_{tx}(r, c)$ を計算し、算出した値を出力画像の対応するピクセルに格納する。正規化された相互相関係数を含む結果の行列のサイズは、次のとおりである。

$(W_s + W_t - 1) * (H_s + H_t - 1)$,

W_s、H_s はソース・イメージの幅と高さを示し、W_t、H_t はテンプレートの幅と高さを示す。

画像ピクセルを一致させるためのテンプレート・アンカは、常にテンプレートの幾何学中心に置かれる （「画像の近接性尺度」の項の $\rho_{tx}(r, c)$ の公式を参照）。
ROI の外側のデータに関する必要条件は存在しない。この関数は、テンプレート・イメージとソース・イメージがゼロでパディングされていることを前提とする。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcRoiSize* または *tplRoiSize* のフィールドの値がゼロまたは負である場合、または *srcRoiSize* のフィールドの値が *tplRoiSize* の対応するフィールドの値より小さい場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*tplStep*、または *dstStep* のうち少なくとも 1 つの値がゼロまたは負である場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗したエラー状態を示す。

CrossCorrSame_Norm

画像とテンプレートの間の正規化された相
互相関を計算する。

事例 1：整数出力を生成する操作

```
IppStatus ippiCrossCorrSame_Norm_<mod>(const Ipp8u* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl,
      int tplStep, IppiSize tplRoiSize, Ipp8u* pDst,
      int dstStep, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

事例 2：浮動小数点出力を生成するデータ操作

```
IppStatus ippiCrossCorrSame_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
      int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcRoiSize	ソース ROI のサイズ（ピクセル単位）
pTpl	テンプレート・イメージ・バッファへのポインタ
tplStep	テンプレート・イメージ・バッファ内のステップ（バイト単位）
tplRoiSize	テンプレート ROI のサイズ（ピクセル単位）

pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
scaleFactor	整数結果のスケーリング用の係数

説明

関数 ippiCrossCorrSame_Norm は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージだけに属するピクセルについて正規化された相互相関値 $\rho_{tx}(r, c)$ を計算する。正規化された相互相関値を含む結果の行列のサイズは、以下のソース・イメージのサイズと同じになる。

$W_s * H_s$,

W_s はソース・イメージの幅、H_s はソース・イメージの高さを示す。

画像ピクセルを一致させるためのテンプレート・アンカは、常にテンプレートの幾何学中心に置かれる （「画像の近接性尺度」の項の $\rho_{tx}(r, c)$ の公式を参照）。ROI の外側のデータに関する必要条件は存在しない。この関数は、テンプレート・イメージとソース・イメージがゼロでパディングされていることを前提とする。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcRoiSize* または *tplRoiSize* のフィールドの値がゼロまたは負である場合、または *srcRoiSize* のフィールドの値が *tplRoiSize* の対応するフィールドの値より小さい場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*tplStep*、または *dstStep* のうち少なくとも 1 つの値がゼロまたは負である場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗したエラー状態を示す。

CrossCorrValid_Norm

画像とテンプレートの間の正規化された相
互相関を計算する。

事例 1：整数出力を生成する操作

```
IppStatus ippiCrossCorrValid_Norm_<mod>(const Ipp8u* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl,
      int tplStep, IppiSize tplRoiSize, Ipp8u* pDst,
      int dstStep, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

事例 2：浮動小数点出力を生成するデータ操作

```
IppStatus ippiCrossCorrValid_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
      int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcRoiSize	ソース ROI のサイズ（ピクセル単位）
pTpl	テンプレート・イメージ・バッファへのポインタ
tplStep	テンプレート・イメージ・バッファ内のステップ（バイト単位）
tplRoiSize	テンプレート ROI のサイズ（ピクセル単位）

pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
scaleFactor	整数結果のスケーリング用の係数

説明

関数 ippiCrossCorrValid_Norm は、ippi.h ファイルの中で宣言される。この関数は、ゼロでパディングされたエッジを持たないソース・イメージのピクセルについて正規化された相互相関値 $\rho_{tx}(r, c)$ を計算する。正規化された相互相関値を含む結果の行列のサイズは、次のとおりである。

$$(W_s - W_t + 1) * (H_s - H_t + 1),$$

W_s、H_s はソース・イメージの幅と高さを示し、W_t、H_t はテンプレートの幅と高さを示す。

画像ピクセルを一致させるためのテンプレート・アンカは、常にテンプレートの幾何学中心に置かれる（「画像の近接性尺度」の項の $\rho_{tx}(r, c)$ の公式を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcRoiSize* または *tplRoiSize* のフィールドの値がゼロまたは負である場合、または *srcRoiSize* のフィールドの値が *tplRoiSize* の対応するフィールドの値より小さい場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*tplStep*、または *dstStep* のうち少なくとも 1 つの値がゼロまたは負である場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗したエラー状態を示す。

CrossCorrFull_NormLevel

画像とテンプレートの間の正規化された相
関係数を計算する。

事例 1：整数出力を生成する操作

```
IppStatus ippiCrossCorrFull_NormLevel_<mod>(const Ipp8u* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl,
      int tplStep, IppiSize tplRoiSize, Ipp8u* pDst,
      int dstStep, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

事例 2：浮動小数点出力を生成するデータ操作

```
IppStatus ippiCrossCorrFull_NormLevel_<mod>(const Ipp<srcDatatype>*
      pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>*
      pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int
      dstStep);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcRoiSize	ソース ROI のサイズ（ピクセル単位）
pTpl	テンプレート・イメージ・バッファへのポインタ
tplStep	テンプレート・イメージ・バッファ内のステップ（バイト単位）

tplRoiSize	テンプレート ROI のサイズ（ピクセル単位）
pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
scaleFactor	整数結果のスケーリング用の係数

説明

関数 ippiCrossCorrFull_NormLevel は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージとテンプレート・イメージの間の相関係数を計算して返す。つまり、この関数は、ソース・バッファとテンプレート・バッファ内の各ピクセルについて正規化された相関係数 $\gamma_{tx}(r, c)$ を計算し、算出した値を出力画像の対応するピクセルに格納する。正規化された相互相関係数を含む結果の行列のサイズは、次のとおりである。

$(W_s + W_t - 1) * (H_s + H_t - 1)$,

W_s、H_s はソース・イメージの幅と高さを示し、W_t、H_t はテンプレートの幅と高さを示す。

画像ピクセルを一致させるためのテンプレート・アンカは、常にテンプレートの幾何学中心に置かれる（「画像の近接性尺度」の項の $\gamma_{tx}(r, c)$ の公式を参照）。ROI の外側のデータに関する必要条件は存在しない。この関数は、テンプレート・イメージとソース・イメージがゼロでパディングされていることを前提とする。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcRoiSize* または *tplRoiSize* のフィールドの値がゼロまたは負である場合、または *srcRoiSize* のフィールドの値が *tplRoiSize* の対応するフィールドの値より小さい場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*tplStep*、または *dstStep* のうち少なくとも 1 つの値がゼロまたは負である場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗したエラー状態を示す。

CrossCorrSame_NormLevel

画像とテンプレートの間の正規化された相
関係数を計算する。

事例 1：整数出力を生成する操作

```
IppStatus ippiCrossCorrSame_NormLevel_<mod>(const Ipp8u* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl,
      int tplStep, IppiSize tplRoiSize, Ipp8u* pDst,
      int dstStep, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

事例 2：浮動小数点出力を生成するデータ操作

```
IppStatus ippiCrossCorrSame_NormLevel_<mod>(const Ipp<srcDatatype>*
      pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>*
      pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int
      dstStep);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcRoiSize	ソース ROI のサイズ（ピクセル単位）
pTpl	テンプレート・イメージ・バッファへのポインタ
tplStep	テンプレート・イメージ・バッファ内のステップ（バイト単位）

tplRoiSize	テンプレート ROI のサイズ（ピクセル単位）
pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
scaleFactor	整数結果のスケーリング用の係数

説明

関数 ippiCrossCorrSame_NormLevel は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージだけに属するピクセルについて正規化された相関係数 $\gamma_{tx}(r, c)$ を計算する。正規化された相関係数を含む結果の行列のサイズは、以下のソース・イメージのサイズと同じになる。

W_s * H_s,

W_s はソース・イメージの幅、H_s はソース・イメージの高さを示す。

画像ピクセルを一致させるためのテンプレート・アンカは、常にテンプレートの幾何学中心に置かれる （「画像の近接性尺度」の項の $\gamma_{tx}(r, c)$ の公式を参照）。ROI の外側のデータに関する必要条件は存在しない。この関数は、テンプレート・イメージとソース・イメージがゼロでパディングされていることを前提とする。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcRoiSize* または *tplRoiSize* のフィールドの値がゼロまたは負である場合、または *srcRoiSize* のフィールドの値が *tplRoiSize* の対応するフィールドの値より小さい場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*tplStep*、または *dstStep* のうち少なくとも 1 つの値がゼロまたは負である場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗したエラー状態を示す。

CrossCorrValid_NormLevel

画像とテンプレートの間の正規化された相
互相関を計算する。

事例 1：整数出力を生成する操作

```
IppStatus ippiCrossCorrValid_NormLevel_<mod>(const Ipp8u* pSrc,
      int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl,
      int tplStep, IppiSize tplRoiSize, Ipp8u* pDst,
      int dstStep, int scaleFactor);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

事例 2：浮動小数点出力を生成するデータ操作

```
IppStatus ippiCrossCorrValid_NormLevel_<mod>(const Ipp<srcDatatype>*
      pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>*
      pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int
      dstStep);
```

サポートされる *mod* の値は、次のとおりである。

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

引数

pSrc	ソース・イメージ・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcRoiSize	ソース ROI のサイズ（ピクセル単位）
pTpl	テンプレート・イメージ・バッファへのポインタ
tplStep	テンプレート・イメージ・バッファ内のステップ（バイト単位）

tplRoiSize	テンプレート ROI のサイズ（ピクセル単位）
pDst	デスティネーション・イメージ・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
scaleFactor	整数結果のスケーリング用の係数

説明

関数 ippiCrossCorrValid_NormLevel は、ippi.h ファイルの中で宣言される。この関数は、ゼロでパディングされたエッジを持たないソース・イメージのピクセルについて正規化された相関係数 $\gamma_{tx}(r, c)$ を計算する。正規化された相関係数を含む結果の行列のサイズは、次のとおりである。

$(W_s - W_t + 1) * (H_s - H_t + 1)$,

W_s、H_s はソース・イメージの幅と高さを示し、W_t、H_t はテンプレートの幅と高さを示す。

画像ピクセルを一致させるためのテンプレート・アンカは、常にテンプレートの幾何学中心に置かれる （「画像の近接性尺度」の項の $\gamma_{tx}(r, c)$ の公式を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcRoiSize* または *tplRoiSize* のフィールドの値がゼロまたは負である場合、または *srcRoiSize* のフィールドの値が *tplRoiSize* の対応するフィールドの値より小さい場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*tplStep*、または *dstStep* のうち少なくとも 1 つの値がゼロまたは負である場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗したエラー状態を示す。

12 画像のジオメトリ変換

本章では、画像のサイズ変更、回転、ワープ、再マッピングのようなジオメトリ演算を行うインテル® IPP 関数について説明する。

表 12-1 に、画像のジオメトリ変換を行う IPP 関数の一覧を示す。

表 12-1　画像のジオメトリ変換関数

関数の基本名	操作
Resize	画像のサイズを変更する。
ResizeCenter	画像のサイズを変更して、シフトする。
GetResizeFract	タイル画像のサイズ変更係数を再計算する。
ResizeShift	画像タイルのサイズを変更する。
Mirror	画像をミラーリングする。
Remap	画像内のピクセルに対して、任意の再マッピングを実行する。
Rotate	画像を原点（0,0）の回りに回転し、シフトする。
GetRotateShift	回転中心と角度を指定して、ippiRotate のためのシフト値を計算する。
AddRotateShift	指定したシフトを使用して、指定した中心の回りに画像を回転するためのシフト値を計算する。
GetRotateQuad	画像の ROI を ippiRotate 関数で変換した結果の四角形を計算する。
GetRotateBound	画像の ROI を ippiRotate 関数で変換した場合の境界矩形を計算する。
RotateCenter	画像を任意の回転中心回りに回転する。
Shear	画像のシアー変換を実行する。
GetShearQuad	画像の ROI を ippiShear 関数で変換した結果の四角形を計算する。
GetShearBound	画像の ROI を ippiShear 関数で変換した場合の境界矩形を計算する。
WarpAffine	アフィン変換を使用して、画像をワープする。
WarpAffineBack	画像の逆アフィン変換を実行する。
WarpAffineQuad	指定のソース四角形を指定のデスティネーション四角形に変換するようなアフィン変換を実行する。
GetAffineQuad	画像の ROI を ippiWarpAffine 関数で変換した結果の四角形を計算する。
GetAffineBound	画像の ROI を ippiWarpAffine 関数で変換した場合の境界矩形を計算する。像を再サンプリングす
GetAffineTransform	アフィン変換係数を計算する。

表 12-1　画像のジオメトリ変換関数　（続き）

関数の基本名	操作
WarpPerspective	透視変換を使用して、画像をワープする。
WarpPerspectiveBack	画像の逆透視変換を実行する。
WarpPerspectiveQuad	指定のソース四角形を指定のデスティネーション四角形に変換するような透視変換を実行する。
GetPerspectiveQuad	画像の ROI を ippiWarpPerspective 関数で変換した結果の四角形を計算する。
GetPerspectiveBound	画像の ROI を ippiWarpPerspective 関数で変換した場合の境界矩形を計算する。
GetPerspectiveTransform	透視変換係数を計算する。
WarpBilinear	バイリニア変換を使用して、画像をワープする。
WarpBilinearBack	画像の逆バイリニア変換を実行する。
WarpBilinearQuad	指定のソース四角形を指定のデスティネーション四角形に変換するようなバイリニア変換を実行する。
GetBilinearQuad	画像の ROI を ippiWarpBilinear 関数で変換した結果の四角形を計算する。
GetBilinearBound	画像の ROI を ippiWarpBilinear 関数で変換した場合の境界矩形を計算する。
GetBilinearTransform	バイリニア変換係数を計算する。

画像のジオメトリ変換を行う関数は、画像を再サンプリングする何らかの補間アルゴリズムを使用する。使用する補間アルゴリズムは、関数の *interpolation* パラメータで指定する。指定できる補間アルゴリズムは、直近サンプル、リニア補間、または 3 次たたみ込みの 3 つで、これにオプションのエッジ平滑化を組み合わせることができる。直近サンプルは、最も高速なアルゴリズムである。これに対して、リニア補間と 3 次たたみ込みは、精度は高いが低速である。
補間アルゴリズムは、次の定数のいずれかで指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次補間
IPPI_SMOOTH_EDGE	補間とエッジ平滑化の組み合わせ
IPPI_INTER_SUPER	スーパー・サンプリング補間

関数によっては、補間アルゴリズムと、元の画像の境界線を変換したエッジの平滑化（アンチエイリアシング）とを組み合わせて指定できる。エッジの平滑化を使用するには、パラメータ *interpolation* に、目的の補間アルゴリズムの値 IPPI_SMOOTH_EDGE をビット単位で OR 演算をした値を指定する。
補間アルゴリズムとエッジの平滑化を組み合わせて指定できるのは、このオプションが引数の説明のところに明示されている関数のみである。

ジオメトリ変換における ROI の処理

本章で説明するすべての変換関数は、ソース・イメージとデスティネーション・イメージで定義された矩形の処理対象領域（ROI）を処理する。
ジオメトリ変換関数による ROI の処理方法は、他の関数とは異なる（第 2 章の Regions of Interest in IPP を参照）。最大の相違点は、変換後のソース ROI と、デスティネーション ROI の交差領域だけが処理される点である。すなわち、逆ワープ関数を除くすべてのジオメトリ変換関数は、次の手順で ROI を処理する。

- ソース・イメージの矩形 ROI を、デスティネーション・イメージ内の四角形に変換する。
- この四角形と、デスティネーション・イメージの矩形 ROI との交差領域を検出する。
- デスティネーション・イメージ内の交差領域のみを更新する。

ソース・イメージとデスティネーション・イメージの座標原点は同じでなければならない。

ROI を使用する関数では、ソース・イメージの各スキャンラインを 0 でパディングしてアライメントできるので、実際のスキャンラインのバイト数を、画像の幅から想定されるバイト数より大きくできる。その場合、画像データの各行のサイズは、*srcStep* パラメータの値で決まる。このパラメータ値には、画像の各行の開始位置の間隔をバイト数で指定する。

矩形 ROI を完全に記述するには、原点（左上隅の座標）とサイズが必要である。ジオメトリ変換関数では、ソース・イメージ ROI を、`IppiRect` 型の *srcROI* パラメータで指定する。したがって、ソース・イメージの矩形 ROI は、4 つの値で完全に決定される。
これに対して、デスティネーション・イメージ、`IppiRect` 型の *dstROI* パラメータで指定したり、`IppiSize` 型の *dstRoiSize* パラメータで指定したりする。後者の場合、デスティネーション ROI のサイズだけが渡され、原点は *pDst* ポインタで参照される。

Resize

画像のサイズを変更する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiResize_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
      double xFactor, double yFactor, int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiResize_<mod>(const Ipp<datatype>* const pSrc[3],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[3], int dstStep,
      IppiSize dstRoiSize, double xFactor, double yFactor,
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	16u_P3R	32f_P3R

```
IppStatus ippiResize_<mod>(const Ipp<datatype>* const pSrc[4],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[4], int dstStep,
      IppiSize dstRoiSize, double xFactor, double yFactor,
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R	16u_P4R	32f_P4R

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）

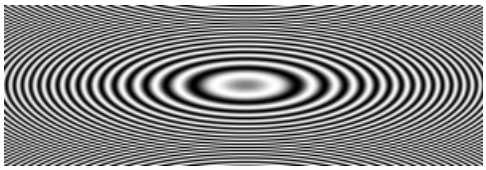
srcROI	ソース・イメージ内の ROI（IppiRect タイプ）
pDst	デスティネーション ROI バッファへのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	デスティネーション ROI のサイズ（ピクセル単位）
xFactor, yFactor	ソース ROI を *x* および *y* 方向にサイズ変更する比率。1 より大きければ、その方向に画像のサイズが拡大される。
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次補間
IPPI_INTER_SUPER	スーパーサンプリング補間

説明

関数 ippiResize は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ ROI のサイズを、*x* 方向に *xFactor*、*y* 方向に *yFactor* の比率で変更する。画像のサイズは、*xFactor*、*yFactor* の値に応じて、それぞれの方向に拡大または縮小される。結果は、*interpolation* パラメータで指定された補間方法で再サンプリングされ、デスティネーション・イメージの ROI に書き込まれる。
図 12-1 は、ippiResize 関数を、サイズが 256 × 256 ピクセルのサンプルイメージに適用した結果を示している。デスティネーション・イメージのサイズは、300 × 100 ピクセルである。

図 12-1　ippiResize 関数でサイズ変更したサンプル・イメージ



次のコード例は、ippiResize 関数の使用方法を示している。

例 12-1　例 12-1 画像のサイズ変更

```
IppStatus resize( void ) {
   Ipp8u x[8*8], y[8*8];
   IppiSize srcsz={8,8}, dstroi={6,6};
   IppiRect srcroi={1,1,6,6};
   IppiSize roi = {8,8};
 int i;
   for( i=0; i<8; ++i ) {
      ippiSet_8u_C1R( (Ipp8u)i, x+8*i+i, 8, roi );
      --roi.width;
      --roi.height;
   }
  return ippiResize_8u_C1R( x, srcsz, 8, srcroi, y, 8,
dstroi, 2.9, 2, IPPI_INTER_NN );
}
```

サイズ、変更後の画像には次のデータが含まれる。

```
01 01 01 01 01 01 CC CC
01 01 01 01 01 01 CC CC
01 01 02 02 02 02 CC CC
01 01 02 02 02 02 CC CC
01 01 02 02 02 03 CC CC
01 01 02 02 02 03 CC CC
CC CC CC CC CC CC CC CC
CC CC CC CC CC CC CC CC
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。

ippStsSizeErr	*srcSize* または *dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsResizeFactorErr	*xFactor* または *yFactor* の値が 0 より小さいか 0 である場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

ResizeCenter

画像のサイズを変更し、指定のポイントが固定されるようにデスティネーション・イメージをシフトする。

事例 1：ピクセル順序データの操作

```
IppStatus ippiResizeCenter_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
      double xFactor, double yFactor, double xCenter, double yCenter,
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiResizeCenter_<mod>(const Ipp<datatype>* const pSrc[3],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[3], int dstStep,
      IppiSize dstRoiSize, double xFactor, double yFactor,
      double xCenter, double yCenter, int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	16u_P3R	32f_P3R

```
IppStatus ippiResizeCenter_<mod>(const Ipp<datatype>* const pSrc[4],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[4], int dstStep,
      IppiSize dstRoiSize, double xFactor, double yFactor,
      double xCenter, double yCenter, int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_P4R      16u_P4R      32f_P4R
```

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各カラー・プレーンへの 3 つまたは 4 つのポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージ内の ROI（IppiRect 型）
pDst	デスティネーション ROI バッファへのポインタ。プレーン順序データの場合は、各カラー・プレーンへの 3 つまたは 4 つのポインタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	デスティネーション ROI のサイズ（ピクセル単位）
xFactor, yFactor	ソース ROI を *x* および *y* 方向にサイズ変更する比率。1 より大きければ、その方向に画像のサイズが拡大される。
xCenter, yCenter	画像がサイズ変更された後もシフトしないポイントの座標
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次補間
IPPI_INTER_SUPER	スーパーサンプリング補間

説明

関数 ippiResizeCenter は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ ROI のサイズを、*x* 方向に *xFactor*、*y* 方向に *yFactor*m の比率で変更する。画像のサイズは、*xFactor*、*yFactor* の値に応じて、それぞれの方向に拡大または縮小される。

ippiResize　関数とは異なり、この関数は、サイズ変更後も指定のポイント (*xCenter*、*yCenter*) が移動しないようにデスティネーション・イメージをシフトする。そのため、指定のポイントを固定してサイズ変更を行ったような効果がある。結果は、*interpolation* パラメータで指定された補間方法で再サンプリングされ、デスティネーション・イメージの ROI に書き込まれる。

次のコード例は、ippiResizeCenter 関数の使用方法を示している。

例 12-2　画像のサイズ変更とシフト

```
IppStatus resizeCenter( void ) {
   Ipp8u x[8*8], y[8*8];
   IppiSize sz = {8,8}, roi = {8,8};
   IppiRect rect = {0,0,8,8};
   int i;
   for( i=0; i<8; ++i ) {
      ippiSet_8u_C1R( (Ipp8u)i, x+8*i+i, 8, roi );
      --roi.width;
      --roi.height;
   }
  return ippiResizeCenter_8u_C1R( x, sz, 8, rect, y, 8, sz,
      2.9, 2.0, 4, 4, IPPI_INTER_NN );
}
```

ippiResizeCenter 関数を適用した画像には、次のデータが含まれる。

```
02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 02
03 03 03 03 03 03 03 03
03 03 03 03 03 03 03 03
03 03 04 04 04 04 04 04
03 03 04 04 04 04 04 04
03 03 04 04 04 04 05 05
03 03 04 04 04 04 05 05
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcSize* または *dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsResizeFactorErr	*xFactor* または *yFactor* の値が 0 より小さいか 0 である場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

GetResizeFract

タイル画像のサイズ変更係数を再計算する。

```
IppStatus ippiGetResizeFract(IppiSize srcSize, IppiRect srcROI,
     double xFactor, double yFactor, double* xFr, double* yFr,
     int interpolation);
```

引数

srcSize	ソース・イメージのサイズ（ピクセル単位）
srcROI	ソース・イメージ内の ROI（IppiRect 型）
pDst	デスティネーション ROI バッファへのポインタ。プレーン順序データの場合は、各カラー・プレーンへの 3 つまたは 4 つのポインタが入った配列
xFactor, yFactor	ソース ROI を *x* および *y* 方向にサイズ変更する比率。1 より大きければ、その方向に画像のサイズが拡大される。
xFr, yFr	再計算されたサイズ変更係数へのポインタ。これらの値は、ippiResizeShift 関数に渡され、画像タイルのサイズ変更に使用される。1 より大きければ、その方向に画像のサイズが拡大される。

interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。
	IPPI_INTER_NN　直近サンプル補間
	IPPI_INTER_LINEAR　リニア補間
	IPPI_INTER_CUBIC　3 次補間

説明

関数 ippiResizeFract は、ippi.h ファイルの中で宣言される。この関数は、タイル・ソース・イメージのサイズを、*x* 方向に *xFactor*、*y* 方向に *yFactor* の比率で変更する場合に使用される。この関数は、ippiREsizeShift 関数に渡されるサイズ変更係数 *xFr* と *yFr* の新しい値を計算する。ippiREsizeShift 関数は、個々のタイルのサイズ変更操作を実行する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsSizeErr	*srcSize* フィールドの値が 0 または負の場合のエラー状態を示す
ippStsResizeFactorErr	*xFactor* または *yFactor* の値が 0 より小さいか 0 である場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

ResizeShift

画像タイルのサイズを変更する

事例 1：ピクセル順序データの操作

```
IppStatus ippiResizeShift_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
      double xFr, double yFr, double xShift, double yShift,
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiResizeShift_<mod>(const Ipp<datatype>* const pSrc[3],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[3], int dstStep,
      IppiSize dstRoiSize, double xFr, double yFr,
      double xShift, double yShift, int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	16u_P3R	32f_P3R

```
IppStatus ippiResizeShift_<mod>(const Ipp<datatype>* const pSrc[4],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[4], int dstStep,
      IppiSize dstRoiSize, double xFr, double yFr,
      double xShift, double yShift, int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R	16u_P4R	32f_P4R

引数

pSrc	ソース・イメージへのポインタ。プレーン順序データの場合は、各カラー・プレーンへの 3 つまたは 4 つのポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージ内の ROI（IppiRect 型）
pDst	デスティネーション ROI バッファへのポインタ。プレーン順序データの場合は、各カラー・プレーンへの 3 つまたは 4 つのポインタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	デスティネーション ROI のサイズ（ピクセル単位）
xFr, yFr	*xFactor*、*yFactor* によるソース・イメージ ROI のサイズ変更に使用される係数

xShift, yShift	処理対象領域（*srcRect*）の位置の補正値
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。 IPPI_INTER_NN 直近サンプル補間 IPPI_INTER_LINEAR リニア補間 IPPI_INTER_CUBIC 3 次補間

説明

関数 ippiResizeShift は、ippi.h ファイルの中で宣言される。この関数は、タイル・ソース・イメージのサイズを、*x* 方向に *xFactor*、*y* 方向に *yFactor* の比率で変更する場合に使用される。この関数を使用する前に、ippiGetResizeFract 関数を呼び出して、*xFr* 係数と *yFr* 係数の値を計算する必要がある。画像のサイズは、*xFactor*、*yFactor* の値に応じて、それぞれの方向に拡大または縮小される。*xFr* 係数または *yFr* 係数の値が 1 より大きい場合は、その方向に画像のサイズが縮小され、1 より小さい場合は、その方向に画像のサイズが拡大される。結果は、*interpolation* パラメータで指定された補間方法で再サンプリングされ、デスティネーション・イメージの ROI に書き込まれる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcSize* または *dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsResizeFactorErr	*xFr* または *yFr* 値が 0 より小さいか 0 である場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

Mirror

画像を水平軸、垂直軸、または両軸に対してミラーリングする

事例 1：非インプレース操作

```
IppStatus ippiMirror_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
      IppiAxis flip);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

事例 2：インプレース操作

```
IppStatus ippiMirror_<mod>(const Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, IppiAxis flip);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

引数

pSrc	ソース・バッファへのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pDst	デスティネーション・バッファへのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタ

srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）
flip	画像をミラーリングする軸を、次のいずれかの値で指定する。

axsHorizontal,	水平軸
axsVertical,	垂直軸
axsBoth,	水平軸と垂直軸の両方

説明

関数 ippiMirror は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージ *pSrc* を、*flip* 値に従って水平軸、垂直軸、または両軸に対してミラーリングし、その結果をデスティネーション・イメージ *pDst* に書き込む。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep*、*dstStep*、または *srcDstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsMirrorFlipErr	*flip* の値が無効である場合のエラー状態を示す。

Remap

ソース・イメージのピクセルにルックアップ座標マッピングを適用する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiRemap_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      const Ipp32f* pxMap, int xMapStep, const Ipp32f* pyMap,
      int yMapStep, Ipp<datatype>* pDst, int dstStep, IppiSize
      dstRoiSize, int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiRemap_<mod>(const Ipp<datatype>* const pSrc[3],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      const Ipp32f* pxMap, int xMapStep, const Ipp32f* pyMap,
      int yMapStep, Ipp<datatype>* const pDst[3], int dstStep,
      IppiSize dstRoiSize, int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	32f_P3R

```
IppStatus ippiRemap_<mod>(const Ipp<datatype>* const pSrc[4],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      const Ipp32f* pxMap, int xMapStep, const Ipp32f* pyMap,
      int yMapStep, Ipp<datatype>* const pDst[4], int dstStep,
      IppiSize dstRoiSize, int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R	32f_P4R

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列

srcSize	ソース・イメージのサイズ（ピクセル単位）
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージ内の ROI（IppiRect 型）
pxMap, pyMap	*x* 座標と *y* 座標のテーブルが入った 2D バッファの先頭へのポインタ。参照した座標がソース ROI の外部のピクセルに対応する場合は、ソース・ピクセルのマッピングは行われない。
xMapStep, yMapStep	*x* 座標と *y* 座標のテーブルが入ったバッファ内のステップ（バイト数単位）
pDst	デスティネーション・バッファへのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	デスティネーション ROI のサイズ（ピクセル単位）
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPP_INTER_NN	直近サンプル補間
IPP_INTER_LINEAR	リニア補間
IPP_INTER_CUBIC	3 次補間

説明

関数 ippiRemap は、ippi.h ファイルの中で宣言される。この関数は、ピクセルを再マッピングすることで、ソース・イメージを変換する。ピクセルの再マッピングでは、*pxMap* と *pyMap* の 2 つのバッファを使用して、目的のデスティネーション・イメージ・ピクセルに書き込むソース・イメージ・ピクセルの座標を探す。使用するルックアップ・テーブルは、アプリケーションで提供する必要がある。
ソース・ピクセルからデスティネーション・ピクセルへの再マッピングは、次の公式に従って行われる。

```
dst_pixel[i,j] = src_pixel[pxMap[i,j], pyMap[i,j]]
```

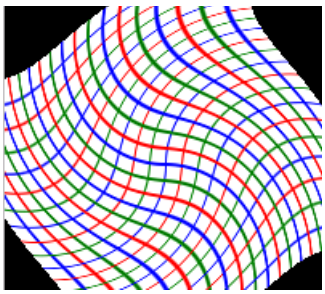
i、*j* は、目的のデスティネーション・イメージ・ピクセル *dst_pixel* の *x* 座標と *y* 座標である。

pxMap[*i*,*j*] には、*dst_pixel* に書き込むソース・イメージ・ピクセル *src_pixel* の *x* 座標が入っている。

pyMap[*i*,*j*] には、*dst_pixel* に書き込むソース・イメージ・ピクセル *src_pixel* の *y* 座標が入っている。

図 12-2 は、青、赤、緑のラインが交互に繰り返す四角形グリッドのサンプル・イメージに、関数 ippiRemap を適用した例を示している。

図 12-2　サンプル・イメージの再マッピング



画像の変換部分は、*interpolation* パラメータで指定された補間方法を使用して再サンプリングされ、デスティネーション・イメージの ROI に書き込まれる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcSize* または *dstRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

Rotate

画像を原点（0,0）の回りに回転し、シフトする。

事例 1：ピクセル順序データの操作

```
IppStatus ippiRotate_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* pDst, int dstStep, IppiRect dstROI,
      double angle, double xShift, double yShift, int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiRotate_<mod>(const Ipp<datatype>* const pSrc[3],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[3], int dstStep,
      IppiRect dstROI, double angle, double xShift, double yShift,
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	16u_P3R	32f_P3R

```
IppStatus ippiRotate_<mod>(const Ipp<datatype>* const pSrc[4],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[4], int dstStep,
      IppiRect dstROI, double angle, double xShift, double yShift,
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R	16u_P4R	32f_P4R

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）

srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージの ROI
pDst	デスティネーション・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別インタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstROI	デスティネーション・イメージの ROI
angle	回転角を度数で指定する。ソース・イメージは、原点（0,0）を中心に時計回りに回転する。
xShift, yShift	回転後に行われる、水平軸と垂直軸方向のシフト量
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次たたみ込み補間
IPPI_SMOOTH_EDGE	上記の補間方法とともにエッジ平滑化オプションを使用する。

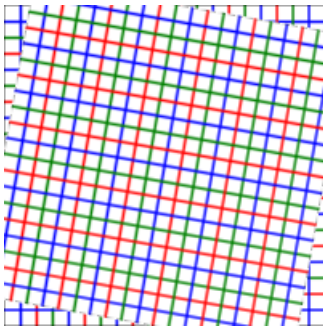
説明

関数 ippiRotate は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージの ROI を原点（0,0）の回りに *angle* 度回転し（*angle* 値が正ならば時計回り）、*x* 軸および *y* 軸方向に *xShift* および *yShift* 値だけシフトする。結果は、*interpolation* パラメータで指定された補間方法で再サンプリングされ、デスティネーション・イメージの ROI に書き込まれる。

任意の回転中心（*xCenter*、*yCenter*）回りに回転する場合は、関数 ippiGetRotateShift を使用して、適切な *xShift*、*yShift* 値を計算し、それを入力パラメータとして ippiRotate を呼び出す。あるいは、ippiRotateCenter 関数を使用する方法もある。

図 12-3 は、サンプル・イメージを 10 度回転した結果を示している。

図 12-3　画像の回転



SMOOTH_EDGE オプションがデスティネーション・イメージの外観に与える効果を下図に示す。どちらの画像も、同じ補間方法を使用してソース・イメージを 9 度回転しているが、SMOOTH_EDGE オプションがオンとオフになっている。このオプションを使用したときのアンチエイリアシング効果が、画像 b の右側の境界線に現れている。

図 12-4　デスティネーション・イメージに見られるエッジ平滑化の効果

a) SMOOTH_EDGE オプションがオフ

b) SMOOTH_EDGE オプションがオン

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	画像のサイズのいずれかの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

GetRotateShift

画像を指定した中心回りに回転するときのシフト値を計算する。

```
IppStatus ippiGetRotateShift(double xCenter, double yCenter,
    double angle, double *xShift, double *yShift)
```

引数

xCenter, yCenter	指定の回転中心座標
angle	画像を座標（*xCenter*、*yCenter*）を中心に時計回りに回転する度数
xShift, yShift	計算された水平軸および垂直軸方向のシフト値へのポインタ。これらのシフト値を ippiRotate 関数に渡すことにより、指定の回転中心 （*xCenter*、*yCenter*）回りの回転が得られる。

説明

関数 ippiGetRotateShift は、ippi.h ファイルの中で宣言される。画像を原点 (0,0) 回りではなく任意の回転中心（*xCenter*、*yCenter*）回りに回転する必要がある場合は、この関数を使用する。この関数で計算されたシフト値 *xShift*、*yShift* を、ippiRotate 関数に渡すことにより、（*xCenter*、*yCenter*）回りの回転が得られる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値はエラーを示す。
ippStsNullPtrErr	*xShift* または *yShift* ポインタが NULL の場合のエラー状態を示す。

AddRotateShift

指定したシフトを使用して、指定した中心の回りに画像を回転するためのシフト値を計算する。

```
IppStatus ippiAddRotateShift(double xCenter, double yCenter,
    double angle, double *xShift, double *yShift)
```

引数

xCenter, yCenter	指定の回転中心座標
angle	画像を座標（*xCenter*、*yCenter*）を中心に時計回りに回転する度数
xShift, yShift	水平軸方向と垂直軸方向の修正されたシフト値へのポインタ。これらの新しいシフト値は、ippiRotate 関数に渡され、（*xCenter*, *yCenter*）の回りの回転とシフトの実行に使用される。

説明

関数 ippiAddRotateShift は、ippi.h ファイルの中で宣言される。この関数は、必要なシフトを使用して、原点（0,0）以外の任意の中心（*xCenter*, *yCenter*）の回りに画像を回転する場合に使用される。この関数によって算出されたシフト値 *xShift*、*yShift* は、ippiRotate 関数に渡され、（*xCenter*, *yCenter*）の回りの回転とシフトの実行に使用される。これらのシフト値は、初期化されていなければならない。例えば、シフト値 （30.3, 26.2）を使用して、点（xCenter, yCenter）の回りに角度 *angle* だけ画像を回転するには、次のコードを記述する必要がある。

```
xShift = 30.3
yShift = 26.2
ippiAddRotateShift(xCenter,yCenter,angle,&xShift,&yShift);
ippiRotate(angle,xShift,yShift);
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値はエラーを示す。
ippStsNullPtrErr	*xShift* または *yShift* ポインタが NULL の場合のエラー状態を示す。

GetRotateQuad

ソース ROI 矩形を ippiRotate 関数で
マッピングした結果の四角形の頂点座標を
計算する。

```
IppStatus ippiGetRotateQuad(IppiRect roi, double quad[4][2],
    double angle, double xShift, double yShift);
```

引数

roi	ソース・イメージの ROI
quad	出力配列。この配列には、ソース *roi* を ippiRotate 関数でマッピングした結果の四角形の頂点座標が入る。
angle	回転角 （度数単位）
xShift, yShift	回転後に行われる、水平軸と垂直軸方向のシフト量

説明

関数 ippiGetRotateQuad は、ippi.h ファイルの中で宣言される。この関数は、ippiRotate のサポート関数として使用する。この関数は、画像を *angle* 度回転し、*xShift*、*yShift* だけシフトする ippiRotate 関数によって、ソース ROI をマッピングした結果の四角形の頂点座標を計算する。

配列 *quad*[4][2] の第 1 次元 [4] は頂点の数に等しい。第 2 次元 [2] は頂点の x 座標と y 座標を意味する。四角形の各頂点は以下の意味を持つ。

quad[0] は、ソース ROI の座標変換された左上隅に対応する。
quad[1] は、ソース ROI の座標変換された右上隅に対応する。
quad[2] は、ソース ROI の座標変換された右下隅に対応する。
quad[3] は、ソース ROI の座標変換された左下隅に対応する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値はエラーを示す。

GetRotateBound

ソース ROI を *ippiRotate* 関数で変換した場合の境界矩形を計算する。

```
IppStatus ippiGetRotateBound(IppiRect roi, double bound[2][2],
    double angle, double xShift, double yShift);
```

引数

roi	ソース・イメージの ROI
bound	出力配列。この配列には、ソース *ROI* を変換した場合の境界矩形の頂点座標が入る。
angle	回転角 （度数単位）
xShift, yShift	回転後に行われる、水平軸と垂直軸方向のシフト量

説明

関数 ippiGetRotateBound は、ippi.h ファイルの中で宣言される。この関数は、ippiRotate のサポート関数として使用する。この関数は、画像を *angle* 度回転し、*xShift*、*yShift* だけシフトする ippiRotate 関数によって、ソース ROI をマッピングした結果の四角形 *quad* の最小境界矩形の頂点座標を計算する。*bound*[0] は左上隅の x, y 座標を指定し、*bound*[1] は右下隅の x, y 座標を指定する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

RotateCenter

画像を任意の回転中心回りに回転する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiRotateCenter_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* pDst, int dstStep, IppiRect dstROI,
      double angle, double xCenter, double yCenter, int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiRotateCenter_<mod>(const Ipp<datatype>* const pSrc[3],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[3], int dstStep,
      IppiRect dstROI, double angle, double xCenter, double yCenter,
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	16u_P3R	32f_P3R

```
IppStatus ippiRotateCenter_<mod>(const Ipp<datatype>* const pSrc[4],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[4], int dstStep,
      IppiRect dstROI, double angle, double xCenter, double yCenter,
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R	16u_P4R	32f_P4R

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）

srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージの ROI
pDst	デスティネーション・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別インタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstROI	デスティネーション・イメージの ROI
angle	回転角 （度数単位）
xCenter, yCenter	回転中心の座標
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次たたみ込み補間
IPPI_SMOOTH_EDGE	上記の補間方法とともにエッジ平滑化オプションを使用する。

説明

関数 ippiRotateCenter は、ippi.h ファイルの中で宣言される。この関数は、ソース・イメージの ROI を座標（*xCenter*、*yCenter*）の回りに *angle* 度回転（*angle* 値が正ならば時計回り）する。結果は、*interpolation* パラメータで指定された補間方法で再サンプリングされ、デスティネーション・イメージの ROI に書き込まれる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	画像のサイズのいずれかの値が 0 または負の場合のエラー状態を示す。

ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

Shear

ソース・イメージのシアー変換を実行する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiShear_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* pDst, int dstStep, IppiRect dstROI,
      double xShear, double yShear, double xShift, double yShift,
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiShear_<mod>(const Ipp<datatype>* const pSrc[3],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstROI,
      double xShear, double yShear, double xShift, double yShift,
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	32f_P3R

```
IppStatus ippiShear_<mod>(const Ipp<datatype>* const pSrc[4],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstROI,
      double xShear, double yShear, double xShift, double yShift,
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R	32f_P3R

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージの ROI
pDst	デスティネーション・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別インタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstROI	デスティネーション・イメージの ROI
xShear, yShear	シアー変換係数
xShift, yShift	水平軸および垂直軸方向のシフト値
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次たたみ込み補間
IPPI_SMOOTH_EDGE	上記の補間方法とともにエッジ平滑化オプションを使用する。

説明

関数 ippiShear は、ippi.h ファイルの中で宣言される。この関数で実行されるシアー変換は、次の公式に従って、ソース・イメージのピクセル座標（*x*,*y*）をマッピングする。

$$x' = x + xShear * y + xShift$$

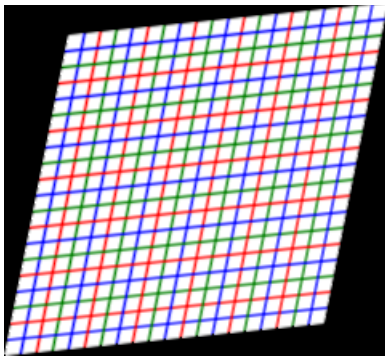
$$y' = yShear * x + y + yShift$$

x' と y' は、シアー変換後の画像のピクセル座標を表す。シアー変換は、ippiWarpAffine 関数で実行されるアフィン変換の特殊な場合である。

画像の変換部分は、*interpolation* パラメータで指定された補間方法を使用して再サンプリングされ、デスティネーション・イメージの ROI に書き込まれる。

図 12-5 は、サンプル・イメージにシアー変換関数 ippiShear を適用した例を示している。

図 12-5　サンプル・イメージのシアー変換



戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	画像のサイズのいずれかの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

GetShearQuad

ソース ROI 矩形をシアー変換でマッピング
した結果の四角形の頂点座標を計算する。

```
IppStatus ippiGetShearQuad(IppiRect roi, double quad[4][2],
    double xShear, double yShear, double xShift, double yShift);
```

引数

roi	ソース・イメージの ROI
quad	出力配列。この配列には、ソース *ROI* を ippiShear 関数でマッピングした結果の四角形の頂点座標が入る。
xShear, *yShear*	シアー変換係数
xShift, *yShift*	水平軸および垂直軸方向のシフト値

説明

関数 ippiGetShearQuad は、ippi.h ファイルの中で宣言される。この関数は、ippiShear のサポート関数として使用する。この関数は、係数 *xShear*、*yShear* とシフト値 *xShift*、*yShift* を使用するシアー変換関数 ippiShear によって、ソース ROI をマッピングした結果の四角形の頂点座標を計算する。
配列 *quad*[4][2] の第 1 次元 [4] は頂点の数に等しい。第 2 次元 [2] は頂点の x 座標と y 座標を意味する。四角形の各頂点は以下の意味を持つ。

quad[0] は、ソース ROI の座標変換された左上隅に対応する。
quad[1] は、ソース ROI の座標変換された右上隅に対応する。
quad[2] は、ソース ROI の座標変換された右下隅に対応する。
quad[3] は、ソース ROI の座標変換された左下隅に対応する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値はエラーを示す。

GetShearBound

ソース ROI を *ippiShear* 関数で変換した場合の境界矩形を計算する。

```
IppStatus ippiGetShearBound(IppiRect roi, double bound[2][2],
    double xShear, double yShear, double xShift, double yShift);
```

引数

roi	ソース・イメージの ROI
bound	出力配列。この配列には、ソース *ROI* を変換した場合の境界矩形の頂点座標が入る。
xShear, yShear	シアー変換係数
xShift, yShift	水平軸および垂直軸方向のシフト値

説明

関数 ippiGetShearBound は、ippi.h ファイルの中で宣言される。この関数は、ippiShear のサポート関数として使用する。この関数は、係数 *xShear*、*yShear* とシフト値 *xShift*、*yShift* を使用するシアー変換関数 ippiShear によって、ソース ROI をマッピングした結果の四角形 *quad* の最小境界矩形の頂点座標を計算する。
bound[0] は左上隅の x, y 座標を指定し、*bound*[1] は右下隅の x, y 座標を指定する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

WarpAffine

ソース・イメージの一般アフィン変換を実行する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiWarpAffine_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* pDst, int dstStep, IppiRect dstROI,
      const double coeffs[2][3], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiWarpAffine_<mod>(const Ipp<datatype>* const pSrc[3],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstROI,
      const double coeffs[2][3], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	32f_P3R

```
IppStatus ippiWarpAffine_<mod>(const Ipp<datatype>* const pSrc[4],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstROI,
      const double coeffs[2][3], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R	32f_P4R

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）

srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージの ROI
pDst	デスティネーション・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別インタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstROI	デスティネーション・イメージの ROI
coeffs	アフィン変換係数
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次たたみ込み補間
IPPI_SMOOTH_EDGE	上記の補間方法とともにエッジ平滑化オプションを使用する。

説明

関数 ippiWarpAffine は、ippi.h ファイルの中で宣言される。このアフィン・ワープは、次の公式に従って、ソース・イメージのピクセル座標（*x,y*）を変換する。

$$x' = c_{00}*x + c_{01}*y + c_{02}$$

$$y' = c_{10}*x + c_{11}*y + c_{12}$$

x' と y' は、変換後の画像のピクセル座標を表し、c_{ij} は、配列 *coeffs* で渡されたアフィン変換係数を表す。

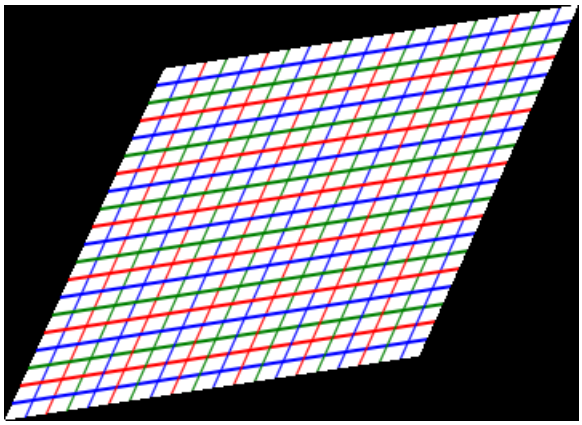
アフィン・ワープは、スケーリング、回転、平行移動、引き伸ばし、シアリングなど基本的な変換が結合した汎用リニア変換である。アフィン・ワープでは、平行なラインは常に平行なラインに変換され、変換後もライン上のポイント間距離は変わらない。

画像の変換部分は、*interpolation* パラメータで指定された補間方法を使用して再サンプリングされ、デスティネーション・イメージの ROI に書き込まれる。

図 12-6 は、サンプル・イメージにアフィン・ワープ関数 ippiWarpAffine を適用した例を示している。

アフィン変換パラメータを計算するには、本章で後に説明する ippiGetAffineQuad、ippiGetAffineBound、ippiGetAffineTransform の各関数を使用する。

図 12-6　画像のアフィン変換



戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	画像のサイズのいずれかの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

WarpAffineBack

画像の逆アフィン変換を実行する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiWarpAffineBack_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* pDst, int dstStep, IppiRect dstROI,
      const double coeffs[2][3], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiWarpAffineBack_<mod>(const Ipp<datatype>* const pSrc[3],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstROI,
      const double coeffs[2][3], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	32f_P3R

```
IppStatus ippiWarpAffineBack_<mod>(const Ipp<datatype>* const pSrc[4],
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstROI,
      const double coeffs[2][3], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R	32f_P4R

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージの ROI

pDst	デスティネーション・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別インタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstROI	デスティネーション・イメージの ROI
coeffs	アフィン変換係数
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。 `IPPI_INTER_NN` 直近サンプル補間 `IPPI_INTER_LINEAR` リニア補間 `IPPI_INTER_CUBIC` 3 次たたみ込み補間

説明

関数 ippiWarpAffineBack、ippi.h ファイルの中で宣言される。この関数は、ippiWarpAffine 関数で定義される変換の逆変換を実行する。変換後の画像のピクセル座標 x' および y' は、次の式で求められる。

$$c_{00}*x' + c_{01}*y' + c_{02} = x$$

$$c_{10}*x' + c_{11}*y' + c_{12} = y$$

x と y はソース・イメージ内のピクセル座標を表す。c_{ij} は配列 *coeffs* で与えられた係数である。したがって、ippiWarpAffineBack を呼び出す前にアプリケーション・プログラムで変換係数を逆変換する必要はない。

逆変換関数におけるソースおよびデスティネーション ROI の処理方法は、他のジオメトリ変換関数と異なることに注意する。逆変換関数では、次のロジックを使用する。

- デスティネーション ROI 内の各ピクセルの座標に逆変換を適用する。その結果、ソース・イメージ内のいずれかのピクセルの座標が得られる。
- 得られたソース・ピクセルがソース ROI 内にあれば、それに従ってデスティネーション ROI 内の対応するピクセルを変更する。ソース ROI 内になければ、変更は行わない。

上記のアルゴリズムは、次のような擬似的コードで表される。

```
for (j = dstROI.y; j < dstROI.y + dstROI.height; j ++) {
    for (i = dstROI.x; i < dstROI.x + dstROI.width; i ++) {
        sx = TransformX(i, j);
        sy = TransformY(i, j);
        if ( sx >= srcROI.x && sx < srcROI.x + srcROI.width &&
sy >= srcROI.y && sy < srcROI.y + srcROI.height) {
            dst[i, j] = Interpolate(sx, sy);}
    }
}
```

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	画像のサイズのいずれかの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

WarpAffineQuad

ソース・イメージに対して、指定のソース四角形を指定のデスティネーション四角形に変換するようなアフィン・ワープを実行する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiWarpAffineQuad_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI, const double
      srcQuad[4][2], Ipp<datatype>* pDst, int dstStep, IppiRect dstROI,
      const double dstQuad[4][2], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiWarpAffineQuad_<mod>(const Ipp<datatype>*
      const pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcROI,
      const double srcQuad[4][2], Ipp<datatype>* const pDst[3],
      int dstStep, IppiRect dstROI, const double dstQuad[4][2],
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	32f_P3R

```
IppStatus ippiWarpAffineQuad_<mod>(const Ipp<datatype>*
      const pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcROI,
      const double srcQuad[4][2], Ipp<datatype>* const pDst[4],
      int dstStep, IppiRect dstROI, const double dstQuad[4][2],
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R	32f_P4R

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージの ROI
srcQuad	ソース・イメージ内に指定された四角形
pDst	デスティネーション・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別インタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstROI	デスティネーション・イメージの ROI

dstQuad	デスティネーション・イメージ内に指定された四角形
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次たたみ込み補間
IPPI_SMOOTH_EDGE	上記の補間方法とともにエッジ平滑化オプションを使用する。

説明

関数 ippiWarpAffineQuad は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpAffine 関数によるピクセルのマッピングと同じ公式を使用して、ソース・イメージ ROI にアフィン変換を適用する。ただし、ippiWarpAffineQuad では、参照用に指定されたソース四角形 *srcQuad* からデスティネーション・イメージ内の指定された四角形 *dstQuad* へのマッピングに基づいて、変換係数を内部で計算する点が異なる。

四角形 *srcQuad*[4][2] または *dstQuad*[4][2] を指定する配列の第 1 次元 [4] は、頂点の数に等しい。第 2 次元 [2] は、頂点の x 座標と y 座標を意味する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	画像のサイズのいずれかの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

GetAffineQuad

ソース ROI 矩形をアフィン変換でマッピングした結果の四角形の頂点座標を計算する。

```
IppStatus ippiGetAffineQuad(IppiRect roi, double quad[4][2],
    const double coeffs[2][3]);
```

引数

roi	ソース ROI（ippRect 型）
quad	出力配列。この配列には、ソース *ROI* をアフィン変換関数 ippiWarpAffine でマッピングした結果の四角形の頂点座標が入る。
coeffs	指定されたアフィン変換係数

説明

関数 ippiGetAffineQuad は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpAffine のサポート関数として使用する。この関数は、係数 *coeffs* を使用するアフィン変換関数 ippiWarpAffine によって、ソース ROI をマッピングした結果の四角形の頂点座標を計算する。

配列 *quad*[4][2] の第 1 次元 [4] は頂点の数に等しい。第 2 次元 [2] は頂点の x 座標と y 座標を意味する。四角形の各頂点は以下の意味を持つ。

quad[0] は、ソース ROI の座標変換された左上隅に対応する。
quad[1] は、ソース ROI の座標変換された右上隅に対応する。
quad[2] は、ソース ROI の座標変換された右下隅に対応する。
quad[3] は、ソース ROI の座標変換された左下隅に対応する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値はエラーを示す。

GetAffineBound

ソース ROI を *ippiWarpAffine* 関数で変換した場合の境界矩形を計算する。

```
IppStatus ippiGetAffineBound(IppiRect roi, double bound[2][2],
    const double coeffs[2][3]);
```

引数

roi	ソース・イメージの ROI
bound	出力配列。この配列には、ソース *ROI* を変換した場合の境界矩形の頂点座標が入る。
coeffs	指定されたアフィン変換係数

説明

関数 ippiGetAffineBound は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpAffine のサポート関数として使用する。この関数は、係数 *coeffs* を使用するアフィン変換関数 ippiWarpAffine によって、ソース ROI をマッピングした結果の四角形 *quad* の最小境界矩形の頂点座標を計算する。

bound[0] は左上隅の x, y 座標を指定し、*bound*[1] は右下隅の x, y 座標を指定する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

GetAffineTransform

ソース ROI と、*ippiWarpAffine* 関数でそのソース ROI をマッピングした結果の四角形を指定して、アフィン変換係数を計算する。

```
IppStatus ippiGetAffineTransform(IppiRect roi, const double
    quad[4][2], double coeffs[2][3]);
```

引数

roi	ソース・イメージの ROI
quad	アフィン変換関数 ippiWarpAffine でソース *ROI* をマッピングした結果の四角形の頂点座標
coeffs	出力配列。この配列には、目的のアフィン変換係数が入る。

説明

関数 ippiGetAffineTransform は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpAffine のサポート関数として使用する。この関数では、ソース ROI と、ippiWarpAffine 関数でそのソース ROI をマッピングした結果の四角形 *quad* を指定して、アフィン変換係数 *coeffs* を計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値はエラーを示す。

WarpPerspective

指定された変換係数を使用して、ソース・イメージの透視ワープを実行する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiWarpPerspective_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* pDst, int dstStep, IppiRect dstROI,
      const double coeffs[3][3], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiWarpPerspective_<mod>(const Ipp<datatype>* const
      pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstROI,
       const double coeffs[3][3], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	32f_P3R

```
IppStatus ippiWarpPerspective_<mod>(const Ipp<datatype>* const
      pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstROI,
      const double coeffs[3][3], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R	32f_P4R

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）

srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージの ROI
pDst	デスティネーション・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別インタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstROI	デスティネーション・イメージの ROI
coeffs	透視変換係数
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次たたみ込み補間
IPPI_SMOOTH_EDGE	上記の補間方法とともにエッジ平滑化オプションを使用する。

説明

関数 ippiWarpPerspective は、ippi.h ファイルの中で宣言される。この透視変換関数は、次の公式に従って、ソース・イメージのピクセル座標（*x,y*）を変換する。

$$x' = (c_{00}*x + c_{01}*y + c_{02})/(c_{20}*x + c_{21}*y + c_{22})$$

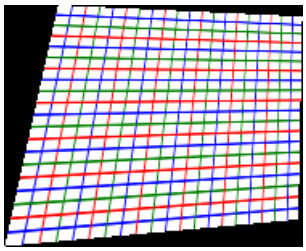
$$y' = (c_{10}*x + c_{11}*y + c_{12})/(c_{20}*x + c_{21}*y + c_{22})$$

x' と y' は変換後の画像内のピクセル座標を表す。c_{ij} は、配列 *coeffs* で渡された透視変換係数である。

画像の変換部分は、*interpolation* パラメータで指定された補間方法を使用して再サンプリングされ、デスティネーション・イメージの ROI に書き込まれる。

図 12-7 は、サンプル・イメージに透視変換関数 ippiWarpPerspective を適用した例を示している。

図 12-7　画像の透視変換



ippiWarpPerspective 関数によるソース ROI の変換結果を見積もるには、後述の節で説明する ippiGetPerspectiveQuad と ippiGetPerspectiveBound の2つの関数を使用する。ソース ROI を指定の四角形にマッピングするような透視変換係数を計算するには、ippiGetPerspectiveTransform 関数を使用する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	画像のサイズのいずれかの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

WarpPerspectiveBack

ソース・イメージの逆透視ワープを実行する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiWarpPerspectiveBack_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* pDst, int dstStep, IppiRect dstROI,
      const double coeffs[3][3], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiWarpPerspectiveBack_<mod>(const Ipp<datatype>* const
      pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstROI,
       const double coeffs[3][3], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	32f_P3R

```
IppStatus ippiWarpPerspectiveBack_<mod>(const Ipp<datatype>* const
      pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstROI,
       const double coeffs[3][3], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R	32f_P4R

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）

srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージの ROI
pDst	デスティネーション・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別インタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstROI	デスティネーション・イメージの ROI
coeffs	透視変換係数
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次たたみ込み補間

説明

関数 ippiWarpPerspectiveBack は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpPerspective 関数で定義される変換の逆変換を実行する。変換後の画像のピクセル座標 x' および y' は、次の式で求められる。

$$(c_{00}*x' + c_{01}*y' + c_{02})/(c_{20}*x' + c_{21}*y' + c_{22}) = x$$

$$(c_{10}*x' + c_{11}*y' + c_{12})/(c_{20}*x' + c_{21}*y' + c_{22}) = y$$

x と y はソース・イメージ内のピクセル座標を表す。c_{ij} は配列 *coeffs* で与えられた係数である。したがって、ippiWarpPerspectiveBack を呼び出す前にアプリケーション・プログラムで変換係数を逆変換する必要はない。

逆変換関数におけるソースおよびデスティネーション ROI の処理方法は、他のジオメトリ変換関数と異なることに注意する。詳細については、ippiWarpAffineBack 関数の説明を参照のこと。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	画像のサイズのいずれかの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

WarpPerspectiveQuad

ソース・イメージに対して、指定のソース四角形を指定のデスティネーション四角形に変換するような透視ワープを実行する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiWarpPerspectiveQuad_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI, const double
      srcQuad[4][2], Ipp<datatype>* pDst, int dstStep, IppiRect
      dstROI,
      const double dstQuad[4][2], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiWarpPerspectiveQuad_<mod>(const Ipp<datatype>*
      const pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcROI,
      const double srcQuad[4][2], Ipp<datatype>* const pDst[3],
      int dstStep, IppiRect dstROI, const double dstQuad[4][2],
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	32f_P3R

```
IppStatus ippiWarpPerspectiveQuad_<mod>(const Ipp<datatype>*
      const pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcROI,
      const double srcQuad[4][2], Ipp<datatype>* const pDst[4],
      int dstStep, IppiRect dstROI, const double dstQuad[4][2],
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

```
      8u_P4R       32f_P4R
```

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージの ROI
srcQuad	ソース・イメージ内に指定された四角形
pDst	デスティネーション・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別インタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstROI	デスティネーション・イメージの ROI
dstQuad	デスティネーション・イメージ内に指定された四角形
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次たたみ込み補間

説明

関数 ippiWarpPerspectiveQuad は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpPerspective 関数によるピクセルのマッピングと同じ公式を使用して、ソース・イメージ ROI に透視変換を適用する。ただし、ippiWarpPerspectiveQuad では、参照用に指定されたソース四角形 *srcQuad* からデスティネーション・イメージ内の指定された四角形 *dstQuad* へのマッピングに基づいて、変換係数を内部で計算する点が異なる。

四角形 *srcQuad*[4][2] または *dstQuad*[4][2] を指定する配列の第 1 次元 [4] は、頂点の数に等しい。第 2 次元 [2] は、頂点の *x* 座標と y 座標を意味する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	画像のサイズのいずれかの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

GetPerspectiveQuad

ソース ROI 矩形を透視変換でマッピングした結果の四角形の頂点座標を計算する。

```
IppStatus ippiGetPerspectiveQuad(IppiRect roi, double quad[4][2],
    const double coeffs[3][3]);
```

引数

roi	ソース・イメージの ROI
quad	出力配列。この配列には、ソース *ROI* を透視変換関数 ippiWarpPerspective でマッピングした結果の四角形の頂点座標が入る。

coeffs	指定された透視変換係数

説明

関数 ippiGetPerspectiveQuad は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpPerspective のサポート関数として使用する。この関数は、係数 *coeffs* を使用する透視変換関数 ippiWarpPerspective によって、ソース ROI をマッピングした結果の四角形の頂点座標を計算する。

配列 *quad*[4][2] の第 1 次元 [4] は頂点の数に等しい。第 2 次元 [2] は頂点の x 座標と y 座標を意味する。四角形の各頂点は以下の意味を持つ。

quad[0] は、ソース ROI の座標変換された左上隅に対応する。
quad[1] は、ソース ROI の座標変換された右上隅に対応する。
quad[2] は、ソース ROI の座標変換された右下隅に対応する。
quad[3] は、ソース ROI の座標変換された左下隅に対応する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値はエラーを示す。

GetPerspectiveBound

ソース ROI を *ippiWarpPerspective* 関数で変換した場合の境界矩形を計算する。

```
IppStatus ippiGetPerspectiveBound(IppiRect roi, double bound[2][2],
    const double coeffs[3][3]);
```

引数

roi	ソース・イメージの ROI
bound	出力配列。この配列には、ソース *ROI* を変換した場合の境界矩形の頂点座標が入る。
coeffs	指定された透視変換係数

説明

関数 ippiGetPerspectiveBound は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpPerspective のサポート関数として使用する。この関数は、係数 *coeffs* を使用する透視変換関数 ippiWarpPerspective によって、ソース ROI をマッピングした結果の四角形 *quad* の最小境界矩形の頂点座標を計算する。

bound[0] は左上隅の x, y 座標を指定し、*bound*[1] は右下隅の x, y 座標を指定する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

GetPerspectiveTransform

ソース ROI と、*ippiWarpPerspective* 関数でそのソース ROI をマッピングした結果の四角形を指定して、透視変換係数を計算する。

```
IppStatus ippiGetPerspectiveTransform(IppiRect roi, const double
    quad[4][2], double coeffs[3][3]);
```

引数

roi	ソース・イメージの ROI
quad	透視変換関数 ippiWarpPerspective でソース *ROI* をマッピングした結果の四角形の頂点座標
coeffs	出力配列。この配列には、目的の透視変換係数が入る。

説明

関数 ippiGetPerspectiveTransform は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpPerspective のサポート関数として使用する。この関数では、ソース ROI と、ippiWarpPerspective 関数でそのソース ROI をマッピングした結果の四角形 *quad* を指定して、透視変換係数 *coeffs* を計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値はエラーを示す。

WarpBilinear

指定された変換係数を使用して、ソース・イメージのバイリニア・ワープを実行する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiWarpBilinear_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* pDst, int dstStep, IppiRect dstROI,
      const double coeffs[2][4], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiWarpBilinear_<mod>(const Ipp<datatype>* const
      pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstROI,
      const double coeffs[2][4], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	32f_P3R

```
IppStatus ippiWarpBilinear_<mod>(const Ipp<datatype>* const
      pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstROI,
      const double coeffs[2][4], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R	32f_P4R

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージの ROI
pDst	デスティネーション・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別インタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstROI	デスティネーション・イメージの ROI
coeffs	バイリニア変換係数
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次たたみ込み補間
IPPI_SMOOTH_EDGE	上記の補間方法とともにエッジ平滑化オプションを使用する。

説明

関数 ippiWarpBilinear は、ippi.h ファイルの中で宣言される。このバイリニア・ワープ関数は、次の公式に従って、ソース・イメージのピクセル座標（*x*,*y*）を変換する。

$$x' = c_{00}*xy + c_{01}*x + c_{02}*y + c_{03}$$

$$y' = c_{10}*xy + c_{11}*x + c_{12}*y + c_{13}$$

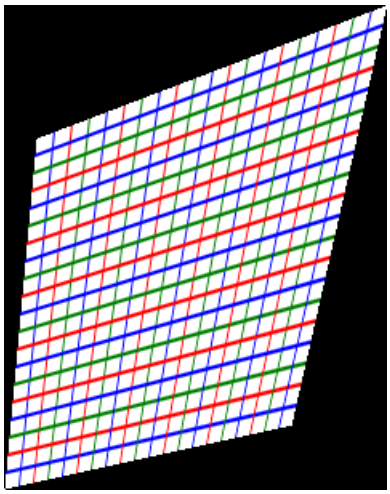
x' と y' は変換後の画像内のピクセル座標を表す。c_{ij} は、配列 *coeffs* で渡されたバイリニア変換係数である。

バイリニア変換を行っても、ライン上のポイント間距離は変わらない。

ソース・イメージの変換部分は、*interpolation* パラメータで指定された補間方法を使用して再サンプリングされ、デスティネーション・イメージの ROI に書き込まれる。

図 12-8 は、サンプル・イメージにバイリニア・ワープ関数 ippiWarpBilinear を適用した例を示している。

図 12-8　画像のバイリニア変換



ippiWarpBilinear 関数によるソース ROI の変換結果を見積もるには、後述の節で説明する ippiGetBilinearQuad と ippiGetBilinearBound の 2 つの関数を使用する。ソース ROI を指定の四角形にマッピングするようなバイリニア変換係数を計算するには、ippiGetBilinearTransform 関数を使用する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	画像のサイズのいずれかの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

WarpBilinearBack

ソース・イメージの逆バイリニア・ワープを実行する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiWarpBilinearBack_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* pDst, int dstStep, IppiRect dstROI,
      const double coeffs[2][4], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiWarpBilinearBack_<mod>(const Ipp<datatype>* const
      pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstROI,
       const double coeffs[2][4], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R	32f_P3R

```
IppStatus ippiWarpBilinearBack_<mod>(const Ipp<datatype>* const
      pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcROI,
      Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstROI,
      const double coeffs[2][4], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R	32f_P4R

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージの ROI
pDst	デスティネーション・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別インタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstROI	デスティネーション・イメージの ROI
coeffs	バイリニア変換係数
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次たたみ込み補間

説明

関数 ippiWarpBilinearBack は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpBilinear 関数で定義される変換の逆変換を実行する。変換後の画像のピクセル座標 x' および y' は、次の式で求められる。

$$c_{00}*x'y' + c_{01}*x' + c_{02}*y' + c_{03} = x$$

$$c_{10}*x'y' + c_{11}*x' + c_{12}*y' + c_{13} = y$$

x と y はソース・イメージ内のピクセル座標を表す。c_{ij} は配列 *coeffs* で与えられた係数である。したがって、ippiWarpBilinearBack を呼び出す前にアプリケーション・プログラムで変換係数を逆変換する必要はない。

逆変換関数におけるソースおよびデスティネーション ROI の処理方法は、他のジオメトリ変換関数と異なることに注意する。詳細は、ippiWarpAffineBack 関数の説明を参照のこと。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	画像のサイズのいずれかの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

WarpBilinearQuad

ソース・イメージに対して、指定のソース四角形を指定のデスティネーション四角形に変換するようなバイリニア・ワープを実行する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>* pSrc,
      IppiSize srcSize, int srcStep, IppiRect srcROI, const double
      srcQuad[4][2], Ipp<datatype>* pDst, int dstStep, IppiRect dstROI,
      const double dstQuad[4][2], int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

事例 2：プレーン順序データの操作

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>*
      const pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcROI,
      const double srcQuad[4][2], Ipp<datatype>* const pDst[3],
      int dstStep, IppiRect dstROI, const double dstQuad[4][2],
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P3R　　32f_P3R

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>*
      const pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcROI,
      const double srcQuad[4][2], Ipp<datatype>* const pDst[4],
      int dstStep, IppiRect dstROI, const double dstQuad[4][2],
      int interpolation);
```

サポートされる *mod* の値は、次のとおりである。

8u_P4R　　32f_P4R

引数

pSrc	ソース・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別のポインタが入った配列
srcSize	ソース・イメージのサイズ（ピクセル単位）
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
srcROI	ソース・イメージの ROI
srcQuad	ソース・イメージ内に指定された四角形
pDst	デスティネーション・イメージ原点へのポインタ。プレーン順序データの場合は、各プレーンへの個別インタが入った配列
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）
dstROI	デスティネーション・イメージの ROI
dstQuad	デスティネーション・イメージ内に指定された四角形
interpolation	画像を再サンプリングする際の補間タイプ。次のいずれかの値を指定する。

IPPI_INTER_NN	直近サンプル補間
IPPI_INTER_LINEAR	リニア補間
IPPI_INTER_CUBIC	3 次たたみ込み補間
IPPI_SMOOTH_EDGE	上記の補間方法とともにエッジ平滑化オプションを使用する。

説明

関数 ippiWarpBilinearQuad は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpBilinear 関数によるピクセルのマッピングと同じ公式を使用して、ソース・イメージ ROI にバイリニア変換を適用する。ただし、ippiWarpBilinearQuad では、参照用に指定されたソース四角形 *srcQuad* からデスティネーション・イメージ内の指定された四角形 *dstQuad* へのマッピングに基づいて、変換係数を内部で計算する点が異なる。

四角形 *srcQuad*[4][2] または *dstQuad*[4][2] を指定する配列の第 1 次元 [4] は、頂点の数に等しい。第 2 次元 [2] は、頂点の x 座標と y 座標を意味する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pSrc* または *pDst* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	画像のサイズのいずれかの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。
ippStsInterpolationErr	*interpolation* の値が無効である場合のエラー状態を示す。

GetBilinearQuad

ソース ROI 矩形をバイリニア変換でマッピングした結果の四角形の頂点座標を計算する。

```
IppStatus ippiGetBilinearQuad(IppiRect roi, double quad[4][2],
    const double coeffs[2][4]);
```

引数

roi	ソース・イメージの ROI
quad	出力配列。この配列には、ソース *roi* をバイリニア変換関数 ippiWarpBilinear でマッピングした結果の四角形の頂点座標が入る。
coeffs	指定されたバイリニア変換係数

説明

関数 ippiGetBilinearQuad は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpBilinear のサポート関数として使用する。この関数は、係数 *coeffs* を使用するバイリニア変換関数 ippiWarpBilinear によって、ソース ROI をマッピングした結果の四角形の頂点座標を計算する。

配列 *quad*[4][2] の第 1 次元 [4] は頂点の数に等しい。第 2 次元 [2] は頂点の x 座標と y 座標を意味する。四角形の各頂点は以下の意味を持つ。

quad[0] は、ソース ROI の座標変換された左上隅に対応する。
quad[1] は、ソース ROI の座標変換された右上隅に対応する。
quad[2] は、ソース ROI の座標変換された右下隅に対応する。
quad[3] は、ソース ROI の座標変換された左下隅に対応する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値はエラーを示す。

GetBilinearBound

ソース ROI を *ippiWarpBilinear* 関数で変換した場合の境界矩形を計算する。

```
IppStatus ippiGetBilinearBound(IppiRect roi, double bound[2][2],
    const double coeffs[2][4]);
```

引数

roi	ソース・イメージの ROI
bound	出力配列。この配列には、ソース *ROI* を変換した場合の境界矩形の頂点座標が入る。

coeffs	指定されたバイリニア変換係数

説明

関数 ippiGetBilinearBound は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpBilinear のサポート関数として使用する。この関数は、係数 *coeffs* を使用するバイリニア変換関数 ippiWarpBilinear によって、ソース ROI をマッピングした結果の四角形 quad の最小境界矩形の頂点座標を計算する。

bound[0] は左上隅の x, y 座標を指定し、*bound*[1] は右下隅の x, y 座標を指定する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

GetBilinearTransform

ソース ROI と、*ippiWarpBilinear* 関数でそのソース ROI をマッピングした結果の四角形を指定して、バイリニア変換係数を計算する。

```
IppStatus ippiGetBilinearTransform(IppiRect roi, const double
    quad[4][2], double coeffs[2][4]);
```

引数

roi	ソース・イメージの ROI
quad	ippiWarpBilinear バイリニア変換関数でソース *ROI* をマッピングした結果の四角形の頂点座標
coeffs	出力配列。この配列には、目的のバイリニア変換係数が入る。

説明

関数 ippiGetBilinearTransform は、ippi.h ファイルの中で宣言される。この関数は、ippiWarpBilinear のサポート関数として使用する。この関数では、ソース ROI と、ippiWarpBilinear 関数でそのソース ROI をマッピングした結果の四角形 *quad* を指定して、バイリニア変換係数 *coeffs* を計算戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値はエラーを示す。

13 ウェーブレット変換

本章では、画像処理用 IPP に実装されている二次元離散ウェーブレット変換（DWT）関数について説明する。

離散ウェーブレット変換による多重解像度解析は、多くのアプリケーションにおいて、ウィンドウ生成や離散フーリエ解析といった手法に取って代わるすぐれた手段の 1 つである。一方、順方向二次元ウェーブレット変換については、空間内にあるいくつかの関数をもとにして 1 つの画像を分解することと見なすことができる。また、ウェーブレット変換はサブバンドでのフィルタリングや再サンプリングにも関係がある。

画像処理用インテル® IPP には、1 レベル離散ウェーブレットの分解関数と再構成関数が含まれている。変換コンテキスト構造体の初期化と割り当て解除に必要なインターフェイスも用意されている。表 13-1 に、このグループの関数の一覧を示す。

表 13-1　画像ウェーブレット変換関数

関数の基本名	操作
WTFwdInitAlloc	メモリを割り当て、順方向ウェーブレット変換コンテキスト構造体を初期化する。
WTFwdFree	順方向ウェーブレット変換コンテキスト構造体に割り当てられていたメモリを解放する。
WTFwdGetBufSize	順方向ウェーブレット変換用の外部ワーク・バッファのサイズを求める。
WTFwd	画像の 1 レベル・ウェーブレット分解を実行する。
WTInvInitAlloc	メモリを割り当て、逆方向ウェーブレット変換コンテキスト構造体を初期化する。
WTInvFree	逆方向ウェーブレット変換コンテキスト構造体に割り当てられていたメモリを解放する。
WTInvGetBufSize	逆方向ウェーブレット変換用の外部ワーク・バッファのサイズを求める。
WTInv	画像の 1 レベル・ウェーブレット再構成を実行する。

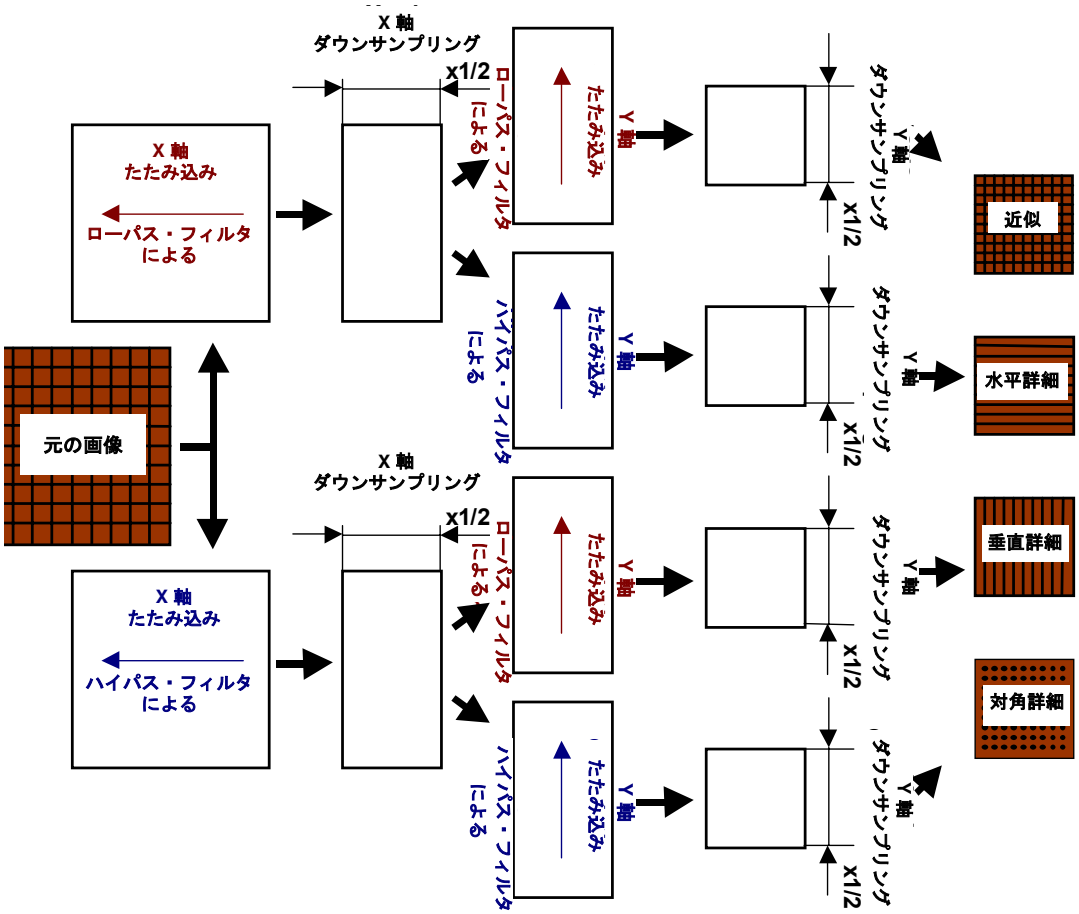
適切なフィルタ・タップを初期化関数で指定することによって、ウェーブレット変換の種類が設定できる。

インテル IPP では、切り離し可能なたたみ込みについては、一次元の有限インパルス応答フィルタしか利用できないのに注意する。

インテル IPP のウェーブレット分解関数/再構成関数は高速多相アルゴリズムを使用

する。これは、切り離し可能なたたみ込みと二項再サンプリングを別の順序で行う従来の方法と同じである。次の図は、ウェーブレットに基づいて画像を分解する際のアルゴリズムを示している。

図 13-1　ウェーブレット分解アルゴリズムの等価図



1 つのソース・イメージを分解すると、「近似画像」、「水平詳細画像」、「垂直詳細画像」、「対角詳細画像」といった、サイズの同じ 4 つの出力画像が得られる。

それぞれの分解成分の意味は、次のとおりである。

- 「近似画像」は、垂直方向と水平方向にローパス・フィルタをかけて得られる。

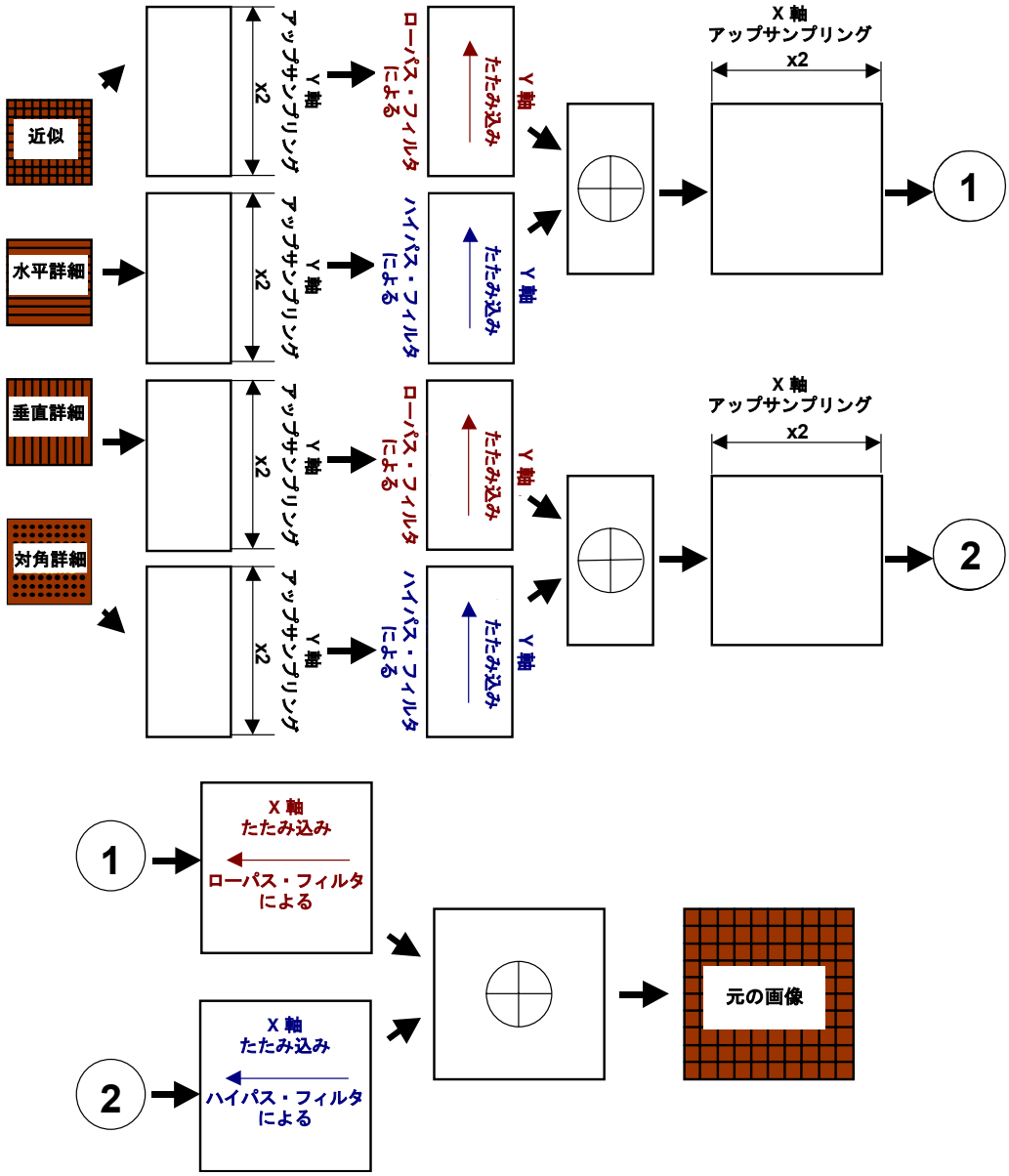
- 「水平詳細画像」は、垂直方向にハイパス・フィルタをかけ、水平方向にローパス・フィルタをかけて得られる。
- 「垂直詳細画像」は、垂直方向にローパス・フィルタをかけ、水平方向にハイパス・フィルタをかけて得られる。
- 「対角詳細画像」は、垂直方向と水平方向にハイパス・フィルタをかけて得られる。

本書では、区別するのに便利なので上記の画像名を使用する。

ウェーブレットに基づく画像再構成は、たたみ込み処理と二項アップサンプリングを別々に連続して実行する。
再構成関数は、分解処理で生成される画像と同じ 4 つの入力画像を使用する。

次の図は、1 つの画像をウェーブレットに基づいて再構成する際のアルゴリズムを示している。

図 13-2　ウェーブレット再構成アルゴリズムの等価図



ウェーブレット変換関数では、画像の処理対象領域（ROI）を指定できる。ただし、この関数は画像の ROI データの境界延長部分については一切内部で処理を実行しない。

つまり、ソース・イメージは、たたみ込み処理に必要なすべての境界データをあらかじめ含んでいなければならない。延長された画像境界サイズの計算方法の詳細については、ippiWTFwd 関数と ippiWTInv 関数の説明を参照のこと。

WTFwdInitAlloc

メモリを割り当て、順方向ウェーブレット
変換コンテキスト構造体を初期化する。

```
IppStatus ippiWTFwdInitAlloc_32f_C1R (IppiWTFwdSpec_32f_C1R** pSpec,
     const Ipp32f* pTapsLow, int lenLow, int anchorLow,
      const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
IppStatus ippiWTFwdInitAlloc_32f_C3R (IppiWTFwdSpec_32f_C3R** pSpec,
     const Ipp32f* pTapsLow, int lenLow, int anchorLow,
      const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

引数

pSpec	新たに割り当てられ、初期化された順方向 DWT コンテキスト構造体へのポインタのポインタ
pTapsLow	ローパス・フィルタ・タップへのポインタ
lenLow	ローパス・フィルタの長さ
anchorLow	ローパス・フィルタのアンカ位置
pTapsHigh	ハイパス・フィルタ・タップへのポインタ
lenHigh	ハイパス・フィルタの長さ
anchorHigh	ハイパス・フィルタのアンカ位置

説明

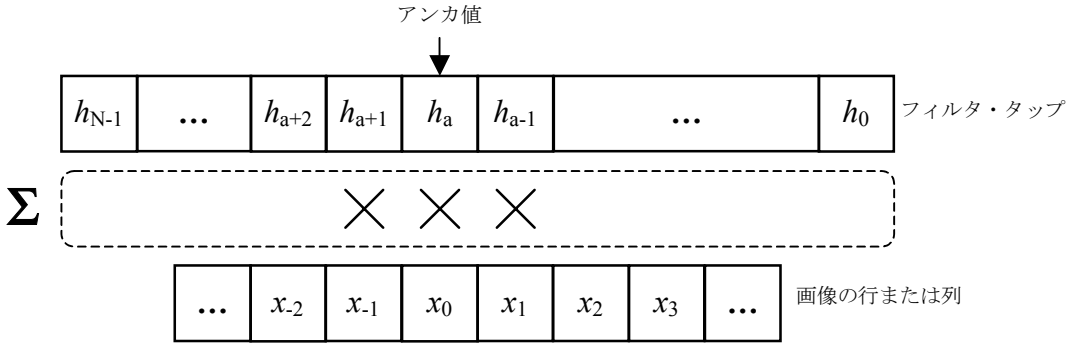
関数 ippiWTFwdInitAlloc は、ippi.h ファイルの中で宣言される。この関数は、1 レベル・ウェーブレット分解のコンテキスト構造体 *pSpec* にメモリを割り当て、この構造体を初期化する。

ここでは、画像の分解に使用するウェーブレット・フィルタ・バンクのパラメータを順方向ウェーブレット変換コンテキスト構造体がいくつか含んでいるものと仮定する。

このフィルタ・バンクは、2 つの解析フィルタから成り、ローパス分解フィルタ（目の粗いフィルタ）とハイパス分解フィルタ（目の細かいフィルタ）を含んでいる。

初期化するためには、係数 *pTapsLow*、*pTapsHigh* とアンカ位置 *anchorLow*、*anchorHigh* を組み合わせて、アプリケーションから 2 つの解析フィルタに与えなければならない。次の図に示すように、アンカ値によって、最初に左端にくるフィルタの位置が決まる。位置の基準となるのは画像の行または列である。

図 13-3　ウェーブレット分解を実行する際のアンカ値と最初のフィルタ位置



図中の *a* はアンカ値、N はフィルタ長、x_0 は処理された行または列の開始ピクセル、x_{-1}、x_{-2}、... は計算に必要なその他の境界ピクセルである。アンカ値とフィルタ長によって、分解に使用するソース・イメージの上下左右の境界サイズがすべて決まる。境界サイズの計算を C 言語でどのように表現するかについては、ippiWTFwd 関数の説明で示す。

いったん割り当てられ、初期化されたコンテキスト構造体は、複数の計算スレッドで使用でき、複数のウェーブレット分解レベルでも使用できる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pTapsLow*、*pTapsHigh*、または *pSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	フィルタ長 *lenLow* または *lenHigh* が 2 未満の場合のエラー状態を示す。
ippStsAnchorErr	アンカ位置 *anchorLow* または *anchorHigh* が 0 未満の場合のエラー状態を示す。
ippStsMemAllocErr	コンテキスト構造体にメモリが割り当てられなかった場合のエラー状態を示す。

WTFwdFree

順方向ウェーブレット変換コンテキスト構造体に割り当てられていたメモリを解放する。

```
IppStatus ippiWTFwdFree_32f_C1R (IppiWTFwdSpec_32f_C1R* pSpec);
IppStatus ippiWTFwdFree_32f_C3R (IppiWTFwdSpec_32f_C3R* pSpec);
```

引数

pSpec	割り当てられ、初期化された順方向 DWT コンテキスト構造体へのポインタ

説明

関数 ippiWTFwdFree は、ippi.h ファイルの中で宣言される。この関数は、順方向ウェーブレット変換コンテキスト構造体 *pSpec* に割り当てられていたメモリを解放する。割り当てと初期化を ippiWTFwdInitAlloc 関数で行ったコンテキスト構造体をそれぞれ解放するには、ippiWTFwdFree を呼び出す必要がある。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	コンテキスト構造体を指す *pSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	不正なコンテキスト構造体へのポインタが渡された場合のエラー状態を示す。

WTFwdGetBufSize

順方向ウェーブレット変換用の外部ワーク・バッファのサイズを求める。

```
IppStatus ippiWTFwdGetBufSize_C1R(const IppiWTFwdSpec_32f_C1R* pSpec,
      int* size);
```

```
IppStatus ippiWTFwdGetBufSize_C3R(const IppiWTFwdSpec_32f_C3R* pSpec,
      int* size);
```

引数

pSpec	割り当てられ、初期化された順方向 DWT コンテキスト構造体へのポインタ
size	順方向ウェーブレット変換に必要なワーク・バッファのサイズを受け取る変数へのポインタ

説明

関数 ippiWTFwdGetBufSize は、ippi.h ファイルの中で宣言される。この関数は、順方向ウェーブレット変換関数 ippiWTFwd が動作するのに必要なワーク・バッファのサイズ（バイト単位）を計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指示されたポインタのいずれかがNULLの場合エラー状態を示す。
ippStsContextMatchErr	不正なコンテキスト構造体へのポインタが渡された場合のエラー状態を示す。

WTFwd

画像の 1 レベル・ウェーブレット分解を実行する。

```
IppStatus ippiWTFwd_32f_C1R (const Ipp32f* pSrc,  int srcStep,
        Ipp32f* pApproxDst,   int approxStep,
        Ipp32f* pDetailXDst,  int detailXStep,
        Ipp32f* pDetailYDst,  int detailYStep,
        Ipp32f* pDetailXYDst, int detailXYStep,
        IppiSize dstRoiSize, IppiWTFwdSpec_32f_C1R* pSpec,
        Ipp8u* pBuffer);
```

```
IppStatus ippiWTFwd_32f_C3R (const Ipp32f* pSrc,  int srcStep,
        Ipp32f* pApproxDst,   int approxStep,
        Ipp32f* pDetailXDst,  int detailXStep,
        Ipp32f* pDetailYDst,  int detailYStep,
        Ipp32f* pDetailXYDst, int detailXYStep,
        IppiSize dstRoiSize, IppiWTFwdSpec_32f_C3R* pSpec,
        Ipp8u* pBuffer);
```

引数

pSrc	ソース・イメージの ROI へのポインタ
srcStep	ソース・イメージ・バッファ内のステップ（バイト単位）
pApproxDst	近似デスティネーション・イメージの ROI へのポインタ
approxStep	近似イメージ・バッファ内のステップ（バイト単位）
pDetailXDst	水平詳細デスティネーション・イメージの ROI へのポインタ
detailXStep	水平詳細イメージ・バッファ内のステップ（バイト単位）
pDetailYDst	垂直詳細デスティネーション・イメージの ROI へのポインタ
detailYStep	垂直詳細イメージ・バッファ内のステップ（バイト単位）
pDetailXYDst	対角詳細デスティネーション・イメージの ROI へのポインタ
detailXYStep	対角詳細イメージ・バッファ内のステップ（バイト単位）
dstRoiSize	デスティネーション・イメージすべての ROI のサイズ
pSpec	割り当てられ、初期化された順方向 DWT コンテキスト構造体へのポインタ
pBuffer	中間処理用に割り当てられたバッファへのポインタ

説明

関数 ippiWTFwd は、ippi.h ファイルの中で宣言される。この関数は、1 レベル・ウェーブレット分解で 1 つのソース・イメージを 4 つの細分画像に分解する。ソース・イメージは *pSrc* 引数によって指され、4 つのソース・イメージはそれぞれ

pApproxDst、*pDetailXDst*、*pDetailYDst*、*pDetailXYDst* の引数によって指されている。ippiWTFwd 関数の働きを示した等価アルゴリズムについては、図 13-1 を参照のこと。

各種ウェーブレット・パラメータは、*pSpec* によって指される順方向変換コンテキスト構造体に含まれているが、これは、ippiWTFwd を呼び出す前に、ippiWTFwdInitAlloc 関数で割り当てと初期化を行わなければならない。
この分解関数には、一時的な計算に使用するバッファが 1 つ必要である。これについては *pBuffer* 引数で指す。ippiWTFwdGetBufSize 関数を呼び出すと、必要なバッファ・サイズが求められる。このバッファ・サイズは、処理する画像のサイズには関係なく、したがってサイズの異なる複数の画像を分解するときに、同じバッファが使用できるのに注意する。しかし、マルチスレッド・モードのときは、異なるスレッドに同じバッファを使用するのは推奨しない。より高いパフォーマンスがほしければ、アラインメントの合ったメモリ・ロケーションをこのバッファに使用する。これについては、ippsMalloc 関数で割り当てられる。

pSrc 引数は、ソース・イメージの矩形 ROI とメモリ・ロケーションを指しているのに注意する。サイズは、srcWidth × srcHeight であり、下に示すように出力の ROI のサイズによって異なる。

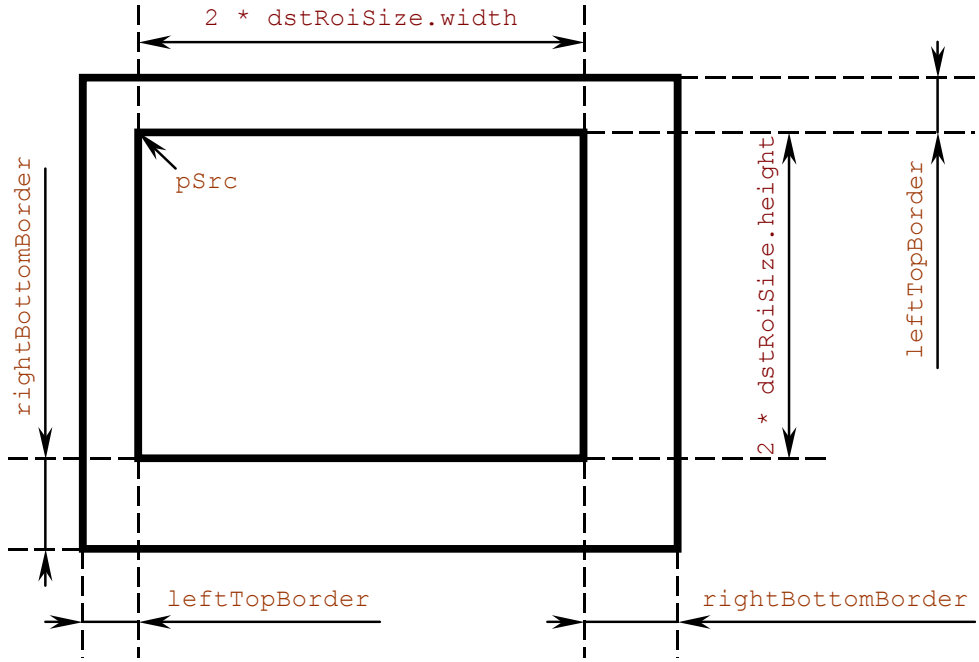
```
srcWidth  = 2 * dstRoiSize.width
srcHeight = 2 * dstRoiSize.height
```

したがって、ソース・イメージの ROI には、デスティネーション・ピクセルをいくつか計算するのに必要な境界ピクセルが含まれていない。つまり、ippiWTFwd 関数を使用する前に、隣接しているメモリ・ロケーションをいくつか埋めて、何らかの境界拡大モデル（対称、折り返しなど）をソース・イメージの ROI に適用しなければならない。

そうすると、ソース・イメージに割り当てられていたメモリ・ブロックのサイズは、追加された境界ピクセルに対処できるように、ROI と正規の境界の外側にまで拡がるはずである。

ソース・イメージの ROI と拡大された画像領域を次の図に示す。

図 13-4　ウェーブレット分解用に拡大されたソース・イメージ



以下に示す C 言語の式を使用して、拡大した画像境界サイズを計算できる。

```
int leftBorderLow    = lenLow    - 1 - anchorLow;
int leftBorderHigh   = lenHigh   - 1 - anchorHigh;

int rightBorderLow   = lenLow    - 2 - leftBorderLow;
int rightBorderHigh  = lenHigh   - 2 - leftBorderHigh;

int leftTopBorder     = IPP_MAX(leftBorderLow,  leftBorderHigh);
int rightBottomBorder = IPP_MAX(rightBorderLow, rightBorderHigh);
```

使用する各パラメータの意味については、ippiWTFwdInitAlloc 関数の説明を参照のこと。

左と上の境界は同じサイズであることに注意する。右と下の境界も同じサイズである。

境界ピクセルのぶんだけ拡がったソース・イメージ領域は、次のように定義できる。

```
srcWidthWithBorders  = srcWidth  + leftTopBorder + rightBottomBorder;
srcHeightWithBorders = srcHeight + leftTopBorder + rightBottomBorder;
```

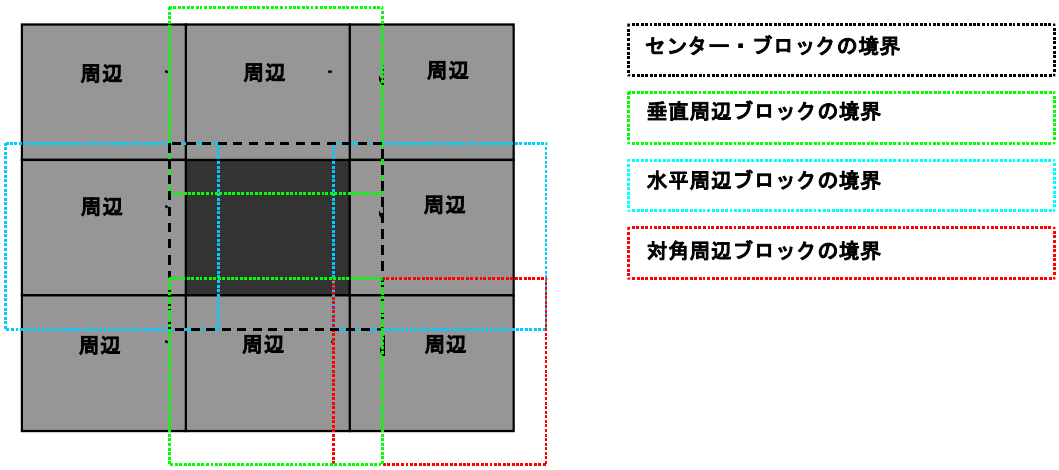
4 つのデスティネーション・イメージはすべて、*dstRoiSize* 引数で指定される同じサイズである。

逆に言えば、分解によって得られたいくつかの成分画像からソース・イメージ全体のウェーブレット再構成を実行する必要がある場合は、その再構成パスのときに、拡大されたいくつかの成分画像を使用しなければならない。詳細については、ippiWTInv 関数の説明を参照のこと。
ウェーブレット変換関数に使用されている ROI と概念は、大きな画像をブロックごと（すなわち、「タイル」ごと）に処理する場合に適用できる。それには、次のようにしてソース・イメージを細分し、部分的に重なり合ったいくつかのブロックに分ける必要がある。

- 各ブロックの主要部分（ROI）は、周辺のブロックに隣接しているが、近傍の ROI とは重ならない。
- 各ブロックの拡大境界部分は周辺ブロックの ROI と部分的に重なる。

この細分のようすを次の図に示す。

図 13-5　境界が部分的に重なるようにして画像を複数のブロックに分ける



戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定したポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	値が 0 または負のフィールドが *dstRoiSize* に含まれている場合のエラー状態を示す。
ippStsStepErr	バッファに含まれているステップ数が 0 以下の場合のエラー状態を示す。
ippStsContextMatchErr	不正なコンテキスト構造体へのポインタが渡された場合のエラー状態を示す。

WTInvInitAlloc

メモリを割り当て、逆方向ウェーブレット変換コンテキスト構造体を初期化する。

```
IppStatus ippiWTInvInitAlloc_32f_C1R (IppiWTInvSpec_32f_C1R** pSpec,
        const Ipp32f* pTapsLow,  int lenLow,  int anchorLow,
        const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
IppStatus ippiWTInvInitAlloc_32f_C3R (IppiWTInvSpec_32f_C3R** pSpec,
        const Ipp32f* pTapsLow,  int lenLow,  int anchorLow,
        const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

引数 :

pSpec	新たに割り当てられ、初期化された逆方向 DWT コンテキスト構造体へのポインタのポインタ
pTapsLow	ローパス・フィルタ・タップへのポインタ
lenLow	ローパス・フィルタの長さ
anchorLow	ローパス・フィルタのアンカ位置
pTapsHigh	ハイパス・フィルタ・タップへのポインタ
lenHigh	ハイパス・フィルタの長さ
anchorHigh	ハイパス・フィルタのアンカ位置

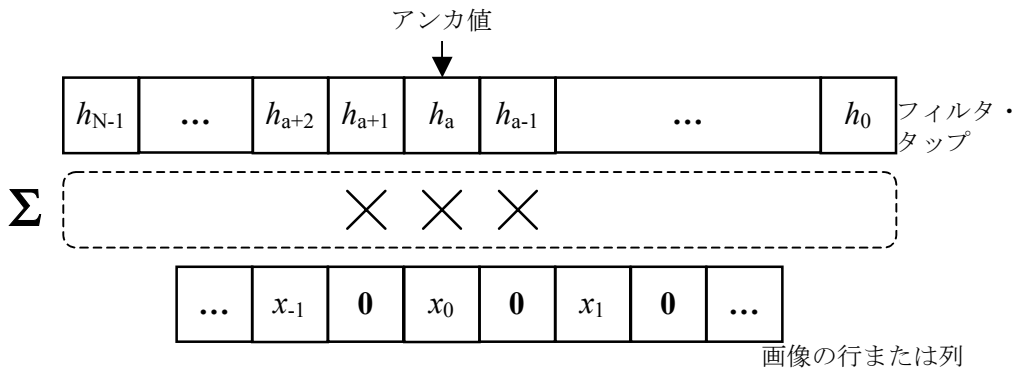
説明

関数 ippiWTInvInitAlloc は、ippi.h ファイルの中で宣言される。この関数は、1 レベル・ウェーブレット再構成のコンテキスト構造体 *pSpec* にメモリを割り当て、この構造体を初期化する。

ここでは、画像の再構成に使用するウェーブレット・フィルタ・バンクのパラメータをいくつか逆方向ウェーブレット変換コンテキスト構造体が含んでいると仮定する。このフィルタ・バンクは、2 つの合成フィルタから成り、ローパス再構成フィルタ（目の粗いフィルタ）とハイパス再構成フィルタ（目の細かいフィルタ）を含んでいる。

初期化するためには、係数 *pTapsLow*、*pTapsHigh* とアンカ位置 *anchorLow*、*anchorHigh* を組み合わせて、アプリケーションから 2 つの合成フィルタに与えなければならない。下の図に示すように、アンカ値によって、最初に左端にくるフィルタの位置が決まる。位置の基準となるのは画像の行または列である。

図 13-6　ウェーブレット再構成を行うときのアンカ値と最初のフィルタ位置



図中の *a* はアンカ値、N はフィルタ長、x_0 は処理された行または列の開始ピクセル、x_{-1}、x_{-2}、... は計算に必要なその他の境界ピクセルである。この図からわかるように、アンカ位置については、アップサンプリングされたソース・データの位置を基準に指定されている。アンカ値とフィルタ長によって、再構成に使用するソース・イメージの上下左右の境界サイズがすべて決まる。境界サイズの計算を C 言語でどのように表現するかは、ippiWTInv 関数の説明に示す。

いったん割り当てられ、初期化されたコンテキスト構造体は、複数の計算スレッドで使用でき、複数のウェーブレット再構成レベルでも使用できる。

戻り値

ippStsNoErr	エラーがないのを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pTapsLow*、*pTapsHigh*、または *pSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	フィルタ長 *lenLow* または *lenHigh* が 2 未満の場合のエラー状態を示す。
ippStsAnchorErr	アンカ位置 *anchorLow* または *anchorHigh* が 0 未満の場合のエラー状態を示す。
ippStsMemAllocErr	コンテキスト構造体にメモリが割り当てられなかった場合のエラー状態を示す。

WTInvFree

逆方向ウェーブレット変換コンテキスト構造体に割り当てられていたメモリを解放する。

```
IppStatus ippiWTInvFree_32f_C1R (IppiWTInvSpec_32f_C1R* pSpec);
IppStatus ippiWTInvFree_32f_C3R (IppiWTInvSpec_32f_C3R* pSpec);
```

引数

pSpec	割り当てられ、初期化された逆方向 DWT コンテキスト構造体へのポインタ

説明

関数 ippiWTInvFree は、ippi.h ファイルの中で宣言される。この関数は、逆方向ウェーブレット変換コンテキスト構造体 *pSpec* に割り当てられていたメモリを解放する。割り当てと初期化を ippiWTInvInitAlloc 関数で行ったコンテキスト構造体をそれぞれ解放するには、ippiWTInvFree を呼び出す必要がある。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

ippStsNullPtrErr	コンテキスト構造体を指す *pSpec* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	不正なコンテキスト構造体へのポインタが渡された場合のエラー状態を示す。

WTInvGetBufSize

逆方向ウェーブレット変換用の外部ワーク・バッファのサイズを求める。

```
IppStatus ippiWTInvGetBufSize_C1R( const IppiWTInvSpec_32f_C1R* pSpec,
      int* size );
IppStatus ippiWTInvGetBufSize_C3R( const IppiWTInvSpec_32f_C3R* pSpec,
      int* size );
```

引数

pSpec	割り当てられ、初期化された逆方向 DWT コンテキスト構造体へのポインタ
size	逆方向ウェーブレット変換に必要なワーク・バッファのサイズを受け取る変数へのポインタ

説明

関数 ippiWTInvGetBufSize は、ippi.h ファイルの中で宣言される。この関数は、逆方向ウェーブレット変換関数 ippiWTInv が動作するのに必要なワーク・バッファのサイズ（バイト単位）を計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定したポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	不正なコンテキスト構造体へのポインタが渡された場合のエラー状態を示す。

WTInv

画像の 1 レベル・ウェーブレット再構成を
実行する。

```
IppStatus ippiWTInv_32f_C1R (
        const Ipp32f* pApproxSrc,   int approxStep,
        const Ipp32f* pDetailXSrc,  int detailXStep,
        const Ipp32f* pDetailYSrc,  int detailYStep,
        const Ipp32f* pDetailXYSrc, int detailXYStep,
        IppiSize srcRoiSize, Ipp32f* pDst,  int dstStep,
        IppiWTInvSpec_32f_C1R* pSpec, Ipp8u* pBuffer);
IppStatus ippiWTInv_32f_C3R (
        const Ipp32f* pApproxSrc,   int approxStep,
        const Ipp32f* pDetailXSrc,  int detailXStep,
        const Ipp32f* pDetailYSrc,  int detailYStep,
        const Ipp32f* pDetailXYSrc, int detailXYStep,
        IppiSize srcRoiSize, Ipp32f* pDst,  int dstStep,
        IppiWTInvSpec_32f_C3R* pSpec, Ipp8u* pBuffer);
```

引数

pApproxSrc	近似ソース・イメージの ROI へのポインタ
approxStep	近似イメージ・バッファ内のステップ（バイト単位）
pDetailXSrc	水平詳細ソース・イメージの ROI へのポインタ
detailXStep	水平詳細イメージ・バッファ内のステップ（バイト単位）
pDetailYSrc	垂直詳細ソース・イメージの ROI へのポインタ
detailYStep	垂直詳細イメージ・バッファ内のステップ（バイト単位）
pDetailXYSrc	対角詳細ソース・イメージの ROI へのポインタ
detailXYStep	対角詳細イメージ・バッファ内のステップ（バイト単位）
srcRoiSize	ソース・イメージすべての ROI のサイズ
pDst	デスティネーション・イメージの ROI へのポインタ
dstStep	デスティネーション・イメージ・バッファ内のステップ（バイト単位）

pSpec	割り当てられ、初期化された逆方向 DWT コンテキスト構造体へのポインタ
pBuffer	中間処理用に割り当てられたバッファへのポインタ

説明

関数 ippiWTInv は、ippi.h ファイルの中で宣言される。この関数は、4 つの成分画像にウェーブレット再構成の操作を実行して出力画像 *pDst* を得る。
ippiWTInv 関数の動作を示した等価アルゴリズムについては、図 13-2 を参照のこと。各種ウェーブレット・パラメータは、*pSpec* によって指される逆方向変換コンテキスト構造体に含まれているが、これは、ippiWTInv を呼び出す前に、ippiWTInvInitAlloc 関数で割り当てと初期化を行わなければならない。この再構成関数には、一時的な計算に使用するバッファが 1 つ必要である。
これについては *pBuffer* 引数で指す。ippiWTInvGetBufSize 関数を呼び出すと、必要なバッファ・サイズが求まる。このバッファ・サイズは、処理する画像のサイズには関係なく、したがってサイズの異なる複数の画像を再構成するときに、同じバッファが使用できるのに注意する。しかし、マルチスレッド・モードのときは、異なるスレッドに同じバッファを使用するのは推奨しない。より高いパフォーマンスがほしい場合は、アラインメントの合ったメモリ・ロケーションをこのバッファに使用する。これについては、ippsMalloc 関数で割り当てられる。

pApproxSrc、*pDetailXSrc*、*pDetailYSrc*、*pDetailXYSrc* の各ポインタ引数は、境界を除いてソース・イメージの ROI を指している。ソース・イメージの ROI はすべて同じサイズ *srcRoiSize* であるが、デスティネーション・イメージのサイズは、次に示した関係によりそれぞれ異なる。

```
dstWidth  = 2 * srcRoiSize.width;
dstHeight = 2 * srcRoiSize.height;
```

ソース・イメージの ROI は、デスティネーション・ピクセルをいくつか計算するのに必要な境界ピクセルを含んでいないため、隣接しているメモリ・ロケーションをいくつか埋めることにより、何らかの境界拡大モデル（対称、折り返しなど）を、すべてのソース・イメージの ROI に適用しなければならない。
ソース・イメージが異なれば境界サイズも異なる場合があるのに注意する。以下に示した C 言語の式を使用して、拡がった画像境界サイズを計算できる。

```
int leftBorderLow    = (lenLow  - 1 - anchorLow) / 2;
int leftBorderHigh   = (lenHigh - 1 - anchorHigh) / 2;

int rightBorderLow   = (anchorLow  + 1) / 2;
int rightBorderHigh  = (anchorHigh + 1) / 2;
```

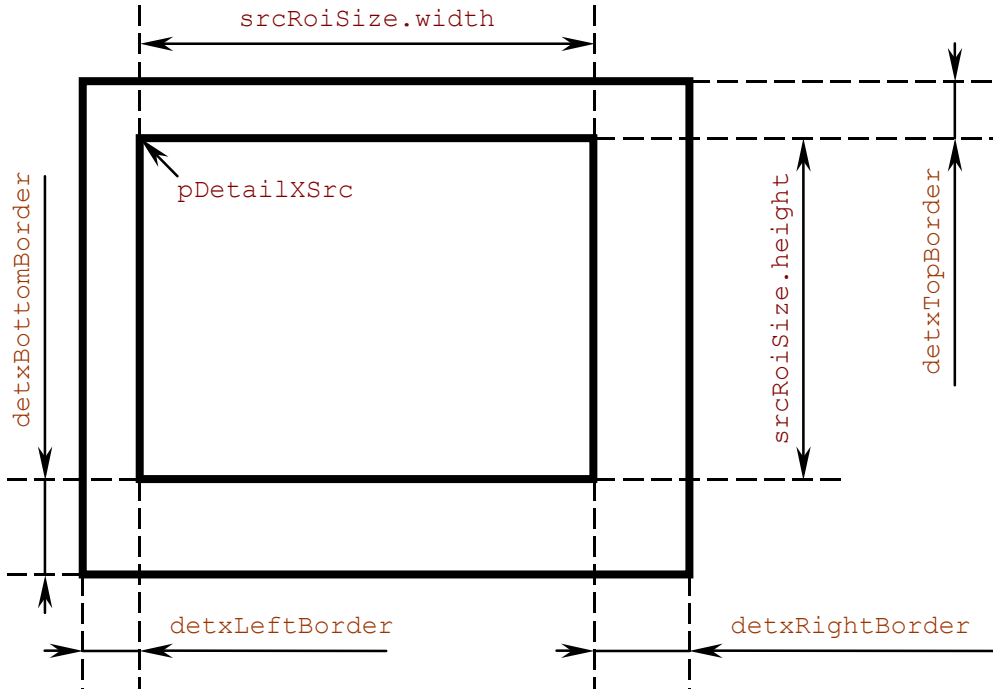
```
int apprLeftBorder   = leftBorderLow;
int apprRightBorder  = rightBorderLow;
int apprTopBorder    = leftBorderLow;
int apprBottomBorder = rightBorderLow;

int detxLeftBorder   = leftBorderLow;
int detxRightBorder  = rightBorderLow;
int detxTopBorder    = leftBorderHigh;
int detxBottomBorder = rightBorderHigh;

int detyLeftBorder   = leftBorderHigh;
int detyRightBorder  = rightBorderHigh;
int detyTopBorder    = leftBorderLow;
int detyBottomBorder = rightBorderLow;

int detxyLeftBorder   = leftBorderHigh;
int detxyRightBorder  = rightBorderHigh;
int detxyTopBorder    = leftBorderHigh;
int detxyBottomBorder = rightBorderHigh;
```

使用する各パラメータの意味については、ippiWTInvInitAlloc 関数の説明を参照のこと。

上記の関係からわかるように、近似画像と対角詳細画像については左と上の境界は常に同じサイズである。同じく、近似画像と対角詳細画像については、右と下の境界も同じサイズである。

したがって、各ソース・イメージに割り当てられたメモリ・ブロックのサイズは、追加された境界ピクセルに対処できるよう、ROI と正規の境界の外側にまで拡がるはずである。
水平詳細ソース・イメージの ROI と拡大された画像領域について次の図に示す。

図 13-7　ウェーブレット再構成用に拡大された水平詳細ソース・イメージ



境界ピクセルのぶんだけ拡がったソース・イメージのサイズは、次のように計算できる。

```
apprWidthWithBorders  = srcWidth  + apprLeftBorder + apprRightBorder;
apprHeightWithBorders = srcHeight + apprTopBorder  + apprBottomBorder;

detxWidthWithBorders  = srcWidth  + detxLeftBorder + detxRightBorder;
detxHeightWithBorders = srcHeight + detxTopBorder  + detxBottomBorder;

detyWidthWithBorders  = srcWidth  + detyLeftBorder + detyRightBorder;
detyHeightWithBorders = srcHeight + detyTopBorder  + detyBottomBorder;

detxyWidthWithBorders  = srcWidth  + detxyLeftBorder + detxyRightBorder;
detxyHeightWithBorders = srcHeight + detxyTopBorder  + detxyBottomBorder;
```

ROI と概念は、大きな画像をブロックごと（すなわち、「タイル」ごと）に再構成するときに適用できる。

それには、各ソース・イメージを細分し、部分的に境界が重なり合ったいくつかのブロックに分ける必要がある。これは、ippiWTFwd 関数の説明で述べたこととほぼ同じである。4 つの成分画像のそれぞれを、矩形ブロックの同じパターンに細分しなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定したポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	値が 0 または負のフィールドが *srcRoiSize* に含まれている場合のエラー状態を示す。
ippStsStepErr	バッファに含まれているステップ数が 0 以下の場合のエラー状態を示す。
ippStsContextMatchErr	不正なコンテキスト構造体へのポインタが渡された場合のエラー状態を示す。

コンピュータ・ビジョン 14

本章には、インテル® IPP ソフトウェアで使用しているコンピュータ・ビジョンのさまざまな概念の基礎知識と、コンピュータ・ビジョン用インテル IPP の各関数の詳しい説明が記載されている。インテル IPP 関数は、機能によってグループ分けをしており、各グループごとに節を分けて説明する。

表 14-1　コンピュータ・ビジョン・アプリケーション用のインテル®IPP 関数

関数名	説明
フィルタ	
BlurInitAlloc	ぼかし操作が実行できるように、たたみ込み構造体を初期化し、メモリを割り当てる。
LaplaceInitAlloc	ラプラシアン・フィルタリングが実行できるように、たたみ込み構造体を初期化し、メモリを割り当てる。
SobelInitAlloc	Sobel フィルタリングが実行できるように、たたみ込み構造体を初期化し、メモリを割り当てる。
ConvolFree	たたみ込み構造体を解放し、BlurInitAlloc、LaplaceInitAlloc、SobelInitAlloc で割り当てられたメモリをすべて解放する。
Blur	画像のぼかし操作を実行する。
Laplace	画像にラプラシアン演算子を適用する。
Scharr_Dx	x- 導関数 Scharr 演算子でソース・イメージのたたみ込みを計算する。
Scharr_Dy	y- 導関数 Scharr 演算子でソース・イメージのたたみ込みを計算する。
Sobel	任意のサイズの Sobel 演算子を使用して、画像のたたみ込みを実行する。
Sobel3x3_Dx	Sobel 3 × 3 カーネルを使用して、画像の x- 導関数を計算する。
Sobel3x3_Dy	Sobel 3 × 3 カーネルを使用して、画像の y- 導関数を計算する。
Sobel3x3_D2x	Sobel 3 × 3 カーネルを使用して、画像の二次 x- 導関数を計算する。
Sobel3x3_D2y	Sobel 3 × 3 カーネルを使用して、画像の二次 y- 導関数を計算する。
Sobel3x3_DxDy	Sobel 3 × 3 カーネルを使用して、画像の二次複合導関数を計算する。

表 14-1　コンピュータ・ビジョン・アプリケーション用のインテル®IPP 関数

関数名	説明
特徴検出関数	
CannyGetSize	関数 ippiCanny 用のテンポラリ・バッファのサイズを事前に計算する。
Canny	画像に含まれているエッジを見つける。
EigenValsVecsGetSize	関数 ippiEigenValsVecs 用のテンポラリ・バッファのサイズを事前に計算する。
EigenValsVecs	コーナー検出ができるよう、画像ブロックの固有値と固有ベクトルを計算する。
MinEigenValGetSize	関数 ippiCornerMinEigenVal 用のテンポラリ・バッファのサイズを事前に計算する。
MinEigenVal	コーナー検出ができるように、画像ブロックの最小固有値を計算する。
MatchTemplateGetBufSize	関数 MatchTemplate が使用するテンポラリ・バッファのサイズを計算する。
MatchTemplate	さまざまな画像領域と所定のテンプレートとの相似性を示す値で、生成された画像を埋める。
距離変換関数	
DistanceTransform	距離変換演算を実行する。
GetDistanceTransformMask	距離変換に用いるメトリックを計算する。
フラッド・フィル関数	
FloodFillGetSize	関数 FloodFill が使用するテンポラリ・バッファのサイズを求める。
FloodFillGetSize_Grad	関数 FloodFill_Grad が使用するテンポラリ・バッファのサイズを求める。
FloodFill	画像上で同じピクセル値を持っている連結領域を、そのピクセルの 4- 近傍値を考慮に入れて新しい値で埋める。
FloodFill_Grad	画像上で近いピクセル値を持っている連結領域を、そのピクセルの 4- 近傍値を考慮に入れて埋める。
モーション・テンプレート関数	
UpdateMotionHistory	所定のタイムスタンプにあるモーション・シルエットを使用して、モーション履歴画像を更新する。
角錐関数	
PyrDownGetBufSize	関数 PyrDown が使用するテンポラリ・バッファのサイズを計算する。
PyrUpGetBufSize	関数 PyrUp が使用するテンポラリ・バッファのサイズを計算する。
PyrDown	画像にガウシアン・フィルタを適用した後、ダウンサンプリングを実行する。
PyrUp	画像のアップサンプリングを実行した後、4 倍したガウシアン・フィルタを適用する。

ippiAdd を使用して背景の差分を計算する

本節では、背景の統計的モデルを構築するのに役立つさまざまな関数について説明する。このモデルは、画像から背景を差し引くときに使用できる。

ここで言う「背景」とは、動きのない画像のピクセルをいくつかひとまとめにしたものを指している。すなわち、カメラの前で動いているどのオブジェクトにも属していないピクセルのことである。「背景」の定義は、ほかのオブジェクト抽出法の場合には異なる場合がある。例えば、立体視法などの助けを借りて、問題のシーンの深度マップが得られる際は、そのシーンのうちカメラから遠く離れた静止部分を「背景」と定義するときもある。

最も単純な背景モデルは、すべての背景ピクセルの輝度が正規分布に従って互いに独立に変化するモデルだろう。数十個のフレームとその平方値を累積して、背景の特性が計算できる。つまり、すべてのピクセル位置について、関数 ippiAdd を使用すれば、その領域に含まれるいくつかのピクセル値の和 $S(x, y)$ が求められ、関数 ippiAddSquare を使用すれば、その各値の平方和 $Sq(x, y)$ が求められる。さらに、平均値は次の式で求められる。

$m(x, y) = \frac{S(x, y)}{N}$ N は、収集されたフレームの個数である。標準偏差は次の式で求められる。

$stddev(x, y) = sqrt\left(\frac{Sq(x, y)}{N} - \left(\frac{S(x, y)}{N}\right)^{22}\right)$ その後で、

$abs(p(x, y) - m(x, y)) < C * stddev(x, y)$ の条件を満たしている場合、特定のフレーム内の特定のピクセル領域に含まれる問題のピクセルが移動物に属しているとみなせる（式中の C は定数）。C が 3 の場合、いわゆる「3 シグマ法」を満たしている。この背景モデルを得るには、数秒間にわたってカメラの視野から各種のオブジェクトを外す必要がある。そうすれば、カメラから見た画像全体が、その後に続く背景になる。

カメラの移動する場合や、オブジェクトが別のオブジェクトの後ろを通りすぎる場合などに、照明条件や背景シーンが変化していくのに合わせて背景差分計算モデルを変えれば、前述の技法が改善できる。

平均輝度については、単純平均を、関数 ippiAddWeighted で求められる連続平均に置き換えると計算できる。また、いくつかの技法を使用して、背景情報をたくわえながら同時に、移動部分を見つけてそれを取り除ける。そうした技法には、変化検出 （第 5 章の関数 ippiAbsDiff、第 7 章の関数 ippiThreshold を参照）やオプティカル・フローなどがある。

背景差分計算に関連した加算関数には、次のものがある。

`Add_8u32f_C1IR`, `Add_8s32f_C1IR`, `Add_32f_C1IR`,
`Add_8u32f_C1IMR`, `Add_8s32f_C1IMR`, `Add_32f_C1IMR` (Add を参照),

さらに、`AddSquare`、`AddProduct`、`AddWeighted` もすべて含まれる。

フィルタ

本節では、コンピュータ・ビジョンに使用するさまざまな固定フィルタについて説明する。導関数演算子の説明が中心になる。この導関数演算子は、一次 / 二次 / 三次導関数の Sobel 演算子群と呼ばれている。

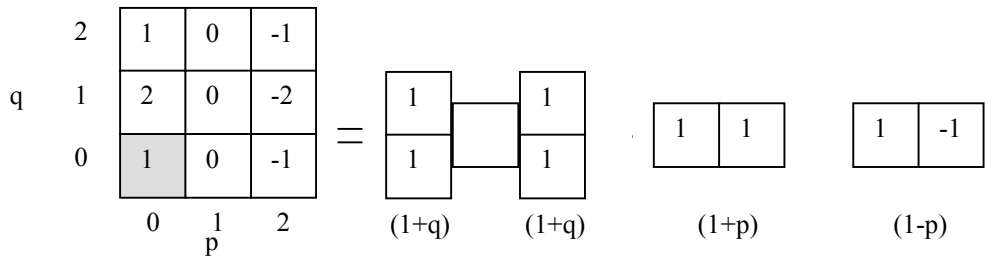
一次導関数の Sobel 演算子

一次 *x*- 導関数の Sobel 演算子は、次の多項式で表せる。

$$1 + 2q + q^2 - p^2 - 2p^2q - p^2q^2 = (1+q)^2(1-p^2) = (1+q)(1+q)(1+p)(1-p)$$

また、図 14-1 に示すように、いくつかのたたみ込みプリミティブに分解できる。図中、左下の灰色の正方形に入っている数字は、p-q 座標系の原点を示している。

図 14-1　一次 x- 導関数の Sobel 演算子



一次 *x*- 導関数の Sobel 演算子と一次 *y*- 導関数の Sobel 演算子の序列は次のように表せる。

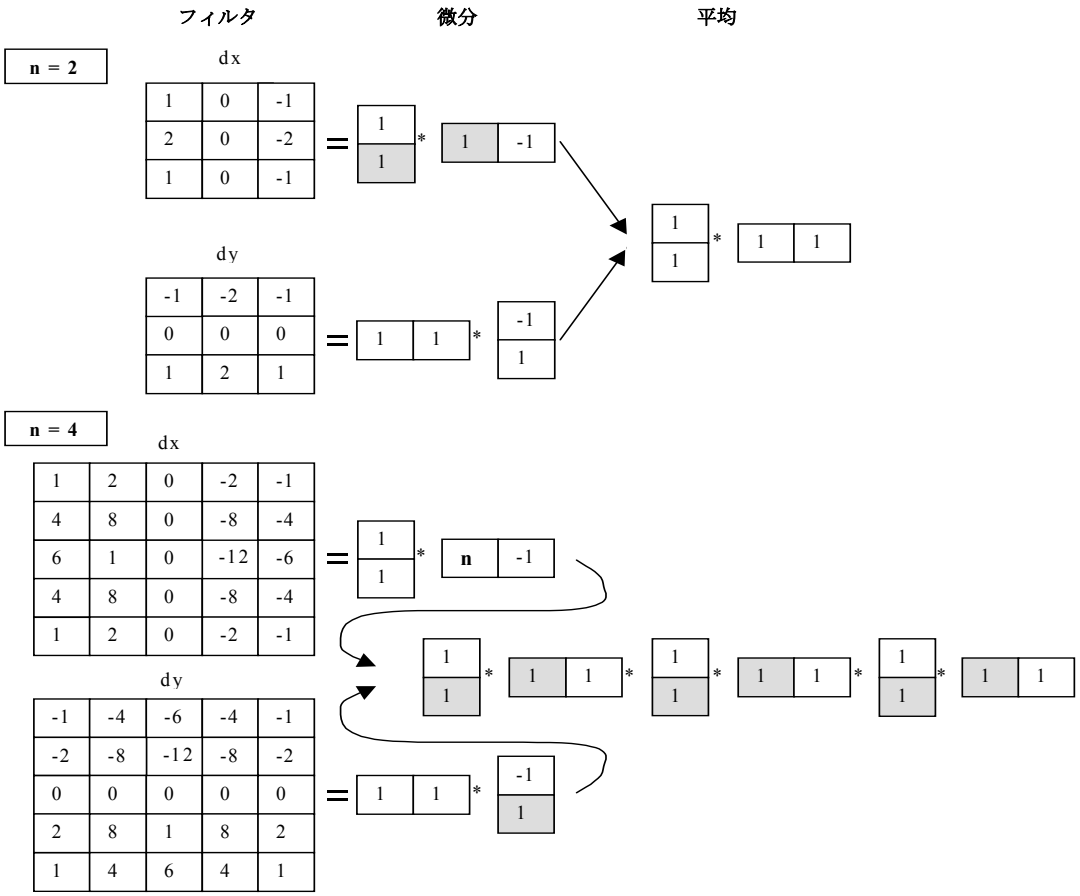
$$\frac{\partial}{\partial x} \Rightarrow (1+p)^{n-1}(1+q)^n(1-p)\ , \qquad (1\text{-}1)$$

$$\frac{\partial}{\partial y} \Rightarrow (1+p)^n(1+q)^{n-1}(1-q) \qquad (1\text{-}2)$$

ただし、$n > 0$ である。

図 14-2 は、$n = 2$、4 の場合について、式 1-1 と式 1-2 の一次導関数の Sobel 演算子を、「加算 - 減算」プリミティブのいくつかのたたみ込み演算子に分解するときの図である。

図 14-2　一次導関数の Sobel 演算子 （n = 2、4 の場合）



二次導関数の Sobel 演算子

二次導関数の Sobel 演算子は、式 1-1 および式 1-2 に似た多項式分解で表せる。二次導関数の式を以下に示す。

$$\frac{\partial^2}{\partial x^2} \Rightarrow (1+p)^{n-2}(1+q)^n(1-p)^2 \ , \tag{1-3}$$

$$\frac{\partial^2}{\partial y^2} \Rightarrow (1+p)^n(1+q)^{n-2}(1-q)^2 \ , \tag{1-4}$$

$$\frac{\partial^2}{\partial x \partial y} \Rightarrow (1+p)^{n-1}(1+q)^{n-1}(1-p)(1-q) \tag{1-5}$$

ただし、$n = 2$、3、... である。

多項式の表現形式からわかるように、図 14-3 は、$n = 2$、4 の場合について、「加算 - 減算」プリミティブの切り離し可能ないくつかのたたみ込み演算子に分解できるフィルタを表している。多項式分解については各演算子の上に示してある。

図 14-3　二次導関数の Sobel 演算子 （n = 2、4 の場合）

n = 2

$\delta^2/\delta x^2 = (1+q)^2(1-p)^2$

1	-2	1
2	-4	2
1	-2	1

$\delta^2/\delta y^2 = (1+p)^2(1-q)^2$

1	2	1
-2	-4	-2
1	2	1

$\delta^2/\delta x\delta y = (1+q)(1+p)(1-q)(1-p)$

-1	0	1
0	0	0
1	0	-1

n = 4

$\delta^2/\delta x^2 = (1+p)^2(1+q)^4(1-p)^2$

1	0	-2	0	1
4	0	-4	0	4
6	0	-12	0	6
4	0	-8	0	4
1	0	-2	0	1

$\delta^2/\delta y^2 = (1+q)^2(1+p)^4(1-q)^2$

1	4	6	4	1
0	0	0	0	0
-2	-8	-12	-8	-2
0	0	0	0	0
1	4	6	4	1

$\delta^2/\delta x\delta y = (1+p)^3(1+q)^3(1-p)(1-q)$

-1	-2	0	2	1
-2	-4	0	4	2
0	0	0	0	0
2	4	0	-4	-2
1	2	0	-2	-1

三次導関数の Sobel 演算子

三次導関数の Sobel 演算子は、以下の多項式分解形式で表せる。

$$\frac{\partial^3}{\partial x^3} \Rightarrow (1+p)^{n-3}(1+q)^n(1-p)^3 \quad (1\text{-}6)$$

$$\frac{\partial^3}{\partial y^3} \Rightarrow (1+p)^n(1+q)^{n-3}(1-q)^3 \quad (1\text{-}7)$$

$$\frac{\partial^3}{\partial x^2 \partial y} \Rightarrow (1-p)^2(1+p)^{n-2}(1+q)^{n-1}(1-q) \quad (1\text{-}8)$$

$$\frac{\partial^3}{\partial x \partial y^2} \Rightarrow (1-p)(1+p)^{n-1}(1+q)^{n-2}(1-q)^2 \quad (1\text{-}9)$$

ただし、$n = 3$、4、... である。

三次導関数フィルタが適用できるのは、$n = 4$ の場合と、一般的な場合のみである。

ラプラシアン近似

次に示すように、二次導関数 x および y の和により、ラプラシアンの近似値が得られる。

$$L = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \quad (1\text{-}10)$$

したがって、この節で定義する二次導関数を表す式を使用して、画像のラプラシアンが計算できる。

BlurInitAlloc

ぼかし操作ができるよう、たたみ込み構造体を初期化し、メモリを割り当てる。

```
IppStatus ippiBlurInitAlloc(int roiWidth, IppDataType dataType, int
    kerSize, IppiConvState** ppState);
```

引数

roiWidth	ソース・イメージの ROI の幅（ピクセル単位）
dataType	ソース・イメージのデータ・タイプ
kerSize	ぼかしウィンドウのサイズ
ppState	初期化するたたみ込み構造体への返されるポインタのポインタ

説明

関数 ippiBlurInitAlloc は、ippcv.h ファイルの中で宣言される。この関数は、ぼかし操作が実行できるように、たたみ込み構造体を初期化し、メモリを割り当てる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*ppState* ポインタ が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiWidth* の値が 0 または負のとき、または *kerSize* が 3 未満か偶数の場合のエラー状態を示す。
ippStsDataTypeErr	*dataType* が Ipp8u、Ipp8s、Ipp32f のいずれにも等しくない場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

LaplaceInitAlloc

ラプラシアン・フィルタリングが実行できるように、たたみ込み構造体を初期化し、メモリを割り当てる。

```
IppStatus ippiLaplaceInitAlloc(int roiWidth, IppDataType dataType, int
    kerSize, IppiConvState** ppState);
```

引数

roiWidth	ソース・イメージの ROI の幅（ピクセル単位）
dataType	ソース・イメージのデータ・タイプ
kerSize	カーネルのサイズ
ppState	初期化するたたみ込み構造体への返されるポインタのポインタ

説明

関数 ippiLaplaceInitAlloc は、ippcv.h ファイルの中で宣言される。この関数は、ラプラシアン・フィルタリングが実行できるように、たたみ込み構造体を初期化し、メモリを割り当てる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*ppState* ポインタ が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiWidth* の値が 0 または負のとき、または *kerSize* が 3 未満か偶数の場合のエラー状態を示す。
ippStsDataTypeErr	*dataType* が Ipp8u、Ipp8s、Ipp32f のいずれにも等しくない場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

SobelInitAlloc

Sobel フィルタリングが実行できるように、たたみ込み構造体を初期化し、メモリを割り当てる。

```
IppStatus ippiSobelInitAlloc(int roiWidth, IppDataType dataType,
      int kerSize, int origin, int dx, int dy, IppiConvState** ppState);
```

引数

roiWidth	ソース・イメージの ROI の幅（ピクセル単位）
dataType	ソース・イメージのデータ・タイプ
kerSize	カーネルのサイズ
origin	画像の原点。次のいずれかになる。 IPCV_ORIGIN_TL　画像の左上隅 IPCV_ORIGIN_BL　画像の左下隅
dx	*x*- 導関数の次数
dy	*y*- 導関数の次数
ppState	初期化するたたみ込み構造体への返されるポインタのポインタ

説明

関数 ippiSobelInitAlloc は、ippcv.h ファイルの中で宣言される。この関数は、Sobel フィルタリングが実行できるよう、たたみ込み構造体を初期化し、メモリを割り当てる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*ppState* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiWidth*、*dx*、*dy* のいずれかの値が 0 または負の場合、または *kerSize* が偶数か 3 未満の場合のエラー状態を示す。
ippStsDataTypeErr	*dataType* が Ipp8u、Ipp8s、Ipp32f のいずれにも等しくない場合のエラー状態を示す。
ippStsMemAllocErr	メモリ割り当てに失敗した場合のエラー状態を示す。

ConvolFree

たたみ込み構造体を解放し、BlurInitAlloc、LaplaceInitAlloc、SobelInitAlloc によって割り当てられたメモリをすべて解放する。

```
IppStatus ippiConvolFree(ippConvState** ppState);
```

引数

ppState	たたみ込み構造体へのポインタのポインタ

説明

関数 ippiConvolFree は、ippcv.h ファイルの中で宣言される。この関数は、たたみ込み構造体を解放し、BlurInitAlloc、LaplaceInitAlloc、SobelInitAlloc の各関数によって割り当てられたメモリをすべて解放する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*ppState* ポインタ が NULL の場合のエラー状態を示す。

Blur

ぼかし用カーネルでソース・イメージのたたみ込みを計算する。

```
IppStatus ippiBlur_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
     Ipp<dstDatatype>* pDst, int dstStep, IppiSize* pRoiSize,
     IppiConvState* pState, int stage);
```

サポートされる *mod* の値は、次のとおりである。

```
8u16s_C1R    8s16s_C1R    32f_C1R
```

引数

pSrc	ソース・イメージへのポインタ

srcStep	ソース・イメージ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
pRoiSize	現在の画像サイズで初期化される ROI 構造体へのポインタ。この関数によって、取得された行数が *pRoiSize* に格納される。
pState	たたみ込み構造体へのポインタ
stage	処理ステージ。次のいずれかになる。

IPPI_WHOLE	関数は画像全体を処理する。
IPPI_START	関数は循環バッファを初期化し、大きな画像の上部を処理する。生成される部分はソース部分より小さくなるが、構造化要素の高さが 1 の場合は、ソース部分と同じサイズになる。
IPPI_END	関数は循環バッファをフラッシュし、大きな画像の下部を処理する。生成される部分はソース部分より大きくなるが、構造化要素の高さが 1 の場合は、ソース部分と同じサイズになる。
IPPI_MIDDLE	関数は大きな画像の中間部分を処理する。生成される部分は、ソース部分と同じサイズになる。

説明

関数 ippiBlur は、ippcv.h ファイルの中で宣言される。この関数は、ぼかし用のカーネルでソース・イメージ *pSrc* のたたみ込みを計算し、その結果を *pDst* に格納する。このカーネルは、すべての要素が 1 である次のような行列である。

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* が *pRoiSize.width* <pixelSize>* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsBadArgErr	*stage* が IPP_START、IPP_MIDDLE、IPP_END、IPP_WHOLE のいずれにも等しくない場合のエラー状態を示す。

Laplace

ラプラシアン・カーネルでソース・イメージのたたみ込みを計算する。

```
IppStatus ippiLaplace_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
     Ipp<dstDatatype>* pDst, int dstStep, IppiSize* pRoiSize,
     IppiConvState* pState, int stage);
```

サポートされる *mod* の値は、次のとおりである。

```
8u16s_C1R    8s16s_C1R    32f_C1R
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
pRoiSize	現在の画像サイズで初期化される ROI 構造体へのポインタ。この関数によって、取得された行数が *pRoiSize* に格納される。
pState	たたみ込み構造体へのポインタ

stage	処理ステージ。次のいずれかになる。
	IPPI_WHOLE　関数は画像全体を処理する。
	IPPI_START　関数は循環バッファを初期化し、大きな画像の上部を処理する。生成される部分はソース部分より小さくなるが、構造化要素の高さが 1 の場合は、ソース部分と同じサイズになる。
	IPPI_END　関数は循環バッファをフラッシュし、大きな画像の下部を処理する。生成される部分はソース部分より大きくなるが、構造化要素の高さが 1 の場合は、ソース部分と同じサイズになる。
	IPPI_MIDDLE　関数は大きな画像の中間部分を処理する。生成される部分は、ソース部分と同じサイズになる。

説明

関数 ippiLaplace は、ippcv.h ファイルの中で宣言される。この関数は、「ラプラシアン近似」で述べたように、ソース・イメージ *pSrc* のラプラシアンを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* が *pRoiSize.width***<pixelSize>* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsBadArgErr	*stage* が IPP_START、IPP_MIDDLE、IPP_END、IPP_WHOLE のいずれにも等しくない場合のエラー状態を示す。

Scharr_Dx

x- 導関数の Scharr 演算子でソース・イメージのたたみ込みを計算する。

```
IppStatus ippiScharr_Dx_<mod>(const Ipp<srcDatatype>* pSrc,
     int srcStep,  Ipp<dstDatatype>* pDst, int dstStep,
     IppiSize roiSize, int shift);
```

サポートされる *mod* の値は、次のとおりである。

```
8u16s_C1R    8s16s_C1R    32f_C1R
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）
shift	シフト操作でのシフト値

説明

関数 ippiScharr_Dx は、ippcv.h ファイルの中で宣言される。この関数は、次に示すように、x- 導関数 Scharr カーネルでソース・イメージ *pSrc* のたたみ込みを計算する。

$$\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}$$

このフィルタは、画像の垂直エッジを増す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。

ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* が *pRoiSize.width***<pixelSize>* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。

Scharr_Dy

y- 導関数の Scharr 演算子でソース・イメージのたたみ込みを計算する。

```
IppStatus ippiScharr_Dy_<mod>(const Ipp<srcDatatype>* pSrc,
      int srcStep,  Ipp<dstDatatype>* pDst, int dstStep, int origin,
      IppiSize roiSize, int shift);
```

サポートされる *mod* の値は、次のとおりである。

```
8u16s_C1R    8s16s_C1R    32f_C1R
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）
origin	画像の原点。次のいずれかになる。 IPCV_ORIGIN_TL　画像の左上隅 IPCV_ORIGIN_BL　画像の左下隅
shift	シフト操作でのシフト値

説明

関数 ippiScharr_Dy は、ippcv.h ファイルの中で宣言される。この関数は、次に示すように、y- 導関数 Scharr カーネルでソース・イメージ *pSrc* のたたみ込みを計算する。

$$\begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}$$

このフィルタは、画像の垂直エッジを増す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* が *pRoiSize.width* <pixelSize>* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsBadArgErr	*origin* が IPCV_ORIGIN_TL にも IPCV_ORIGIN_BL にも等しくない場合のエラー状態を示す。

Sobel

任意のサイズの Sobel カーネルでソース・イメージのたたみ込みを計算する。

```
IppStatus ippiSobel_<mod>(const Ipp<srcDataType>* pSrc, int srcStep,
     Ipp<dstDataType>* pDst, int dstStep, IppiSize* pRoiSize,
     IppiConvState* pState, int stage);
```

サポートされる *mod* の値は、次のとおりである。

```
8u16s_C1R    8s16s_C1R    32f_C1R
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
pRoiSize	現在の画像サイズで初期化される ROI 構造体へのポインタ。この関数によって、取得された行数が *pRoiSize* に格納される。
pState	たたみ込み構造体へのポインタ
stage	処理ステージ。次のいずれかになる。 IPPI_WHOLE　関数は画像全体を処理する。 IPPI_START　関数は循環バッファを初期化し、大きな画像の上部を処理する。生成される部分はソース部分より小さくなるが、構造化要素の高さが 1 の場合は、ソース部分と同じサイズになる。 IPPI_END　関数は循環バッファをフラッシュし、大きな画像の下部を処理する。生成される部分はソース部分より大きくなるが、構造化要素の高さが 1 の場合は、ソース部分と同じサイズになる。

IPPI_MIDDLE　関数は大きな画像の中間部分を処理する。生成される部分は、ソース部分と同じサイズになる。

説明

関数 ippiSobel は、ippcv.h ファイルの中で宣言される。この関数は、ユーザが定義したサイズの Sobel 演算子でソース・イメージ *pSrc* のたたみ込みを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* が *pRoiSize.width* <pixelSize>* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsBadArgErr	*stage* が IPP_START、IPP_MIDDLE、IPP_END、IPP_WHOLE のいずれにも等しくない場合のエラー状態を示す。

Sobel3x3_Dx

一次 x- 導関数の Sobel 演算子でソース・イメージのたたみ込みを計算する。

```
IppStatus ippiSobel3x3_Dx_<mod>(const Ipp<srcDataType>* pSrc, int
srcStep, Ipp<dstDataType>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

8u16s_C1R　8s16s_C1R　32f_C1R

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）

説明

関数 ippiSobel3x3_Dx は、ippcv.h ファイルの中で宣言される。この関数は、一次 x- 導関数 Sobel 演算子でソース・イメージ *pSrc* のたたみ込みを計算する。この演算子は、以下の値を持つ 3 × 3 サイズのマトリックスである。

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

このフィルタは、画像の垂直エッジを増す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* が *pRoiSize.width* <pixelSize>* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。

Sobel3x3_Dy

一次 y- 導関数の Sobel 演算子でソース・イメージのたたみ込みを計算する。

```
IppStatus ippiSobel3x3_Dy_<mod>(const Ipp<srcDataType>* pSrc, int srcStep, Ipp<dstDataType>* pDst, int dstStep, int origin IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u16s_C1R    8s16s_C1R    32f_C1R
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）
origin	画像の原点。次のいずれかになる。 IPCV_ORIGIN_TL　画像の左上隅 IPCV_ORIGIN_BL　画像の左下隅

説明

関数 ippiSobel3x3_Dy は、ippcv.h ファイルの中で宣言される。この関数は、一次 y- 導関数 Sobel 演算子でソース・イメージ *pSrc* のたたみ込みを計算する。この演算子は、以下の値を持つ 3 × 3 サイズのマトリックスである。

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

このフィルタは、画像の水平エッジを増す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* が *pRoiSize.width* <pixelSize>* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsBadArgErr	*origin* が IPCV_ORIGIN_TL にも IPCV_ORIGIN_BL にも等しくない場合のエラー状態を示す。

Sobel3x3_D2x

二次 x- 導関数の Sobel 演算子でソース・イメージのたたみ込みを計算する。

```
IppStatus ippiSobel3x3_D2x_<mod>(const Ipp<srcDataType>* pSrc, int
     srcStep, Ipp<dstDataType>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u16s_C1R    8s16s_C1R    32f_C1R
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）

説明

関数 ippiSobel3x3_D2x は、ippcv.h ファイルの中で宣言される。この関数は、二次 x- 導関数の Sobel 演算子でソース・イメージ *pSrc* のたたみ込みを計算する。この演算子は、以下の値を持つ 3 × 3 サイズのマトリックスである。

$$\begin{bmatrix} 1 & -2 & 1 \\ 2 & -4 & 2 \\ 1 & -2 & 1 \end{bmatrix}$$

このフィルタは、画像の垂直エッジを増す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* が *pRoiSize.width* <pixelSize>* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。

Sobel3x3_D2y

二次 y- 導関数の垂直 Sobel 演算子でソース・イメージのたたみ込みを計算する。

```
IppStatus ippiSobel3x3_D2y_<mod>(const Ipp<srcDataType>* pSrc, int
     srcStep, Ipp<dstDataType>* pDst, int dstStep, IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u16s_C1R    8s16s_C1R    32f_C1R
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）

説明

関数 ippiSobel3x3_D2y は、ippcv.h ファイルの中で宣言される。この関数は、二次 y- 導関数の Sobel 演算子でソース・イメージ *pSrc* のたたみ込みを計算する。この演算子は、以下の値を持つ 3 × 3 サイズのマトリックスである。

$$\begin{bmatrix} 1 & 2 & 1 \\ -2 & -4 & -2 \\ 1 & 2 & 1 \end{bmatrix}$$

このフィルタは、画像の水平エッジを増す。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* が *pRoiSize.width* <pixelSize>* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。

Sobel3x3_DxDy

二次複合導関数の Sobel 演算子でソース・イメージのたたみ込みを計算する。

```
IppStatus ippiSobel3x3_DxDy_<mod>(const Ipp<srcDataType>* pSrc,
      int srcStep, Ipp<dstDataType>* pDst, int dstStep, int origin,
      IppiSize roiSize);
```

サポートされる *mod* の値は、次のとおりである。

```
8u16s_C1R    8s16s_C1R    32f_C1R
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）
origin	画像の原点。次のいずれかになる。 IPCV_ORIGIN_TL　画像の左上隅 IPCV_ORIGIN_BL　画像の左下隅

説明

関数 ippiSobel3x3_DxDy は、ippcv.h ファイルの中で宣言される。この関数は、次のカーネルを持つ二次複合導関数の Sobel 演算子でソース・イメージ *pSrc* のたたみ込みを計算する。

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* が *pRoiSize.width* <pixelSize>* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsBadArgErr	*origin* が IPCV_ORIGIN_TL にも IPCV_ORIGIN_BL にも等しくない場合のエラー状態を示す。

特徴検出関数

本節では、特徴検出関数について説明する。一般的に、エッジ、リッジ、ブロブを検出するときには、一連の Sobel 導関数フィルタを使用する。特に、角錐などのメトリック空間画像の場合はこれらのフィルタを使用する。

以下にいくつか式を示すが、その記号の意味は次のとおりである。

- D_x および D_y は、それぞれ、一次 *x*- 導関数と一次 *y*- 導関数である。
- D_{xx} および D_{yy} は、それぞれ、二次 *x*- 導関数と二次 *y*- 導関数である。
- D_{xy} は、*x* と *y* の偏導関数である。
- D_{xxx} および D_{yyy} は、それぞれ、三次 *x*- 導関数と三次 *y*- 導関数である。
- D_{xxy} および D_{xyy} は、*x* と *y* の三次偏導関数である。

コーナー検出

Sobel の一次導関数演算子を使用して、画像の *x*- 導関数と *y*- 導関数を調べる。次に、せまい処理対象領域を定義し、そこに含まれているコーナーを検出する。

x および *y* の導関数の和の 2 × 2 行列は、次のようになる。

$$C = \begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix} \tag{1.11}$$

$det(C - \lambda I) = 0$ を解くと、固有値がいくつか得られる。この λ は、いくつかの固有値のうちの列ベクトルの 1 つであり、*I* は単位行列である。上の式の 2 × 2 行列の場合は、次のような閉じた形式で解をいくつか書き込める。

$$\lambda = \frac{\Sigma D_x^2 + \Sigma D_y^2 \pm \sqrt{(\Sigma D_x^2 + \Sigma D_y^2)^2 - 4 \cdot (\Sigma D_x^2 \Sigma D_y^2 - (\Sigma D_x D_y)^2)}}{2} \tag{1.12}$$

$\lambda_1, \lambda_2 > t$ の場合は、その位置にコーナーが 1 つあると考えられ、*t* はしきい値である。オブジェクト認識や形状認識のとき特に有効な方法である。

Canny エッジ検出法

本節では、J.Canny が提唱した、古典的なエッジ検出法について述べる （[Canny86] を参照）。この検出法では、入力としてグレー・スケール画像を 1 枚使用し、白黒画像が 1 枚出力される。出力された画像には、非ゼロ・ピクセルによって、検出された各エッジに印が付く。このアルゴリズムは、次の 3 つの段階から構成される。

第 1 段階：微分

二次元のたたみ込みの場合は、画像データを *x* 方向と *y* 方向に微分する。任意の方向にたたみ込まれた画像関数の表面の勾配については、任意の 2 方向の既知の傾きから計算できる。

計算された *x* と *y* の傾き値を使用して、その斜辺とアークタンジェントから、傾斜の振幅と角度が求められる。

注： この第 1 段階を実行するのが ippiSobel の各関数で、その出力を、Canny エッジ検出法の各関数が使用する。

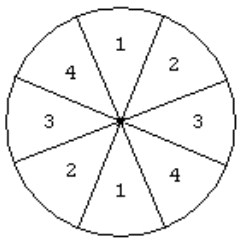
第 2 段階：極大点以外を隠す

画像の各点での輝度の変化率に応じて、いくつかの極大点に各エッジを配置しなければならない。もっと正確に言えば、極大点以外は隠さなければならない。勾配関数の頂点の 1 つに局所極大が発生する。その一方で、勾配関数の導関数を 0 に設定する。ただし、この場合は、極大点以外については、エッジ方向に対して平行に隠すよりも、エッジ方向に対して垂直に隠したほうがよい。エッジ強度は、拡がった輪郭に沿って切れずに続いている場合が予測されるからである。

このアルゴリズムは、傾斜角を、図 14-4 に示した 4 つのセクタのうちの 1 つに変形することから始まる。次に、3 × 3 近傍を振幅の配列に渡す。各点で、近傍の中心要素と、その 2- 近傍を、セクタの値から与えられた傾斜線に沿って比較する。

中心の値が極大値以外の場合（すなわち、その近傍より大きくない場合）、消える。

図 14-4　勾配セクタ



第 3 段階：エッジのしきい値操作

Canny 演算子では、いわゆる「ヒステリシス」の特性を持ったしきい値操作を行う。しきい値は 1 つだけ設定する場合が多いが、その場合は、各エッジの値がしきい値をまたいで上下にゆれていると、線が途切れ途切れに見える。一般的に、この現象を「ストリーキング」と呼ぶ。ヒステリシス特性を持ったしきい値、すなわちエッジの上限値と下限値を組にして設定すれば、ストリーキングを打ち消せる。1 本の線分を考えてみると、上限値を上回っている値については、ただちに有効な値と見なす。下限値を下回っている値については、ただちに無効と見なす。上限値と下限値との間にある各点は、反応の強いピクセルに連結していれば、有効な点であると見なす。この方法では、ストリーキングの発生する可能性が激減する。その理由は、線分の各点が上限値と下限値を両方またいで上下にゆれていない限りストリーキングが発生しないからである。

J.Canny は、予測される信号対雑音比をもとにして、上限値 : 下限値の比を 2 ～ 3:1 にするのを推奨している。

CannyGetSize

関数 ippiCanny 用のテンポラリ・バッファのサイズを計算する。

```
IppStatus ippiCannyGetSize(IppiSize roiSize, int* pBufSize);
```

引数

roiSize	画像の ROI のサイズ（ピクセル単位）
pBufSize	テンポラリ・バッファのサイズを返す変数へのポインタ

説明

関数 ippiCannyGetSize は、ippcv.h ファイルの中で宣言される。この関数は、ippiCanny 用のテンポラリ・バッファのサイズを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pBufSize* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。

Canny

エッジ検出ができるように Canny アルゴリズムを実装する。

```
IppStatus ippiCanny_16s8u_C1R(Ipp16s* pSrcDx, int srcDxStep,
      Ipp16s* pSrcDy, int srcDyStep, Ipp8u* pDstEdges, int dstEdgeStep,
      IppiSize roiSize, Ipp32f lowThresh, Ipp32f highThresh,
      void* pBuffer);
```

引数

pSrcDx	ソース・イメージの *x*- 導関数へのポインタ
srcDxStep	ソース・イメージ *pSrcDx* 内のステップ（バイト単位）
pSrcDy	ソース・イメージの *y*- 導関数へのポインタ
srcDyStep	ソース・イメージ *pSrcDy* 内のステップ（バイト単位）
pDstEdges	検出されたエッジの出力配列へのポインタ
dstEdgeStep	出力画像内のステップ（バイト単位）
roiSize	ソース・イメージの ROI のサイズ（ピクセル単位）
lowThresh	エッジを検出するための下側しきい値
highThresh	エッジを検出するための上側しきい値
pBuffer	事前に割り当てられたテンポラリ・バッファへのポインタ

説明

関数 ippiCanny は、ippcv.h ファイルの中で宣言される。この関数は、Canny アルゴリズムを使用して、ソース・イメージに含まれているエッジを検出し、それを出力画像 *pDstEdges* に保存する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcDxStep*、*srcDyStep*、または *dstEdgeStep* のいずれかが *roi.width* <pixelSize>* より小さい場合のエラー状態を示す。
ippStsBadArgErr	*lowThresh* が負の場合、または *highThresh* が *lowThresh* より小さい場合のエラーを示す。

EigenValsVecsGetSize

関数 ippiEigenValsVecs 用のテンポラリ・バッファのサイズを計算する。

```
IppStatus ippiEigenValsVecsGetSize(int roiWidth, int apertureSize, int
avgWindow, int* pBufSize);
```

引数

roiWidth	ソース・イメージの ROI の幅（ピクセル単位）
apertureSize	この関数が使用する派生演算子のサイズ
avgWindow	ぼかしウィンドウのサイズ
pBufSize	テンポラリ・バッファのサイズを返す変数へのポインタ

説明

関数 ippiEigenValsVecsGetSize は、ippcv.h ファイルの中で宣言される。この関数は、関数 ippiEigenValsVecs が使用するテンポラリ・バッファのサイズを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合、または *apertureSize* または *avgWindow* のいずれかが偶数か 3 未満の場合のエラー状態を示す。

EigenValsVecs

コーナー検出ができるように、画像ブロックの固有値と固有ベクトルを計算する。

```
IppStatus ippiEigenValsVecs_8u32f_C1R(const Ipp<srcDataType>* pSrc,
     int srcStep, Ipp32f* pEigenVV, int eigStep, IppiSize roiSize, int
     apertureSize, int avgWindow, void* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
8u32f_C1R    8s32f_C1R    32f_C1R
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pEigenVV	計算結果の保存先である画像
eigStep	出力画像内のステップ（バイト単位）
roiSize	ソース・イメージの ROI のサイズ（ピクセル単位）
apertureSize	導関数演算子のサイズ
avgWindow	ぼかしウィンドウのサイズ

pBuffer	テンポラリ・バッファへのポインタ

説明

関数 ippiEigenValsVecs は、ippcv.h ファイルの中で宣言される。この関数は、注目ピクセルの周辺にあるブロックを 1 つ取り込み、一次導関数 *Dx* と *Dy* を計算する。この計算は画像のピクセルすべてに実行される。次に、以下の行列の固有値と固有ベクトルを計算する。

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix}$$

この加算はブロック全体に対して実行される。

画像 *eigenVV* は、次の形式になっている。ソース・イメージの各ピクセルはいずれも、λ1、λ2、*x*1、*y*1、*x*2、*y*2 の 6 つの浮動小数点値を含んでいる。これらの値は、次のように定義している。

λ1, λ2	上の行列の固有値（値順には並んでいない）
*x*1, *y*1	λ1 に対応する、正規化された固有ベクトルの座標
*x*2, *y*2	λ2 に対応する、正規化された固有ベクトルの座標

特異行列の場合、あるいは固有値の 1 つが次の固有値よりかなり小さい場合は、これらの 6 つの値がすべて 0 に設定される。

Sobel 演算子は微分に使用する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合、または *apertureSize* または *avgWindow* のいずれかが偶数か 3 未満の場合のエラー状態を示す。
ippStsStepErr	*srcStep* が *roiSize.width*<*pixelSize*> より小さいか、*eigStep* が *roiSize.width***sizeof*（*Ipp32f*）**6* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。

MinEigenValGetSize

関数 MinEigenVal 用のテンポラリ・バッファのサイズを計算する。

```
IppStatus ippiMinEigenValGetSize(int roiWidth, int apertureSize, int avgWindow, int* pBufSize);
```

引数

roiWidth	ソース・イメージの ROI の幅（ピクセル単位）
apertureSize	この関数が使用する派生演算子のサイズ
avgWindow	ぼかしウィンドウのサイズ
pBufSize	テンポラリ・バッファのサイズを返す変数へのポインタ

説明

関数 ippiMinEigenValGetSize は、ippcv.h ファイルの中で宣言される。この関数は、関数 MinEigenVal が使用するテンポラリ・バッファのサイズを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合、または *apertureSize* または *avgWindow* のいずれかが偶数か 3 未満の場合のエラー状態を示す。

MinEigenVal

コーナー検出ができるように、画像ブロックの最小固有値を計算する。

```
IppStatus ippiMinEigenVal_<mod>(const Ipp<srcDataType>* pSrc, int srcStep,
     Ipp32f* pMinEigenVal, int minValStep, IppiSize roiSize,
     int apertureSize, int avgWindow, void* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
8u32f_C1R    8s32f_C1R    32f_C1R
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pMinEigenVal	最小固有値の入った画像へのポインタ
minValStep	出力画像内のステップ（バイト単位）
roiSize	ソース・イメージの ROI のサイズ（ピクセル単位）
apertureSize	導関数演算子のサイズ
avgWindow	平均をとるウィンドウ
pBuffer	テンポラリ・バッファへのポインタ

説明

関数 ippiMinEigenVal は、ippcv.h ファイルの中で宣言される。この関数は、注目ピクセルの周辺にあるブロックを 1 つ取り込み、一次導関数 *Dx* と *Dy* を計算する。この計算は画像のピクセルすべてに実行される。次に、以下の行列の最小固有値が計算する。

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix}$$

この加算はブロック全体に対して実行される。

Sobel 演算子は微分に使用する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*pRoiSize* フィールドの値が 0 または負の場合、あるいは *apertureSize* または *avgWindow* が偶数か 3 未満の場合のエラー状態を示す。
ippStsStepErr	*srcStep* が *roiSize.width*<*pixelSize*> より小さいか、*eigenvvStep* が *roiSize.width***sizeof*(*Ipp32f*) より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。

MatchTemplateGetBufSize

関数 MatchTemplate 用のテンポラリ・バッファのサイズを計算する。

```
IppStatus  ippiMatchTemplateGetBufSize_<method>(IppiSize roiSize,
    IppiSize templSize, IppDataType dataType, int* pBufferSize);
```

サポートしている <method> の値は次のとおりである。

SqDiff	Corr	Coeff
SqDiffNorm	CorrNormed	CoeffNormed

引数

roiSize	ソース・イメージの ROI のサイズ（ピクセル単位）
templSize	テンプレートのサイズ
dataType	ソース・イメージとテンプレート・データのデータ・タイプ
pBufferSize	テンポラリ・バッファのサイズを返す変数へのポインタ

説明

関数 ippiMatchTemplateGetBufSize は、ippcv.h ファイルの中で宣言される。この関数は、関数 MatchTemplate の対応する種類が使用するテンポラリ・バッファの最小サイズを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*templSize* フィールドの値が 0 または負の場合、あるいは ROI のサイズ（*roiSize.width* × *roiSize.height*）がテンプレートのサイズ（*W* × *H*）より小さい場合のエラー状態を示す。
ippStsDataTypeErr	*dataType* が Ipp8u、Ipp8s、Ipp32f のいずれでもない場合のエラーを示す。

MatchTemplate

さまざまな画像領域と所定のテンプレートの相似性を示す値で、生成された画像を埋める。

```
IppStatus ippiMatchTemplate_<method>_<mod>(const Ipp<srcDataType>*
     pImage, int imageStep, IppiSize roiSize, const Ipp<srcDataType>*
     pTemplate, int templStep, IppiSize templSize, Ipp32f* pResult,
     int resultStep, void* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
8u32f_C1R    8s32f_C1R    32f_C1R
```

サポートされる <method> の値は、次のとおりである

```
SqDiff          Corr            Coeff
SqDiffNorm      CorrNormed      CoeffNormed
```

引数

pImage	検索が実行される画像の ROI へのポインタ
imageStep	画像内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）
pTempl	検索対象の ROI のうち*pImage*より大きくなれないROIへのポインタ
templStep	テンプレート内のステップ（バイト単位）
templSize	テンプレートの ROI のサイズ
pResult	生成された画像へのポインタ。画像の ROI のサイズを $W \times H$ とし、テンプレートの ROI のサイズを $w \times h$ にすると、出力画像の ROI のサイズは（$W - w + 1$）×（$H - h + 1$）である。
pBuffer	MatchTemplateGetBufSize で最小サイズが計算されたテンポラリ・バッファへのポインタ

説明

関数 ippiMatchTemplate は、ippcv.h ファイルの中で宣言される。この関数は、画像に含まれているさまざまな領域の検出ができるように、特定のテンプレートに似た一連のメソッドを実装している。

ソース・イメージのサイズを $W \times H$ ピクセルとし、テンプレートのサイズを $w \times h$ ピクセルとすると、生成される画像のサイズは（$W - w + 1$）×（$H - h + 1$）ピクセルとなり、各点（x, y）でのピクセル値を見ると、テンプレートと画像矩形の相似性がわかる。このとき、左上の座標は（x, y）であり、右下の座標は（$x + w - 1, y + h - 1$）である。相似性は、以下に列挙したいくつかの方法で計算できる。

平方差（method = SqDiff）：

$$S(x,y) = \sum_{y'=0}^{h-1} \sum_{x'=0}^{w-1} [T(x',y') - I(x+x',y+y')]^2 ,$$

I（*x*,*y*）は、座標（*x*,*y*）にある画像ピクセル値である。*T*（*x*,*y*）は、座標（*x*,*y*）にあるテンプレートのピクセル値である。

正規化平方差（`method = SqDiffNormed`）：

$$S(x,y) = \frac{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}[T(x',y') - I(x+x',y+y')]^2}{\sqrt{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}T(x',y')^2 \sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}I(x+x',y+y')^2}}$$

相互相関（`method = Corr`）：

$$C(x,y) = \sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}T(x',y')\cdot I(x+x',y+y')$$

正規化相互相関（`method = CorrNormed`）：

$$\tilde{C}(x,y) = \frac{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}T(x',y')\cdot I(x+x',y+y')}{\sqrt{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}T(x',y')^2 \sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}I(x+x',y+y')^2}}$$

相関係数（`method = Coeff`）：

$$R(x,y) = \sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}(T(x',y') - \bar{T})\cdot(I(x+x',y+y') - \overline{I(x,y)})$$

ここで、$\bar{T}$はテンプレート・ラスタ内のピクセル値の平均であり、$\overline{(x,y)}$ は画像の現在の「枠」内のピクセル値の平均である。

正規化相関係数（`method = CoeffNormed`）：

$$\tilde{R}(x,y) = \frac{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}(T(x',y') - \bar{T})\cdot(I(x+x',y+y') - \overline{I(x,y)})}{\sqrt{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}(T(x',y') - \bar{T})^2 \sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}(I(x+x',y+y') - \overline{I(x,y)})^2}}$$

生成された画像が返った後、その画像内のどの位置にテンプレートがあるかは、生成された画像の輝度の局所極大か大局極大のいずれかとして探せる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*templSize* フィールドの値が 0 または負の場合、あるいは ROI のサイズ（$W \times H$）がテンプレートのサイズ（$W \times H$）より小さい場合のエラー状態を示す。
ippStsStepErr	*templStep* または *imageStep* が *roiSize.width* <pixelSize>* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。

距離変換関数

本節では、距離変換関数について説明する。

距離変換関数はオブジェクトまでの距離を計算するのに使用する。入力に使用するのは、特徴ピクセルと非特徴ピクセルを含む 1 枚の画像である。この関数は、出力画像に含まれている非特徴ピクセルすべてに対して、最も近い特徴ピクセルまでの距離を示したラベルを付けていく働きをする。特徴ピクセルには 0 の印が付く。
距離変換は、骨格検出や形状解析など幅広い分野に使用されている。

DistanceTransform

ソース・イメージの非ゼロ・ピクセルすべてについて、最も近いゼロ・ピクセルまでの距離を計算する。

```
IppStatus ippiDistanceTransform_3x3_8u32f_C1R(const Ipp8u* pSrc,
    int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize,
     Ipp32f* pMetrics);
IppStatus ippiDistanceTransform_5x5_8u32f_C1R(const Ipp8u* pSrc,
    int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize,
     Ipp32f* pMetrics);
```

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pDst	出力距離画像へのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）
pMetrics	使用するメトリックを指定する配列へのポインタ

説明

距離変換関数 ippiDistanceTransform は、ippcv.h ファイルの中で宣言される。この関数は、固定セットからの原子間距離の総和を使用して、最も近いゼロ・ピクセルから特定のピクセルまでの実際の距離の近似値を求める。このセットは、3 × 3 マスクの 2 つの値と、5 × 5 マスクの 3 つの値で構成される。

図 14-5 は、ゼロ点をその中心に持つ 7 × 7 画像の距離変換を実行した結果を示している。この例は、3 × 3 マスクに対応している。3 × 3 マスクの場合は、次の 2 つの数値でメトリックを指定する。

- 共通のエッジを持つピクセル間の距離
- 共通のコーナーを持つピクセル間の距離

この場合は、1 と 1.5 の値がこれに相当する。

図 14-5　3 × 3 マスク

4.5	4	3.5	3	3.5	4	4.5
4	3	2.5	2	2.5	3	4
3.5	2.5	1.5	1	1.5	2.5	3.5
3	2	1	0	1	2	3
3.5	2.5	1.5	1	1.5	2.5	3.5
4	3	2.5	2	2.5	3	4
4.5	4	3.5	3	3.5	4	4.5

図 14-6 は、同じ画像に対する 5 × 5 マスクの場合の距離変換の結果を示している。

このマスクの場合は、メトリックを指定するために、さらにもう 1 つ数字（追加距離）が追加される。これは、チェスのナイトの動きに相当するピクセル間の距離である。

この例での追加距離は 2 である。

図 14-6　5 × 5 マスク

4.5	3.5	3	3	3	3.5	4
3.5	3	2	2	2	3	3.5
3	2	1.5	1	1.5	2	3
3	2	1	0	1	2	3
3	2	1.5	1	1.5	2	3
3.5	3	2	2	2	3	3.5
4	3.5	3	3	3	3.5	4

戻り値

`ippStsNoErr`	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
`ippStsNullPtrErr`	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
`ippStsSizeErr`	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
`ippStsStepErr`	*srcStep* または *dstStep* が *roiSize.width*<*pixelSize*> より小さい場合のエラー状態を示す。
`ippStsNotEvenStepErr`	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。

GetDistanceTransformMask

メトリックのタイプとマスクのサイズに対して最適なマスクを返す。

```
IppStatus ippiGetDistanceTransformMask(int maskType,
    Ipp32f* pMetrics);
```

引数

maskType　メトリックのタイプとマスクのサイズの両方を指定する。

次に示すように、パラメータ値の末尾の有効数字 （10 進表記法で表現）でメトリックのタイプを指定する。

0　$L_\infty, \Delta = max(|x_1 - x_2|, |y_1 - y_2|)$,

1　$L_1, \Delta = |x_1 - x_2| + |y_1 - y_2|$,

2　$L_2, \Delta = sqrt((x_1 - x_2)^2 + (y_1 - y_2)^2)$
（ユークリッド距離をメトリックとする）

次のように、先頭の 10 進数字でマスクのサイズを指定する。

3　3 × 3 マスク

5　5 × 5 マスク。ユークリッド・メトリックの場合のみ意味を持つ。
その理由は、残り 2 つのメトリックの場合、このマスクでは高い精度が実現できないからではなく、処理が遅いからである。

したがって、*maskType* パラメータの値は 30、31、32、52 を許容できる。

pMetrics　メトリック・パラメータの保存先である出力配列へのポインタ。この配列に含まれる要素数は、次のとおりである。

2　3 × 3 マスク

3　5 × 5 マスク

説明

関数 ippiGetDistanceTransformMask は、ippcv.h ファイルの中で宣言される。この関数は、メトリックのタイプとマスクのサイズに関する各種のメトリック・パラメータで出力配列を埋める。この関数からは、次の結果が返ってくる。

(1, 1)　L_∞、3 × 3 マスク、

(1, 2)	L_1、3 × 3 マスク、
(0.955, 1.3693)	L_2、3 × 3 マスク、
(1.0, 1.4, 2.1969)	L_2、5 × 5 マスク

詳細は、[Borge86] を参照のこと。

戻り値

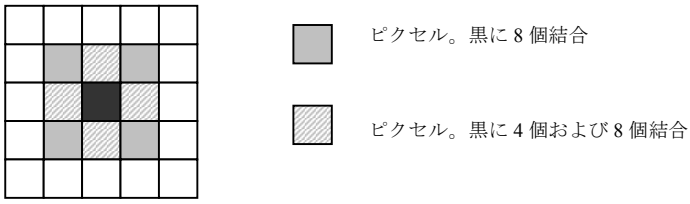
`ippStsNoErr`	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
`ippStsNullPtrErr`	*pMetrics* ポインタが NULL の場合のエラー状態を示す。
`ippStsOutOfRangeErr`	マスクが範囲外の場合のエラー状態を示す。

フラッド・フィル関数

本節では、複数の連結領域に対してフラッド・フィルを実行する関数について説明する。「*フラッド・フィル*」とは、近い値を持つ複数の連結ピクセルで構成する 1 つのグループを、全部同じ値で埋める（すなわち、全部同じ値に設定する）意味である。フラッド・フィルの処理は、指定した一点（これを「シード点」と言う）から始まる。画像の ROI の境界まで達するか、あるいはピクセル値同士の違いが大きくなりすぎたせいで、埋めるべき新しいピクセルが見つからなくなるまで続く。埋まったすべてのピクセルは、次のように近傍ピクセルの解析が行われる。

- 4- 近傍（対角にあるピクセルは除く）。この種の連結性を「4- 連結」と呼ぶ。対応する関数名には 4Con の文字を含む。
- 8- 近傍（対角にあるピクセルも含む）。この種の連結性を「8- 連結」と呼ぶ。対応する関数名には 8Con の文字を含む。

図 14-7　ピクセル連結性のパターン



これらの関数は、次の用途に使用できる。

- 1 枚のグレー・スケール画像の領域分割を行い、一連の単色領域に変える。

- 2 レベル画像を対象に、連結成分それぞれに別々の色で印を付ける。

FloodFillGetSize

関数 FloodFill 用のテンポラリ・バッファのサイズを計算する。

```
IppStatus ippiFloodFillGetSize(IppiSize roiSize, int* pBufSize);
```

引数

roiSize	ソース・イメージの ROI のサイズ（ピクセル単位）
pBufSize	テンポラリ・バッファのサイズを返す変数へのポインタ

説明

関数 ippiFloodFillGetSize は、ippcv.h ファイルの中で宣言される。この関数は、関数 FloodFill が使用するテンポラリ・バッファのサイズを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pBufSize* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。

FloodFillGetSize_Grad

関数 FloodFill_Grad 用のテンポラリ・バッファのサイズを計算する。

```
IppStatus ippiFloodFillGetSize_Grad(IppiSize roiSize, int* pBufSize);
```

引数

roiSize	ソース・イメージの ROI のサイズ（ピクセル単位）
pBufSize	テンポラリ・バッファのサイズを返す変数へのポインタ

説明

関数 ippiFloodFillGetSize_Grad は、ippcv.h ファイルの中で宣言される。この関数は、関数 FloodFill_Grad が使用するテンポラリ・バッファのサイズを計算する。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	*pBufSize* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。

FloodFill

画像上の連結領域に対してフラッド・フィルを実行する。

```
IppStatus ippiFloodFill_4Con_<mod>(Ipp<DataType>* pImage, int
      imageStep, IppiSize roiSize, IppiPoint seed, int newVal,
      IppiConnectedComp* pRegion, void* pBuffer);
IppStatus ippiFloodFill_8Con_<mod>(Ipp<DataType>* pImage, int
      imageStep, IppiSize roiSize, IppiPoint seed, int newVal,
      IppiConnectedComp* pRegion, void* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IR      32f_C1IR
```

引数

pImage	（インプレース操作の場合）ソースおよびデスティネーション・イメージの ROI へのポインタ

imageStep	画像の ROI 内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）
seed	始点
newVal	埋める値
pRegion	再度埋められた領域に関する情報を保存する連結成分構造体へのポインタ
pBuffer	テンポラリ・バッファへのポインタ

説明

関数 ippiFloodFill は、ippcv.h ファイルの中で宣言される。この関数は、*seed* ピクセルの近傍のうち、すべてのピクセル値が *seed* 値の点に等しい条件を満たしているピクセルを埋める。

_4con のサフィックスが付いている関数は、各ピクセルの 4- 連結近傍（上下左右）を調べる働きをする。_8con のサフィックスが付いている関数は、各ピクセルの 8- 連結近傍 （上下左右と対角）を調べる働きをする。図 14-7 を参照のこと。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*imageStep* が *pRoiSize.width* <pixelSize>* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsOutofRangeErr	*seed* 点が ROI の外側にある場合のエラー状態を示す。

FloodFill_Grad

画像上の連結領域に対してフラッド・フィルを実行する。

```
IppStatus ippiFloodFill_Grad4Con_<mod>(Ipp<DataType>* pImage, int
     imageStep, IppiSize roiSize, IppiPoint seed, int newVal, int
     d_lw, int d_up, IppiConnectedComp* pRegion, void* pBuffer);
IppStatus ippiFloodFill_Grad8Con_<mod>(Ipp<DataType>* pImage, int
     imageStep, IppiSize roiSize, IppiPoint seed, int newVal, int
     d_lw, int d_up, IppiConnectedComp* pRegion, void* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

```
8u_C1IR     32f_C1IR     32f_C1R
```

引数

pImage	（インプレース操作の場合）ソースおよびデスティネーション・イメージの ROI へのポインタ
imageStep	画像の ROI 内のステップ（バイト単位）
roiSize	画像の ROI のサイズ（ピクセル単位）
seed	始点
d_lw	近傍ピクセル同士の差を求めるときの最小値
d_up	近傍ピクセル同士の差を求めるときの最大値
newVal	埋める値
pRegion	再度埋められた領域に関する情報を保存する連結成分構造体へのポインタ
pBuffer	テンポラリ・バッファへのポインタ

説明

関数 ippiFloodFill_Grad は、ippcv.h ファイルの中で宣言される。この関数は、*seed* ピクセルの近傍のピクセルのうち、すべてのピクセル値が互いに近い値である条件を満たしているピクセルを埋める。値 v が以下の諸条件を満たしている場合、そのピクセルは再度埋められた領域に属していると見なせる。

$$v_0 - d_{lw} \le v \le v_0 + d_{up} ,$$

v_0 は、現在の近傍のピクセルのうちの少なくとも 1 個の値である。すでにこれは、再度埋められた領域に属している。d_{lw} および d_{up} は、それぞれ *minDelta* と *maxDelta* である。

_4con のサフィックスが付いている関数は、各ピクセルの 4- 連結近傍（上下左右）を調べる働きをする。_8con のサフィックスが付いている関数は、各ピクセルの 8- 連結近傍（上下左右と対角）を調べる働きをする。図 14-7 を参照のこと。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*imageStep* が *pRoiSize.width* <pixelSize>* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsOutofRangeErr	*seed* 点が ROI の外側にあるか、*minDelta* または *maxDelta* が負か、あるいは *minDelta + maxDelta* が Ipp8u data type に関して 255 より大きい場合のエラー状態を示す。

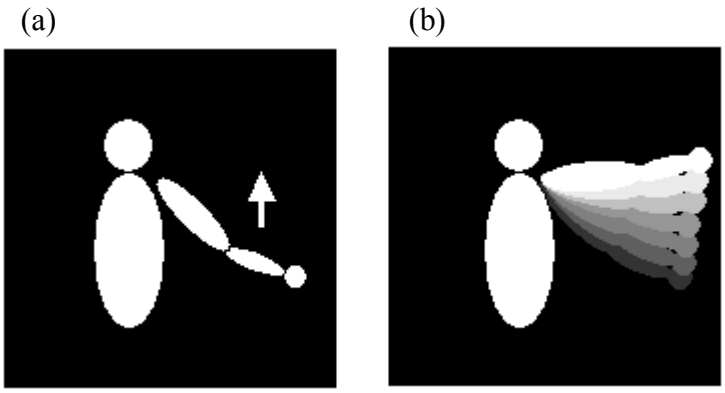
モーション・テンプレート関数

本節では、モーション・テンプレート関数について説明する。
この関数は、問題のモーションが「どこで」「どのようにして」「どの方向に」発生したかを短時間で特定するためのモーション・テンプレート画像を生成する。このアルゴリズムは、Davis、Bobick による [Davis97] および Davis、Bradski による [Davis99] の論文に基づく。この関数は、フレームや背景の差分抽出操作で生成された画像か、ほかの領域分割操作で生成された画像を操作する。したがって、入力画像も出力画像も、そのタイプはすべてグレー・スケールで、カラー・チャネルは 1 本である。ピクセルのタイプについては、`8u`、`8s`、`32f` のいずれも可能である。

モーション描写

図 14-8 (a) は、移動しているオブジェクトまたは人間の最前面のシルエットを獲得しているところである。この人間またはオブジェクトが動くとき、時間的に一番新しい最前面のシルエットを、そのモーション履歴画像内の最高値としてコピーすると、生成されたモーションの「階層化履歴」を作成する。通常この最高値は、当該コードが作動してからの時間を示す浮動小数点タイムスタンプ（ミリ秒単位）でしかない。図 14-8 (b) は、「*モーション履歴画像*」(MHI) とも呼ばれる処理結果を示している。図 14-8 の MHI は、動きがあったことを示している。ピクセル・レベルまたはタイム・デルタしきい値は、MHI に含まれるピクセル値のうち、しきい値を下回るピクセル値が 0 になるように適宜設定する。

図 14-8　モーション画像履歴



下回ったときに値が 0 に設定されるしきい値があるため、時間的に一番新しいモーションが最高値であり、それより古くなるほど値は下がっていく。

MHI 画像の更新

一般的に、浮動小数点画像を使用する理由は、システム時間が異なるからである。つまり、当該アプリケーションを起動してからの経過時間は数ミリ秒単位で読み取られ、その値はさらに、時間的に一番新しいシルエットの値である浮動小数点数値に変換する。次に、過去のシルエットの上に、この最新のシルエットが上書きされる。MHI を作成するには古すぎるピクセルは、後続のしきい値操作によって消える。

UpdateMotionHistory

所定のタイムスタンプにあるモーション・シルエットを使用して、モーション履歴画像を更新する。

```
IppStatus ippiUpdateMotionHistory_8u32f_C1IR(const Ipp8u* pSilhouette,
    int silhStep, Ipp32f* pMhi, int mhiStep, IppiSize roiSize,
     Ipp32f timeStamp, Ipp32f mhiDuration);
```

引数

pSilhouette	モーションが起きたピクセルの値が 0 以外のシルエット画像へのポインタ
roiSize	画像の ROI のサイズ（ピクセル単位）
silhStep	シルエット画像内のステップ（バイト単位）
pMhi	入力兼出力パラメータであるモーション履歴画像へのポインタ
mhiStep	モーション履歴画像内のステップ（バイト単位）
timeStamp	タイムスタンプ （ミリ秒単位）
mhiDuration	MHI ピクセルに適用されるしきい値。このしきい値より古い MHI モーションは削除される。

説明

関数 ippiUpdateMotionHistory は、ippcv.h ファイルの中で宣言される。この関数は、モーション履歴画像を更新する。

MHI ピクセルの値が 0 以外の場合は、そのピクセルは最新の *timestamp* 値に設定される。

MHI ピクセルの値が *mhiDuration* タイムスタンプより小さい場合、すなわちピクセルが古い場合は、その MHI ピクセルは削除される。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtrErr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsNotEvenStepErr	*mhiStep* または *silhStep* が *roiSize.width* <pixelSize>* より小さいか、あるいは浮動小数点画像のステップ数が 4 で割り切れない場合のエラー状態を示す。
ippStsOutOfRangeErr	*mhiDuration* が負の場合のエラーを示す。

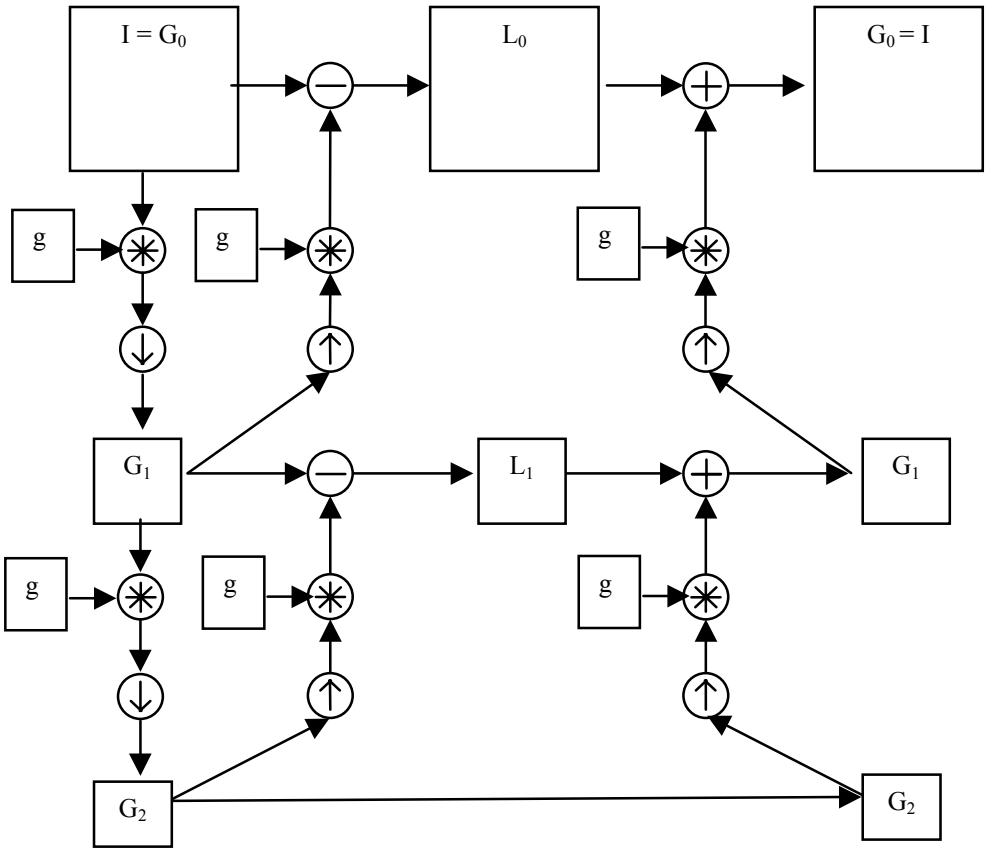
角錐関数

本節で説明する関数は、ガウシアン画像の角錐とラプラシアン画像の角錐について、生成と再構成を行う。

図 14-9 は、ガウシアンまたはラプラシアンの角錐を作成するときの概念を示している。元の画像 G_0 は、ガウシアンでたたみ込まれ、次にダウンサンプリングされ、変換画像 G_1 になる。この操作は、画像のサイズが 1 ピクセルになるまで必要なだけ続けられる。

ラプラシアン角錐は、次のようにするとガウシアン角錐から作成できる。ラプラシアン・レベル k (L_k) は、下位レベルの画像 G_{k+1} をアップサンプリングすると作成できる。ガウシアン・カーネル g で画像をたたみ込むと、アップサンプリング後に消えたピクセルの補間が行われる。生成された画像は、ガウシアンで平滑化した画像 G_k から差し引かれる。元の画像を構築し直すときは、図 14-9 に示したように、この手順を逆にたどる。

図 14-9　3 レベルのガウシアン角錐とラプラシアン角錐



左側のガウシアン画像角錐を使用すると、中央のラプラシアン角錐が作成され、それを使用すると、右側のガウシアン角錐と元の画像が再構成される。図 14-9 の中で、*I* は元の画像、G_k はガウシアン画像、L_k はラプラシアン画像、添字 *k* は角錐のレベル、添字 *g* は、ダウンサンプリングの*前*かアップサンプリングの*後*に画像をたたみ込むときに使用するガウシアン・カーネルである。

インテル IPP ではガウシアン角錐しか実装していない。ラプラシアン角錐を構築するには、インテル IPP 角錐に関する次の一般的な規則に従わなければならない。

$$L_i = G_i - \mathrm{PyrUp}(\mathrm{PyrDown}(G_i)),$$

L_i は、ラプラシアン角錐の i 番目のレベルにある画像である。G_i は、ガウシアン角錐の i 番目のレベルにある画像である。

PyrDownGetBufSize

関数 PyrDown 用のテンポラリ・バッファの
サイズを計算する。

```
IppStatus ippiPyrDownGetBufSize_Gauss5x5(int roiWidth,
    IppDataType dataType, int channels, int* pBufSize);
```

引数

roiWidth	ソース・イメージの ROI の幅（ピクセル単位）
dataType	ソース・イメージのデータ・タイプ
channels	チャネル数
pBufSize	テンポラリ・バッファのサイズを返す変数へのポインタ

説明

関数 ippiPyrDownGetBufSize は、ippcv.h ファイルの中で宣言される。この関数は、関数 PyrDown が使用するテンポラリ・バッファのサイズを計算する。算出されたサイズのバッファを使用して、比較的小さな画像の処理ができる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtr	*pBufSize* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiWidth* が負の場合、または 2 で割り切れない場合のエラー状態を示す。
ippStsNumChannelsErr	*channels* が1でも3でもない場合のエラー状態を示す。
ippStsDataTypeErr	*dataType* が Ipp8u、Ipp8s、Ipp32f のいずれでもない場合のエラー状態を示す。

PyrUpGetBufSize

関数 PyrUp 用のテンポラリ・バッファのサイズを計算する。

```
IppStatus ippiPyrUpGetBufSize_Gauss5x5(int roiWidth,
    IppDataType dataType, int channels, int* pBufSize);
```

引数

roiWidth	ソース・イメージの ROI の幅（ピクセル単位）
dataType	ソース・イメージのデータ・タイプ
channels	チャネル数
pBufSize	テンポラリ・バッファのサイズ

説明

関数 ippiPyrUpGetBufSize は、ippcv.h ファイルの中で宣言される。この関数は、関数 PyrUp が使用するテンポラリ・バッファのサイズを計算する。算出されたサイズのバッファを使用して、比較的小さな画像の処理ができる。

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtr	*pBufSize* ポインタが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiWidth* が 0 または負の場合のエラー状態を示す。
ippStsNumChannelsErr	*channels* が 1 でも 3 でもない場合のエラー状態を示す。
ippStsDataTypeErr	*dataType* が Ipp8u、Ipp8s、Ipp32f のいずれでもない場合のエラー状態を示す。

PyrDown

画像にガウシアン・フィルタを適用し、次にダウンサンプリングを実行する。

```
IppStatus ippiPyrDown_Gauss5x5_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    void* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	8s_C1R	32f_C1R
8u_C3R	8s_C3R	32f_C3R

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
roiSize	ソース・イメージの ROI のサイズ（ピクセル単位）
pBuffer	PyrDownGetBufSize で計算されたサイズのテンポラリ・バッファへのポインタ

説明

関数 ippiPyrDown は、ippcv.h ファイルの中で宣言される。この関数は、5 × 5 のガウシアン・フィルタをソース・イメージ *pSrc* に適用し、次にそれをダウンサンプリングする。つまり、奇数行と奇数列が画像から削除される。次のマスクを使用する。

$$\frac{1}{16}\begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix} \times \frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix} = \frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\4 & 16 & 24 & 16 & 4\\6 & 24 & 36 & 24 & 6\\4 & 16 & 24 & 16 & 4\\1 & 4 & 6 & 4 & 1\end{bmatrix}$$

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合、あるいは 2 で割り切れない場合のエラーを示す。
ippStsStepErr	*srcStep* が *roiSize.width* <pixelSize>* より小さいか、*dstStep* が (*roiSize.width*<pixelSize>*) /*2* より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が4で割り切れない場合のエラー状態を示す。

PyrUp

画像をアップサンプリングし、次にガウシアン・フィルタを適用する。

```
IppStatus ippiPyrUp_Gauss5x5_<mod>(const Ipp<datatype>* pSrc, int
     srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
     void* pBuffer);
```

サポートされる *mod* の値は、次のとおりである。

8u_C1R	8s_C1R	32f_C1R
8u_C3R	8s_C3R	32f_C3R

引数

pSrc	ソース・イメージへのポインタ
srcStep	ソース・イメージ内のステップ（バイト単位）
pDst	デスティネーション・イメージへのポインタ
dstStep	デスティネーション・イメージ内のステップ（バイト単位）
roiSize	ソース・イメージの ROI のサイズ（ピクセル単位）

pBuffer	PyrUpGetBufSize で計算されたサイズのテンポラリ・バッファへのポインタ

説明

関数 ippiPyrUp は、ippcv.h ファイルの中で宣言される。この関数は、ソース・イメージ *pSrc* をアップサンプリングする。正確に言えば、奇数行、ゼロ行、奇数列、ゼロ列を挿入し、次に 4 倍した 5 × 5 ガウシアン・フィルタを適用する。次のマスクを使用する。

$$4 \times \frac{1}{16}\begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix} \times \frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix} = \frac{1}{64}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\4 & 16 & 24 & 16 & 4\\6 & 24 & 36 & 24 & 6\\4 & 16 & 24 & 16 & 4\\1 & 4 & 6 & 4 & 1\end{bmatrix}$$

戻り値

ippStsNoErr	エラーがないことを示す。これ以外の値は、すべてエラーまたは警告を示す。
ippStsNullPtr	指定のポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* が *roiSize.width*<*pixelSize*> より小さいか、*dstStep* が *2*roiSize.width*<*pixelSize*> より小さい場合のエラー状態を示す。
ippStsNotEvenStepErr	浮動小数点画像のステップ数が4で割り切れない場合のエラー状態を示す。

15 画像圧縮関数

本章では、JPEG と JPEG2000 標準（[ISO10918] と [ISO15444] を参照）に従ってデータを準備し、静止画の圧縮とコーティングを実行するインテル® IPP の画像処理関数について説明する。
これらの関数は、以降の節で次のようにグループ分けしている。

サポート関数

JPEG コーティング用関数

カラー変換関数

複合カラー変換関数

量子化関数

量子化、DCT、レベル・シフトを組み合わせた関数

レベル・シフト関数

サンプリング関数

プレーンからピクセルおよびピクセルからプレーンへの変換関数

ハフマン・コーデック関数

JPEG2000 コーティング用関数

ウェーブレット変換関数

JPEG2000 エントロピー・コード化およびデコード関数

コンポーネント変換関数

それぞれの節では、はじめに、その関数が操作する内容を簡単に述べ、関数一覧を示した後、詳しく説明する。

注： 簡単にするため、一覧表に示す関数グループ名の ippi プリフィックスは省略している。関数プロトタイプと本文中に示す場合は、省略せずに完全な名前を使用している

サポート関数

本節では、使用したインテル IPP JPEG コーデック・ソフトウェアの現在のバージョンに関する情報を返すサポート関数について説明する。

ippjGetLibVersion

ライブラリ・バージョンの情報を返す。

```
const IppLibraryVersion*  ippjGetLibVersion(void);
```

説明

関数 ippjGetLibVersion は、ippj.h ファイルの中で宣言される。この関数は、インテル IPP JPEG コーデック・ライブラリの現在のバージョンに関する情報を含む静的データ構造体 IppLibraryVersion へのポインタを返す。このポインタは静的変数を指しているので、このポインタから参照されるメモリを解放する必要はない。IppLibraryVersion 構造体には、以下のフィールドが含まれている。

major	現在のライブラリ・バージョンのメジャー番号
minor	現在のライブラリ・バージョンのマイナー番号
Name	ライブラリの名前
Version	ライブラリ・バージョンの ASCII 文字列
BuildDate	ライブラリ・バージョンが作成された日付

戻り値

この関数の戻り値は、構造体 IppLibraryVersion へのポインタである。

カラー変換関数

本節では、JPEG コーデック専用のインテル IPP カラー変換関数について説明する。

表 15-1 に、これらの関数の一覧を示す。

表 15-1　カラー変換関数

関数の基本名	説明
RGBToY_JPEG	RGB 画像をグレー・スケールに変換する。
BGRToY_JPEG	BGR 画像をグレー・スケールに変換する。
RGBToYCbCr_JPEG	RGB 画像を YCbCr カラー・モデルに変換する。
BGRToYCbCr_JPEG	BGR 画像を YCbCr カラー・モデルに変換する。
CMYKToYCCK_JPEG	CMYK 画像を YCCK カラー・モデルに変換する。
YCbCrToRGB_JPEG	YCbCr 画像を RGB カラー・モデルに変換する。
YCbCrToBGR_JPEG	YCbCr 画像を BGR カラー・モデルに変換する。
YCCKToCMYK_JPEG	YCCK 画像を CMYK カラー・モデルに変換する。

RGBToY_JPEG

RGB 画像をグレー・スケールに変換する。

事例 1：プレーン・データの操作

```
IppStatus ippiRGBToY_JPEG_8u_P3C1R(const Ipp8u* pSrcRGB[3],
    int srcStep, Ipp8u*  pDstY, int dstStep, IppiSize roiSize);
```

事例 2：ピクセル順序データの操作

```
IppStatus ippiRGBToY_JPEG_8u_C3C1R(const Ipp8u* pSrcRGB, int srcStep,
    Ipp8u* pDstY, int dstStep, IppiSize roiSize);
```

引数

pSrcRGB	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstY	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRGBToY_JPEG は、ippj.h ファイルの中で宣言される。この関数は、次の公式を使用して RGB 画像をグレー・スケールに変換する。

```
Y  =  0.299*R + 0.587*G + 0.114*B
```

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

BGRToY_JPEG

BGR 画像をグレー・スケールに変換する。

```
IppStatus ippiBGRToY_JPEG_8u_C3C1R(const Ipp8u* pSrcBGR, int srcStep,
    Ipp8u* pDstY, int dstStep, IppiSize roiSize);
```

引数

pSrcBGR	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstY	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiBGRToY_JPEG は、ippj.h ファイルの中で宣言される。この関数は、次の公式を使用して BGR 画像をグレー・スケールに変換する。

```
Y  =  0.299*R + 0.587*G + 0.114*B
```

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

RGBToYCbCr_JPEG

RGB 画像を YCbCr カラー・モデルに変換する。

事例 1：プレーン・データの操作

```
IppStatus ippiRGBToYCbCr_JPEG_8u_P3R(const Ipp8u* pSrcRGB[3], int
    srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

事例 2：ピクセル順序データの操作

```
IppStatus ippiRGBToYCbCr_JPEG_8u_C3P3R(const Ipp8u* pSrcRGB, int
    srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

引数

pSrcRGB	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstYCbCr	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRGBToYCbCr_JPEG は、ippj.h ファイルの中で宣言される。この関数は、次の公式を使用して RGB 画像を YCbCr カラー・モデルに変換する。

```
Y =   0.299*R    + 0.587*G    + 0.114*B
Cb = -0.16874*R - 0.33126*G + 0.5*B      + 128
Cr =  0.5*R      - 0.41869*G - 0.08131*B + 128
```

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

BGRToYCbCr_JPEG

BGR 画像を YCbCr カラー・モデルに変換する。

```
IppStatus ippiBGRToYCbCr_JPEG_8u_C3P3R(const Ipp8u* pSrcBGR,
    int srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

引数

pSrcBGR	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstYCbCr	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiBGRToYCbCr_JPEG は、ippj.h ファイルの中で宣言される。この関数は、ippiRGBToYCbCr_JPEG 関数の場合と同じ公式を使用して Y、Cb、Cr の各成分値を計算し、BGR 画像を YCbCr カラー・モデルに変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

CMYKToYCCK_JPEG

CMYK 画像を YCCK カラー・モデルに変換する。

事例 1：プレーン・データの操作

```
IppStatus ippiCMYKToYCCK_JPEG_8u_P4R(const Ipp8u* pSrcCMYK[4], int
    srcStep, Ipp8u* pDstYCCK[4], int dstStep, IppiSize roiSize);
```

事例 2：ピクセル順序データの操作

```
IppStatus ippiCMYKToYCCK_JPEG_8u_C4P4R(const Ipp8u* pSrcCMYK, int
    srcStep, Ipp8u* pDstYCCK[4], int dstStep, IppiSize roiSize);
```

引数

pSrcCMYK	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstYCCK	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiCMYKToYCCK_JPEG は、ippj.h ファイルの中で宣言される。この関数は、2 つのステップで CMYK 画像を YCCK 画像に変換する。最初は、RGB 形式に変換する。

```
R = 255 – C
G = 255 – M
B = 255 – Y
```

その後、YCCK 画像に変換する。

```
Y =   0.299*R   + 0.587*G   + 0.114*B
Cb = -0.16874*R - 0.33126*G + 0.5*B     + 128
Cr =  0.5*R     - 0.41869*G - 0.08131*B + 128
```

K チャネルの各値は、変更されずに書き込まれる。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCrToRGB_JPEG

YCbCr 画像を RGB カラー・モデルに変換する。

事例 1：プレーン・データの操作

```
IppStatus ippiYCbCrToRGB_JPEG_8u_P3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp8u* pDstRGB[3], int dstStep, IppiSize roiSize);
```

事例 2：ピクセル順序データの操作

```
IppStatus ippiRGBToYCbCr_JPEG_8u_P3C3R(const Ipp8u* pSrcYCbCr[3], int
    srcStep, Ipp8u* pDstRGB, int dstStep, IppiSize roiSize);
```

引数

pSrcYCbCr	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstRGB	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYCbCrToRGB_JPEG は、ippj.h ファイルの中で宣言される。この関数は、次の公式に従って YCbCr 画像を RGB 画像に変換する。

```
R = Y  + 1.402*Cr - 179.456
G = Y  - 0.34414*Cb - 0.71414*Cr + 135.45984
B = Y  + 1.772*Cb - 226.816
```

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCrToBGR_JPEG

YCbCr 画像を BGR カラー・モデルに変換する。

```
IppStatus ippiYCbCrToBGR_JPEG_8u_P3C3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp8u* pDstBGR, int dstStep, IppiSize roiSize);
```

引数

pSrcYCbCr	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstBGR	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYCbCrToBGR_JPEG は、ippj.h ファイルの中で宣言される。この関数は、ippiYCbCrToRGB_JPEG 関数の場合と同じ公式を使用して R、G、B の各成分値を計算し、BGR 画像を YCbCr 画像に変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

YCCKToCMYK_JPEG

YCCK 画像を CMYK カラー・モデルに変換する。

事例 1：プレーン・データの操作

```
IppStatus ippiYCCKToCMYK_JPEG_8u_P4R(const Ipp8u* pSrcYCCK[4], int
    srcStep, Ipp8u* pDstCMYK[4], int dstStep, IppiSize roiSize);
```

事例 2：ピクセル順序データの操作

```
IppStatus ippiYCCKToCMYK_JPEG_8u_P4C4R(const Ipp8u* pSrcYCCK[4], int
    srcStep, Ipp8u* pDstCMYK, int dstStep, IppiSize roiSize);
```

引数

pSrcYCCK	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstCMYK	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiYCCKToCMYK_JPEG は、ippj.h ファイルの中で宣言される。この関数は、2 つのステップで YCCK 画像を CMYK 画像に変換する。最初は、RGB 形式に変換する。

```
R = Y  + 1.402*Cr - 179.456
G = Y  - 0.34414*Cb - 0.71414*Cr + 135.45984
B = Y  + 1.772*Cb - 226.816
```

その後、CMYK 画像に変換する。

```
C = 255 – R
M = 255 – G
Y = 255 – B
```

K チャネルの各値は、変更されずに書き込まれる。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

複合カラー変換関数

本節では、サンプリング、カラー変換、レベル・シフト（符号なしデータ範囲と符号付きデータ範囲間の変換）を実行するインテル IPP 関数の中でも特に JPEG コーデック専用の関数について説明する。表 15-2 に、これらの関数の一覧を示す。

表 15-2　複合カラー変換関数

関数の基本名	説明
RGBToYCbCr444LS_MCU	RGB 画像を YCbCr カラー・モデルに変換し、444 MCU を作成する。
RGBToYCbCr422LS_MCU	RGB 画像を YCbCr カラー・モデルに変換し、422 MCU を作成する。
RGBToYCbCr411LS_MCU	RGB 画像を YCbCr カラー・モデルに変換し、411 MCU を作成する。
BGRToYCbCr444LS_MCU	BGR 画像を YCbCr カラー・モデルに変換し、444 MCU を作成する。

表 15-2　複合カラー変換関数

関数の基本名	説明
BGRToYCbCr422LS_MCU	BGR 画像を YCbCr カラー・モデルに変換し、422 MCU を作成する。
BGRToYCbCr411LS_MCU	BGR 画像を YCbCr カラー・モデルに変換し、411 MCU を作成する。
CMYKToYCCK444LS_MCU	CMYK 画像を YCCK カラー・モデルに変換し、4444 MCU を作成する。
CMYKToYCCK422LS_MCU	CMYK 画像を YCCK カラー・モデルに変換し、4224 MCU を作成する。
CMYKToYCCK411LS_MCU	CMYK 画像を YCCK カラー・モデルに変換し、4114 MCU を作成する。
YCbCr444ToRGBLS_MCU	444 MCU から YCbCr 画像を作成し、それを RGB カラー・モデルに変換する。
YCbCr422ToRGBLS_MCU	422 MCU から YCbCr 画像を作成し、それを RGB カラー・モデルに変換する。
YCbCr411ToRGBLS_MCU	411 MCU から YCbCr 画像を作成し、それを RGB カラー・モデルに変換する。
YCbCr444ToBGRLS_MCU	444 MCU から YCbCr 画像を作成し、それを BGR カラー・モデルに変換する。
YCbCr422ToBGRLS_MCU	422 MCU から YCbCr 画像を作成し、それを BGR カラー・モデルに変換する。
YCbCr411ToBGRLS_MCU	411 MCU から YCbCr 画像を作成し、それを BGR カラー・モデルに変換する。
YCCK444ToCMYKLS_MCU	4444 MCU から YCCK 画像を作成し、それを CMYK カラー・モデルに変換する。
YCCKToCMYK422LS_MCU	4224 MCU から YCCK 画像を作成し、それを CMYK カラー・モデルに変換する。
YCCKToCMYK411LS_MCU	4114 MCU から YCCK 画像を作成し、それを CMYK カラー・モデルに変換する。

ここで言う MCU は、JPEG Minimum Coded Units（最小符号化単位）の略である。サンプリング係数によって決まるサイズのブロックをいくつかインターリーブ方式でまとめたものか、あるいはインターリーブ以外の方式で走査したものの中に含まれている 1 個のブロックのいずれかを指している。

すべての関数は、[ISO10918], *Annex A.3.1, Level Shift* による定義に従ってレベル・シフト操作を実現している。

RGBToYCbCr444LS_MCU

RGB 画像を YCbCr カラー・モデルに変換し、444 MCU を作成する。

```
IppStatus ippiRGBToYCbCr444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB,
    int srcStep, Ipp16s* pDstMCU[3]);
```

引数

pSrcRGB	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstMCU	デスティネーション・データへのポインタ

説明

関数 ippiRGBToYCbCr444LS_MCU は、ippj.h ファイルの中で宣言される。この関数は、ippiRGBToYCbCr_JPEG 関数の場合と同じ公式を使用して Y、Cb、Cr の各成分値を計算し、RGB 画像を YCbCr 画像に変換する。

また、この関数は、128 を引いてレベル・シフト操作を行って、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へデータを変換し、444 MCU を作成する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

RGBToYCbCr422LS_MCU

RGB 画像を YCbCr カラー・モデルに変換し、422 MCU を作成する。

```
IppStatus ippiRGBToYCbCr422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB,
    int srcStep, Ipp16s* pDstMCU[3]);
```

引数

pSrcRGB	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstMCU	デスティネーション・データへのポインタ

説明

関数 ippiRGBToYCbCr422LS_MCU は、ippj.h ファイルの中で宣言される。この関数は、ippiRGBToYCbCr_JPEG 関数の場合と同じ公式を使用して Y、Cb、Cr の各成分値を計算し、RGB 画像を YCbCr 画像へ変換する。

また、この関数は、128 を引いてレベル・シフト操作を行って、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へデータを変換し、422 MCU を作成する。

単純平均を求めて、サンプリングを実行する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

RGBToYCbCr411LS_MCU

RGB 画像を YCbCr カラー・モデルに変換し、411 MCU を作成する。

```
IppStatus ippiRGBToYCbCr411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB,
    int srcStep, Ipp16s* pDstMCU[3]);
```

引数

pSrcRGB	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstMCU	デスティネーション・データへのポインタ

説明

関数 ippiRGBToYCbCr411LS_MCU は、ippj.h ファイルの中で宣言される。この関数は、ippiRGBToYCbCr_JPEG 関数の場合と同じ公式を使用して Y、Cb、Cr の各成分値を計算し、RGB 画像を YCbCr 画像に変換する。

また、この関数は、128 を引いてレベル・シフト操作を行って、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へデータを変換し、411 MCU を作成する。

単純平均を求めて、サンプリングを実行する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

BGRToYCbCr444LS_MCU

BGR 画像を YCbCr カラー・モデルに変換し、444 MCU を作成する。

```
IppStatus ippiBGRToYCbCr444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

引数

pSrcBGR	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstMCU	デスティネーション・データへのポインタ

説明

関数 ippiBGRToYCbCr444LS_MCU は、ippj.h ファイルの中で宣言される。この関数は、ippiRGBToYCbCr_JPEG 関数の場合と同じ公式を使用して Y、Cb、Cr の各成分値を計算し、BGR 画像を YCbCr 画像に変換する。

また、この関数は、128 を引いてレベル・シフト操作を行って、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へデータを変換し、444 MCU を作成する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

BGRToYCbCr422LS_MCU

BGR 画像を YCbCr カラー・モデルに変換し、422 MCU を作成する。

```
IppStatus ippiBGRToYCbCr422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

引数

pSrcBGR	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstMCU	デスティネーション・データへのポインタ

説明

関数 ippiBGRToYCbCr422LS_MCU は、ippj.h ファイルの中で宣言される。この関数は、ippiRGBToYCbCr_JPEG 関数の場合と同じ公式を使用して Y、Cb、Cr の各成分値を計算し、BGR 画像を YCbCr 画像に変換する。

また、この関数は、128 を引いてレベル・シフト操作を行って、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へデータを変換し、422 MCU を作成する。

単純平均を求めて、サンプリングを実行する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。

ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

BGRToYCbCr411LS_MCU

BGR 画像を YCbCr カラー・モデルに変換し、411 MCU を作成する。

```
IppStatus ippiBGRToYCbCr411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

引数

pSrcBGR	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstMCU	デスティネーション・データへのポインタ

説明

関数 ippiBGRToYCbCr411LS_MCU は、ippj.h ファイルの中で宣言される。この関数は、ippiRGBToYCbCr_JPEG 関数の場合と同じ公式を使用して Y、Cb、Cr の各成分値を計算し、BGR 画像を YCbCr 画像に変換する。

また、この関数は、128 を引いてレベル・シフト操作を行って、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へデータを変換し、411 MCU を作成する。

単純平均を求めて、サンプリングを実行する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

CMYKToYCCK444LS_MCU

CMYK 画像を YCCK カラー・モデルに変換し、4444 MCU を作成する。

```
IppStatus ippiCMYKToYCCK444LS_MCU _8u16s_C4P4R(const Ipp8u* pSrcCMYK,
    int srcStep, Ipp16s* pDstMCU[4]);
```

引数

pSrcCMYK	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstMCU	デスティネーション・データへのポインタ

説明

関数 ippiCMYKToYCCK444LS_MCU は、ippj.h ファイルの中で宣言される。この関数は、関数 ippiCMYKToYCCK_JPEG と同じ方法で CMYK 画像を YCCK 画像に変換する。

また、この関数は、128 を引いてレベル・シフト操作を行って、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へデータを変換し、4444 MCU を作成する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

CMYKToYCCK422LS_MCU

CMYK 画像を YCCK カラー・モデルに変換し、422 MCU を作成する。

```
IppStatus ippiCMYKToYCCK422LS_MCU _8u16s_C4P4R(const Ipp8u* pSrcCMYK,
    int srcStep, Ipp16s* pDstMCU[4]);
```

引数

pSrcCMYK	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstMCU	デスティネーション・データへのポインタ

説明

関数 ippiCMYKToYCCK422LS_MCU は、ippj.h ファイルの中で宣言される。この関数は、関数 ippiCMYKToYCCK_JPEG と同じ方法で CMYK 画像を YCCK 画像に変換する。

また、この関数は、128 を引いてレベル・シフト操作を行って、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へデータを変換し、422 MCU を作成する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

CMYKToYCCK411LS_MCU

CMYK 画像を YCCK カラー・モデルに変換し、411 MCU を作成する。

```
IppStatus ippiCMYKToYCCK411LS_MCU _8u16s_C4P4R(const Ipp8u* pSrcCMYK,
    int srcStep, Ipp16s* pDstMCU[4]);
```

引数

pSrcCMYK	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstMCU	デスティネーション・データへのポインタ

説明

関数 ippiCMYKToYCCK411LS_MCU は、ippj.h ファイルの中で宣言される。この関数は、関数 ippiCMYKToYCCK_JPEG と同じ方法で CMYK 画像を YCCK 画像に変換する。

また、この関数は、128 を引いてレベル・シフト操作を行って、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へデータを変換し、411 MCU を作成する。

単純平均を求めて、サンプリングを実行する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

YCbCr444ToRGBLS_MCU

444 MCU から YCbCr 画像を作成し、それを RGB カラー・モデルに変換する。

```
IppStatus ippiYCbCr444ToRGBLS_MCU _16s8u_P3C3R(const Ipp16s*
    pSrcMCU[3], Ipp8u* pDstRGB, int dstStep);
```

引数

pSrcMCU	ソース・データへのポインタ
pDstRGB	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

説明

関数 ippiYCbCr444ToRGBLS_MCU は、ippj.h ファイルの中で宣言される。この関数は、ippiYCbCrToRGB_JPEG 関数の場合と同じ公式を使用して R、G、B の各成分値を計算して、444 MCU から YCbCr 8 × 8 画像を作成し、それを RGB 形式に変換する。

また、この関数は、128 を加算してレベル・シフト操作を行って、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へデータを変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCr422ToRGBLS_MCU

422 MCU から YCbCr 画像を作成し、それを RGB カラー・モデルに変換する。

```
IppStatus ippiYCbCr422ToRGBLS_MCU _16s8u_P3C3R(const Ipp16s*
    pSrcMCU[3], Ipp8u* pDstRGB, int dstStep);
```

引数

pSrcMCU	ソース・データへのポインタ
pDstRGB	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

説明

関数 ippiYCbCr422ToRGBLS_MCU は、ippj.h ファイルの中で宣言される。この関数は、ippiYCbCrToRGB_JPEG 関数の場合と同じ公式を使用して R、G、B の各成分値を計算して、422 MCU から YCbCr 16 × 8 画像を作成し、それを RGB 形式に変換する。

また、この関数は、128 を加算してレベル・シフト操作を行って、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へデータを変換する。

近傍値を複製して、サンプリングを実行する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCr411ToRGBLS_MCU

411 MCU から YCbCr 画像を作成し、それを RGB カラー・モデルに変換する。

```
IppStatus ippiYCbCr411ToRGBLS_MCU _16s8u_P3C3R(const Ipp16s*
    pSrcMCU[3], Ipp8u* pDstRGB, int dstStep);
```

引数

pSrcMCU	ソース・データへのポインタ
pDstRGB	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

説明

関数 ippiYCbCr411ToRGBLS_MCU は、ippj.h ファイルの中で宣言される。この関数は、ippiYCbCrToRGB_JPEG 関数の場合と同じ公式を使用して R、G、B の各成分値を計算して、411 MCU から YCbCr 16 × 16 画像を作成し、それを RGB 形式に変換する。

また、この関数は、128 を加算してレベル・シフト操作を行って、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へデータを変換する。

近傍値を複製して、サンプリングを実行する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCr444ToBGRLS_MCU

444 MCU から YCbCr 画像を作成し、それを BGR カラー・モデルに変換する。

```
IppStatus ippiYCbCr444ToBGRLS_MCU _16s8u_P3C3R(const Ipp16s*
    pSrcMCU[3], Ipp8u* pDstBGR, int dstStep);
```

引数

pSrcMCU	ソース・データへのポインタ
pDstBGR	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

説明

関数 ippiYCbCr444ToBGRLS_MCU は、ippj.h ファイルの中で宣言される。この関数は、ippiYCbCrToRGB_JPEG 関数の場合と同じ公式を使用して B、G、R の各成分値を計算して、444 MCU から YCbCr 8 × 8 画像を作成し、それを BGR 形式に変換する。

また、この関数は、128 を加算してレベル・シフト操作を行って、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へデータを変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCr422ToBGRLS_MCU

422 MCU から YCbCr 画像を作成し、それを BGR カラー・モデルに変換する。

```
IppStatus ippiYCbCr422ToBGRLS_MCU _16s8u_P3C3R(const Ipp16s*
    pSrcMCU[3], Ipp8u* pDstBGR, int dstStep);
```

引数

pSrcMCU	ソース・データへのポインタ
pDstBGR	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

説明

関数 ippiYCbCr422ToBGRLS_MCU は、ippj.h ファイルの中で宣言される。この関数は、ippiYCbCrToRGB_JPEG 関数の場合と同じ公式を使用して B、G、R の各成分値を計算して、422 MCU から YCbCr 16 × 8 画像を作成し、それを BGR 形式に変換する。

また、この関数は、128 を加算してレベル・シフト操作を行って、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へデータを変換する。

近傍値を複製して、サンプリングを実行する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

YCbCr411ToBGRLS_MCU

411 MCU から YCbCr 画像を作成し、それを BGR カラー・モデルに変換する。

```
IppStatus ippiYCbCr411ToBGRLS_MCU _16s8u_P3C3R(const Ipp16s*
    pSrcMCU[3], Ipp8u* pDstBGR, int dstStep);
```

引数

pSrcMCU	ソース・データへのポインタ
pDstBGR	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

説明

関数 ippiYCbCr411ToBGRLS_MCU は、ippj.h ファイルの中で宣言される。この関数は、ippiYCbCrToRGB_JPEG 関数の場合と同じ公式を使用して B、G、R の各成分値を計算して、411 MCU から YCbCr 16 × 16 画像を作成し、それを BGR 形式に変換する。

また、この関数は、128 を加算してレベル・シフト操作を行って、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へデータを変換する。

近傍値を複製して、サンプリングを実行する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

YCCK444ToCMYKLS_MCU

4444 MCU から YCCK 画像を作成し、それを CMYK カラー・モデルに変換する。

```
IppStatus ippiYCCK444ToCMYKLS_MCU _16s8u_P4C4R(const Ipp16s*
    pSrcMCU[4], Ipp8u* pDstCMYK, int dstStep);
```

引数

pSrcMCU	ソース・データへのポインタ
pDstCMYK	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

説明

関数 ippiYCCK444ToCMYKLS_MCU は、ippj.h ファイルの中で宣言される。この関数は、関数 ippiYCCKToCMYK_JPEG と同じ方法で、4444 MCU から YCCK 8 × 8 画像を作成し、それを CMYK カラー・モデルに変換する。

また、この関数は、128 を加算してレベル・シフト操作を行って、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へデータを変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

YCCKToCMYK422LS_MCU

4224 MCU から YCCK 画像を作成し、それを CMYK カラー・モデルに変換する。

```
IppStatus ippiYCCK422ToCMYKLS_MCU _16s8u_P4C4R(const Ipp16s*
    pSrcMCU[4], Ipp8u* pDstCMYK, int dstStep);
```

引数

pSrcMCU	ソース・データへのポインタ
pDstCMYK	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

説明

関数 ippiYCCK422ToCMYKLS_MCU は、ippj.h ファイルの中で宣言される。この関数は、関数 ippiYCCKToCMYK_JPEG と同じ方法で、4224 MCU から YCCK 16 × 8 画像を作成し、それを CMYK カラー・モデルに変換する。

また、この関数は、128 を加算してレベル・シフト操作を行って、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へデータを変換する。

近傍値を複製して、サンプリングを実行する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

YCCKToCMYK411LS_MCU

4114 MCU から YCCK 画像を作成し、それを CMYK カラー・モデルに変換する。

```
IppStatus ippiYCCK411ToCMYKLS_MCU _16s8u_P4C4R(const Ipp16s*
    pSrcMCU[4], Ipp8u* pDstCMYK, int dstStep);
```

引数

pSrcMCU	ソース・データへのポインタ
pDstCMYK	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

説明

関数 ippiYCCK411ToCMYKLS_MCU は、ippj.h ファイルの中で宣言される。この関数は、関数 ippiYCCKToCMYK_JPEG と同じ方法で、4114 MCU から YCCK 16 × 16 画像を作成し、それを CMYK カラー・モデルに変換する。

また、この関数は、128 を加算してレベル・シフト操作を行って、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へデータを変換する。

近傍値を複製して、サンプリングを実行する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

量子化関数

本節では、量子化および逆量子化の実行、量子化テーブルの作成を行う インテル IPP 関数の中でも特に JPEG コーデック専用の関数について説明する。表 15-3 に、これらの関数の一覧を示す。

表 15-3　量子化関数

関数の基本名	説明
QuantFwdRawTableInit_JPEG	品質係数に従って未処理の量子化テーブルを変更する。
QuantFwdTableInit_JPEG	エンコーダに適した形式の量子化テーブルを作成する。
QuantFwd8x8_JPEG	いくつかの DCT 係数から構成する 8 × 8 ブロックを量子化する。
QuantInvTableInit_JPEG	デコーダに適した形式の量子化テーブルを作成する。
QuantInv8x8_JPEG	8 × 8 ブロック単位で、いくつかの DCT 係数を逆量子化する。

QuantFwdRawTableInit_JPEG

品質係数に従って未処理の量子化テーブルを変更する。

```
IppStatus ippiQuantFwdRawTableInit_JPEG_8u(Ipp8u* pQuantRawTable, int
    qualityFactor);
```

引数

pQuantRawTable	未処理の量子化テーブルへのポインタ
qualityFactor	JPEG 品質係数。[1..100] の範囲でなければならない。

説明

関数 ippiQuantFwdRawTableInit_JPEG は、ippj.h ファイルの中で宣言される。この関数は、IJG JPEG ライブラリ、バージョン 6b に規定されているアルゴリズムを使用して、品質係数に従って最初の未処理の量子化テーブルを変更する。この関数は、JPEG コーデックの実装の際には使用しなくてもよい。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタのいずれかが NULL の場合のエラー状態を示す。

QuantFwdTableInit_JPEG

高速エンコードに適した量子化テーブルを
作成する。

```
IppStatus ippiQuantFwdTableInit_JPEG _8u16u(
    const Ipp8u* pQuantRawTable, Ipp16u* pQuantFwdTable);
```

引数

pQuantRawTable	未処理の量子化テーブルへのポインタ
pQuantFwdTable	エンコーダ用の量子化テーブルへのポインタ

説明

関数 ippiQuantFwdTableInit_JPEG は、ippj.h ファイルの中で宣言される。この関数は、高速エンコードに適した形式の量子化テーブルを作成する。最初の未処理の量子化テーブルはジグザグ順序になっている。この関数は、テーブル内にジグザグに並んでいる諸要素を、通常の順序 （左から右、上から下）に並べ替える。
量子化中に除算が実行されるのを避けるため、関数 ippiQuantFwdTableInit_JPEG は当該配列を 15 ビットだけスケーリングする。
ポインタ *pQuantRawTable* は、[ISO10918], *Annex B.2.4.1*, *Quantization Table Specification Syntax* の規定どおりに、64 個の要素から構成する配列を指している。
ポインタ *pQuantFwdTable* は、Ipp16u 型の値を 64 個含む配列を指している。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。

QuantFwd8x8_JPEG

いくつかの DCT 係数から構成する 8 × 8
ブロックを量子化する。

```
IppStatus ippiQuantFwd8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*
    pQuantFwdTable);
```

引数

pSrcDst	いくつかの DCT 係数から構成する 8×8ブロックへのポインタ
pQuantFwdTable	エンコーダ用の量子化テーブルへのポインタ

説明

関数 ippiQuantFwd8x8_JPEG は、ippj.h ファイルの中で宣言される。この関数は、8 × 8 ブロックを対象に、計算されたいくつかの DCT 係数を量子化する。量子化は、[ISO10918], *Annex A.3.4, DCT Coefficient Quantization and Dequantization* では次のように定義されている。

```
Sq(u,v) = round(S(u,v)/Q(u,v)),
```

ここで
- Sq(u,v) は、量子化された DCT 係数である。
- S(u,v) は、計算された DCT 係数の 1 つである。
- Q(u,v) は量子化テーブルの要素の 1 つである。

最も近い整数に丸める。除算が実行されるのを避けるため、量子化テーブルの各要素は 2^{15} だけスケーリング・アップされる。その後、除算演算は、スケーリングされた量子化テーブルの各要素といくつかの DCT 係数との乗算に置き換わる。そのいくつかの積が 15 ビットだけ元どおりにスケーリング変更される。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。

QuantInvTableInit_JPEG

高速デコードに適した量子化テーブルを作成する。

```
IppStatus ippiQuantInvTableInit_JPEG_8u16u(
    const Ipp8u* pQuantRawTable, Ipp16u* pQuantInvTable);
```

引数

pQuantRawTable	未処理の量子化テーブルへのポインタ

pQuantInvTable	デコード用の量子化テーブルへのポインタ

説明

関数 ippiQuantInvTableInit_JPEG は、ippj.h ファイルの中で宣言される。この関数は、高速デコード用の量子化テーブルを作成する。また、この関数は、テーブル内にジグザグに並んでいる要素を、通常の順序 （左から右、上から下）に並べ替える。そのほうがデコーダに適しているからである。ジグザグ順序の詳細は、[ISO10918], *Annex A.3.6*, *Zig-zag Sequence* を参照のこと。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。

QuantInv8x8_JPEG

いくつかの DCT 係数から構成する 8 × 8 ブロックを逆量子化する。

```
IppStatus ippiQuantInv8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*
    pQuantInvTable);
```

引数

pSrcDst	当該8×8ブロックのいくつかの DCT 係数へのポインタ
pQuantInvTable	デコード用の量子化テーブルへのポインタ

説明

関数 ippiQuantInv8x8_JPEG は、ippj.h ファイルの中で宣言される。この関数は、いくつかの DCT 係数から構成する 8 × 8 ブロックを逆量子化する。逆量子化は、[ISO10918], *Annex A.3.4*, *DCT Coefficient Quantization and Dequantization* に定義されている。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。

量子化、DCT、レベル・シフトを組み合わせた関数

本節では、レベル・シフト、量子化（逆量子化）、順方向（逆方向）DCT を組み合わせた関数の中でも特に JPEG コーデック専用の関数について説明する。表 15-4 に、これらの関数の一覧を示す。

表 15-4　量子化、DCT、レベル・シフトを組み合わせた関数

関数の基本名	説明
DCTQuantFwd8x8LS_JPEG	量子化と順方向 DCT を実行する。
DCTQuantInv8x8LS_JPEG	逆量子化と逆方向 DCT を実行する。

これらの関数は、以下に列挙した [ISO10918] の定義に従っていろいろな成分操作を行う。

- レベル・シフトは、[ISO10918], *Annex A.3.1, Level Shift*
- 量子化 / 逆量子化は、[ISO10918], *Annex A.3.4, DCT Coefficient Quantization and Dequantization*
- DCT は、[ISO10918], *Annex A.3.3, FDCT and IDCT*

DCTQuantFwd8x8LS_JPEG

レベル・シフト、順方向 DCT、量子化を実行する。

```
IppStatus ippiDCTQuantFwd8x8LS_JPEG_8u16s_C1R(const Ipp8u* pSrc,
    int srcStep, Ipp16s* pDst, const Ipp16u* pQuantFwdTable);
```

引数

pSrc	画像の 8 × 8 ブロックへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDst	量子化されたいくつかの DCT 係数から構成するデスティネーション 8 × 8 ブロックへのポインタ
pQuantFwdTable	デコーダ用の量子化テーブルへのポインタ

説明

関数 ippiDCTQuantFwd8x8LS_JPEG は、ippj.h ファイルの中で宣言される。この関数は、順方向 DCT、DCT 係数の量子化、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へのデータ変換を実行する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。

DCTQuantInv8x8LS_JPEG

逆量子化、逆方向 DCT、レベル・シフトを実行する。

```
IppStatus ippiDCTQuantInv8x8LS_JPEG_16s8u_C1R(const Ipp16s* pSrc,
    Ipp8u* pDst,int dstStep, const Ipp16u* pQuantInvTable);
```

引数

pSrc	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pDst	画像に含まれる出力 8 × 8 ブロックへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
pQuantInvTable	デコーダ用の量子化テーブルへのポインタ

説明

関数 ippiDCTQuantInv8x8LS_JPEG は、ippj.h ファイルの中で宣言される。この関数は、逆方向 DCT、DCT 係数の逆量子化、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へのデータ変換を実行する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。

レベル・シフト関数

本節では、レベル・シフト操作を実行するインテル IPP 関数の中でも特に JPEG コーデック専用の関数について説明する。表 15-5 に、これらの関数の一覧を示す。

表 15-5　レベル・シフト関数

関数の基本名	説明
Sub128_JPEG	符号付き表現へレベルをシフトする。
Add128_JPEG	符号なし表現へレベルをシフトする。

このレベル・シフト操作は、[ISO10918], *Annex A.3.1*, *Level Shift* の定義に従って実装されている。

Sub128_JPEG

符号なし 8u 範囲から符号付き 16s 範囲へ
データを変換する。

```
IppStatus ippiSub128_JPEG_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDst	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiSub128_JPEG は、ippj.h ファイルの中で宣言される。この関数は、ソース・イメージの ROI からサンプリングしたものをすべて、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へ変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

Add128_JPEG

サンプリングしたものを元の符号なし表現に戻す。

```
IppStatus ippiAdd128_JPEG_16s8u_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

引数

pSrc	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDst	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiAdd128_JPEG は、ippj.h ファイルの中で宣言される。この関数は、ソース・イメージからサンプリングしたものをすべて、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へ変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

サンプリング関数

本節では、サンプリングを実行するインテル IPP 関数の中でも特に JPEG コーデック専用の関数について説明する。表 15-6 に、これらの関数の一覧を示す。

表 15-6 サンプリング関数

関数の基本名	説明
SampleDownH2V1_JPEG	1 枚の画像を対象にして 2:1 比の水平サンプリングと 1:1 比の垂直サンプリングを実行する。
SampleDownH2V2_JPEG	1 枚の画像を対象にして 2:1 比の水平サンプリングと 2:1 比の垂直サンプリングを実行する。
SampleDownRowH2V1_Box_JPEG	1 つの画像の行を対象にして 2:1 比の水平サンプリングと 1:1 比の垂直サンプリングを実行する。
SampleDownRowH2V2_Box_JPEG	1 つの画像の行を対象にして 2:1 比の水平サンプリングと 2:1 比の垂直サンプリングを実行する。
SampleUpH2V1_JPEG	1 枚の画像を対象にして 1:2 比の水平サンプリングと 1:1 比の垂直サンプリングを実行する。
SampleUpH2V2_JPEG	1 枚の画像を対象にして 1:2 比の水平サンプリングと 1:2 比の垂直サンプリングを実行する。
SampleUpRowH2V1_Triangle_JPEG	1 つの画像の行を対象にして 1:2 比の水平サンプリングと 1:1 比の垂直サンプリングを実行する。
SampleUpRowH2V2_Triangle_JPEG	1 つの画像の行を対象にして 1:2 比の水平サンプリングと 1:2 比の垂直サンプリングを実行する。
SampleDown444LS_MCU	444 MCU を作成し、レベル・シフトを実行する。
SampleDown422LS_MCU	422 MCU を作成し、レベル・シフトを実行する。
SampleDown411LS_MCU	411 MCU を作成し、レベル・シフトを実行する。
SampleUp444LS_MCU	444 MCU からピクセル順序画像を作成し、レベル・シフトを実行する。
SampleUp422LS_MCU	422 MCU からピクセル順序画像を作成し、レベル・シフトを実行する。
SampleUp411LS_MCU	411 MCU からピクセル順序画像を作成し、レベル・シフトを実行する。

SampleDownH2V1_JPEG

1 枚の画像を対象にして 2:1 比の水平サンプリングと 1:1 比の垂直サンプリングを実行する。

```
IppStatus ippiSampleDownH2V1_JPEG_8u_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize
    dstSize);
```

引数

pSrc	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
srcSize	ソース ROI のサイズ（ピクセル単位）
pDst	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
dstSize	デスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiSampleDownH2V1_JPEG は、ippj.h ファイルの中で宣言される。この関数は、1 枚の画像を対象にして 2:1 比の水平サンプリングと 1:1 比の垂直サンプリングを実行する。ダウンサンプリングは、単純「ボックス」フィルタとして実行される。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

SampleDownH2V2_JPEG

1 枚の画像を対象にして 2:1 比の水平サンプリングと 2:1 比の垂直サンプリングを実行する。

```
IppStatus ippiSampleDownH2V2_JPEG_8u_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize
    dstSize);
```

引数

pSrc	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
srcSize	ソース ROI のサイズ（ピクセル単位）
pDst	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
dstSize	デスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiSampleDownH2V2_JPEG は、ippj.h ファイルの中で宣言される。この関数は、1 枚の画像を対象にして 2:1 比の水平サンプリングと 2:1 比の垂直サンプリングを実行する。
ダウンサンプリングは、単純「ボックス」フィルタとして実行される。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

SampleDownRowH2V1_Box_JPEG

1 つの画像の行を対象にして 2:1 比の水平サンプリングと 1:1 比の垂直サンプリングを実行する。

```
IppStatus ippiSampleDownRowH2V1_Box_JPEG_8u_C1(const Ipp8u* pSrc,
    int srcWidth, Ipp8u* pDst);
```

引数

pSrc	ソース・イメージの行へのポインタ
pDst	デスティネーション・イメージの行へのポインタ
srcWidth	ソース行の幅（ピクセル単位）

説明

関数 ippiSampleDownRowH2V1_Box_JPEG は、ippj.h ファイルの中で宣言される。この関数は、1 つの画像の行を対象にして 2:1 比の水平サンプリングと 1:1 比の垂直サンプリングを実行する。
ダウンサンプリングは、単純「ボックス」フィルタとして実行される。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcWidth* の値が 0 または負の場合のエラー状態を示す。

SampleDownRowH2V2_Box_JPEG

1 つの画像の行を対象にして 2:1 比の水平サンプリングと 2:1 比の垂直サンプリングを実行する。

```
IppStatus ippiSampleDownRowH2V2_Box_JPEG_8u_C1(const Ipp8u* pSrc1,
    const Ipp8u* pSrc2, int srcWidth, Ipp8u* pDst);
```

引数

pSrc1	ソース・イメージの行へのポインタ
pSrc2	ソース・イメージの次の行へのポインタ
pDst	デスティネーション・イメージの行へのポインタ
srcWidth	ソース行の幅（ピクセル単位）

説明

関数 ippiSampleDownRowH2V2_Box_JPEG は、ippj.h ファイルの中で宣言される。この関数は、1 つの画像の行を対象にして 2:1 比の水平サンプリングと 2:1 比の垂直サンプリングを実行する。
ダウンサンプリングは、単純「ボックス」フィルタとして実行される。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcWidth* の値が 0 または負の場合のエラー状態を示す。

SampleUpH2V1_JPEG

1 枚の画像を対象にして 1:2 比の水平サンプリングと 1:1 比の垂直サンプリングを実行する。

```
IppStatus ippiSampleUpH2V1_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

引数

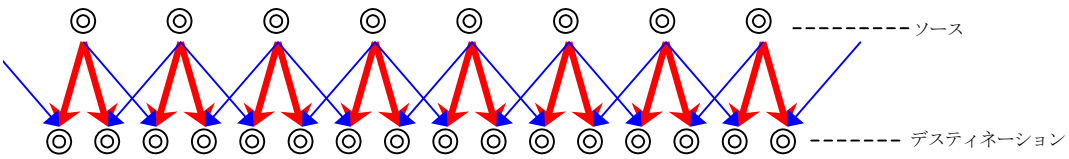
pSrc	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
srcSize	ソース ROI のサイズ（ピクセル単位）
pDst	デスティネーション・データへのポインタ

dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
dstSize	デスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiSampleUpH2V1_JPEG は、ippj.h ファイルの中で宣言される。この関数は、1 枚の画像を対象にして 1:2 比の水平サンプリングと 1:1 比の垂直サンプリングを実行する。サンプリングは、以下の図 15-1 に示す「トライアングル・フィルタ」方式で実行される。

図 15-1　トライアングル・サンプリング方式



図中の赤太線と青細線は、それぞれソース・イメージ側の ¾ と ¼ がデスティネーションのサンプリング値を占めているのを表している。

注： この関数の場合は、ソース・イメージの ROI が定義されている範囲以外でのサンプリングが若干必要である。特に、左端の列は必須であり、右端の列も必要になるときがある。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

SampleUpH2V2_JPEG

1 枚の画像を対象にして 1:2 比の水平サンプリングと 1:2 比の垂直サンプリングを実行する。

```
IppStatus ippiSampleUpH2V2_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

引数

pSrc	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
srcSize	ソース ROI のサイズ（ピクセル単位）
pDst	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
dstSize	デスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiSampleUpH2V2_JPEG は、ippj.h ファイルの中で宣言される。この関数は、1 枚の画像を対象にして 1:2 比の水平サンプリングと 1:2 比の垂直サンプリングを実行する。サンプリングは、図 15-1 に示す「トライアングル・フィルタ」方式で実行される。

注： この関数の場合は、ソース・イメージの ROI が定義されている範囲以外でのサンプリングが若干必要である。特に、左端の列と一番上の行は必須であり、右端の列と一番下の行も必要になるときがある。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	*srcStep* または *dstStep* の値が 0 または負の場合のエラー状態を示す。

SampleUpRowH2V1_Triangle_JPEG

1 つの画像の行を対象にして 1:2 比の水平サンプリングと 1:1 比の垂直サンプリングを実行する。

```
IppStatus ippiSampleUpRowH2V1_Triangle_JPEG_8u_C1(const Ipp8u* pSrc,
    int srcWidth, Ipp8u* pDst);
```

引数

pSrc	ソース・イメージの行へのポインタ
pDst	デスティネーション・イメージの行へのポインタ
srcWidth	ソース行の幅（ピクセル単位）

説明

関数 ippiSampleUpRowH2V1_Triangle_JPEG は、ippj.h ファイルの中で宣言される。この関数は、1 つの画像の行を対象にして 1:2 比の水平サンプリングと 1:1 比の垂直サンプリングを実行する。サンプリングは、図 15-1 に示す「トライアングル・フィルタ」方式で実行される。この関数は、ソース・イメージ行の外側のサンプルを必要としない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。

ippStsSizeErr	*srcWidth* の値が 0 または負の場合のエラー状態を示す。

SampleUpRowH2V2_Triangle_JPEG

1 つの画像の行を対象にして 1:2 比の水平サンプリングと 1:2 比の垂直サンプリングを実行する。

```
IppStatus ippiSampleUpRowH2V2_Triangle_JPEG_8u_C1(const Ipp8u* pSrc1,
    const Ipp8u* pSrc2, int srcWidth, Ipp8u* pDst);
```

引数

pSrc1	ソース・イメージの行へのポインタ
pSrc2	ソース・イメージの次の行へのポインタ
pDst	デスティネーション・イメージの行へのポインタ
srcWidth	ソース行の幅（ピクセル単位）

説明

関数 ippiSampleUpRowH2V2_Triangle_JPEG は、ippj.h ファイルの中で宣言される。この関数は、1 つの画像の行を対象にして 1:2 比の水平サンプリングと 1:2 比の垂直サンプリングを実行する。サンプリングは、図 15-1 に示す「トライアングル・フィルタ」方式で実行される。この関数は、ソース・イメージ行の外側のサンプルを必要としない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
ippStsSizeErr	*srcWidth* の値が 0 または負の場合のエラー状態を示す。

SampleDown444LS_MCU

ピクセル順序データから 444 MCU を作成し、レベル・シフトを実行する。

```
IppStatus ippiSampleDown444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc,
    int srcStep, Ipp16s* pDstMCU[3]));
```

引数

pSrc	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstMCU	デスティネーション・データへのポインタの配列へのポインタ。

説明

関数 ippiSampleDown444LS_MCU は、ippj.h ファイルの中で宣言される。この関数は、インターリーブ・データを、3 つの 8 × 8 ブロック（各チャネルにつき 1 ブロック）で構成される 444 MCU に変換する。

また、この関数は、128 を引いてしてレベル・シフト操作を行って、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へデータを変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

SampleDown422LS_MCU

ピクセル順序データから 422 MCU を作成し、レベル・シフトを実行する。

```
IppStatus ippiSampleDown422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc,
    int srcStep, Ipp16s* pDstMCU[3]));
```

引数

pSrc	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstMCU	デスティネーション・データへのポインタの配列へのポインタ。

説明

関数 ippiSampleDown422LS_MCU は、ippj.h ファイルの中で宣言される。この関数は、4:2:2 ダウンサンプリングを使用して、インターリーブ・データをフル MCU に変換する。したがって、最終的な MCU は、4 つのブロック（チャネル 1 に 2 つの 8 × 8 ブロック、チャネル 2 に 1 つの 8 × 8 ブロック、チャネル 3 に 1 つの 8 × 8 ブロック）で構成される。

ダウンサンプリングは、対応するピクセル値の平均を計算することによって実行される。

また、この関数は、128 を引いてしてレベル・シフト操作を行って、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へデータを変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

SampleDown411LS_MCU

ピクセル順序データから 411 MCU を作成し、レベル・シフトを実行する。

```
IppStatus ippiSampleDown411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc,
    int srcStep, Ipp16s* pDstMCU[3]));
```

引数

pSrc	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstMCU	デスティネーション・データへのポインタの配列へのポインタ。

説明

関数 ippiSampleDown411LS_MCU は、ippj.h ファイルの中で宣言される。この関数は、4:1:1 ダウンサンプリングを使用して、インターリーブ・データをフル MCU に変換する。したがって、最終的な MCU は、6 つのブロック（チャネル 1 に 4 つの 8 × 8 ブロック、チャネル 2 に 1 つの 8 × 8 ブロック、チャネル 3 に 1 つの 8 × 8 ブロック）で構成される。

ダウンサンプリングは、対応するピクセル値の平均を計算することによって実行される。

また、この関数は、128 を引いてしてレベル・シフト操作を行って、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へデータを変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

SampleUp444LS_MCU

444 MCU からピクセル順序画像を作成し、
レベル・シフトを実行する。

```
IppStatus ippiSampleUp444LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp8u* pDst, int dstStep);
```

引数

pSrcMCU	ソース・データへのポインタの配列へのポインタ
pDst	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

説明

関数 ippiSampleUp444LS_MCU は、ippi.h ファイルの中で宣言される。この関数は、444 MCU からピクセル順序画像を作成する。

また、この関数は、128 を加算してレベル・シフト操作を行って、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へデータを変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

SampleUp422LS_MCU

422 MCU からピクセル順序画像を作成し、
レベル・シフトを実行する。

```
IppStatus ippiSampleUp422LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp8u* pDst, int dstStep);
```

引数

pSrcMCU	ソース・データへのポインタの配列へのポインタ
pDst	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

説明

関数 ippiSampleUp422LS_MCU は、ippj.h ファイルの中で宣言される。この関数は、422 MCU からピクセル順序画像を作成する。アップサンプリングは、単純ボックス・フィルタとして実行される。

また、この関数は、128 を加算してレベル・シフト操作を行って、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へデータを変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

SampleUp411LS_MCU

411 MCU からピクセル順序画像を作成し、
レベル・シフトを実行する。

```
IppStatus ippiSampleUp411LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp8u* pDst, int dstStep);
```

引数

pSrcMCU	ソース・データへのポインタの配列へのポインタ
pDst	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

説明

関数 ippiSampleUp411LS_MCU は、ippi.h ファイルの中で宣言される。この関数は、411 MCU からピクセル順序画像を作成する。アップサンプリングは、単純ボックス・フィルタとして実行される。

また、この関数は、128 を加算してレベル・シフト操作を行って、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へデータを変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

プレーンからピクセルおよびピクセルからプレーンへの変換関数

本節では、プレーン（非インターリーブ）データとピクセル順序（インターリーブ）データの間の変換を行う関数について説明する。表 15-7 に、これらの関数の一覧を示す。これらの関数については、本節後半で詳しく説明する。

表 15-7　ピクセルからプレーンおよびプレーンからピクセルへの変換関数

関数の基本名	説明
Split422LS_MCU	422 インターリーブ・データから 422 MCU を作成し、レベル・シフトを実行する。
Join422LS_MCU	422 MCU から 422 インターリーブ・データを作成し、レベル・シフトを実行する。

Split422LS_MCU

422 インターリーブ・データから 422 MCU を作成し、レベル・シフトを実行する。

```
IppStatus ippiSplit422LS_MCU_8u16s_C2P3R(const Ipp8u* pSrc, int SrcStep,
     Ipp16s* pDstMCU[3]);
```

引数

pSrc	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDstMCU	デスティネーション・データへのポインタの配列へのポインタ。

説明

関数 ippiSplit422LS_MCU は、ippi.h ファイルの中で宣言される。この関数は、422 ピクセル順序データから 422 MCU を作成する。つまり、この関数は、リサンプリングを行わずにピクセル順序データをプレーン形式に変換する。最終的な MCU は、4 つのブロック（チャネル 1 に 2 つの 8 × 8 ブロック、チャネル 2 に 1 つの 8 × 8 ブロック、チャネル 3 に 1 つの 8 × 8 ブロック）で構成される。

また、この関数は、128 を引いてレベル・シフト操作を行って、符号なし Ipp8u 範囲 [0..255] から符号付き Ipp16s 範囲 [-128..127] へデータを変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタのいずれかが NULL の場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

Join422LS_MCU

422 MCU から 422 インターリーブ・データを作成し、レベル・シフトを実行する。

```
IppStatus ippiJoin422LS_MCU_16s8u_P3C2R(const Ipp16s* pSrcMCU[3], Ipp8u*
     pDst, int dstStep);
```

引数

pSrcMCU	ソース・データへのポインタの配列へのポインタ
pDst	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）

説明

関数 ippiJoin422LS_MCU は、ippi.h ファイルの中で宣言される。この関数は、422 MCU から 422 インターリーブ・データを作成する。つまり、この関数は、リサンプリングを行わずにプレーン・データをピクセル順序形式に変換する。出力データは、ピクセル Ch1Ch2Ch1Ch3... の反復シーケンスになる。

また、この関数は、128 を加算してレベル・シフト操作を行って、符号付き Ipp16s 範囲 [-128..127] から符号なし Ipp8u 範囲 [0..255] へデータを変換する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタのいずれかが NULL の場合のエラー状態を示す。

ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

ハフマン・コーデック関数

本節では、ハフマン方式のエンコード / デコード関数の中でも特に JPEG コーデック専用の関数について説明する。表 15-8 に、これらの関数の一覧を示す。

表 15-8　ハフマン・コーデック関数

関数の基本名	説明
エンコーダ関数	
EncodeHuffmanRawTableInit_JPEG	ハフマン符号の統計情報を使用して、未処理のハフマン・テーブルをいくつか作成する。
EncodeHuffmanSpecGetBufSize_JPEG	IppiEncodeHuffmanSpec 構造体の長さを返す。
EncodeHuffmanSpecInit_JPEG	エンコーダに適した形式のハフマン・テーブルを作成する。
EncodeHuffmanSpecInitAlloc_JPEG	メモリを割り当て、エンコーダに適した形式のハフマン・テーブルを作成する。
EncodeHuffmanSpecFree_JPEG	ippiEncodeHuffmanSpecInitAlloc_JPEG_8u 関数によって割り当てられたメモリを解放する。
EncodeHuffmanStateGetBufSize_JPEG	IppiEncodeHuffmanState 構造体の長さを返す。
EncodeHuffmanStateInit_JPEG	IppiEncodeHuffmanState 構造体を初期化する。
EncodeHuffmanStateInitAlloc_JPEG	メモリを割り当て、IppiEncodeHuffmanState 構造体を初期化する。
EncodeHuffmanStateFree_JPEG	ippiEncodeHuffmanStateInitAlloc_JPEG 関数によって割り当てられたメモリを解放する。
EncodeHuffman8x8_JPEG	量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックのハフマン・ベースライン・エンコードを実行する。
EncodeHuffman8x8_Direct_JPEG	量子化された DCT 係数で構成される 8x8 ブロックのハフマン・ベースライン・エンコードを直接に実行する。
GetHuffmanStatistics8x8_JPEG	ベースライン・エンコードができるように、ハフマン符号の統計情報を計算する。
GetHuffmanStatistics8x8_DCFirst_JPEG	プログレッシブ・エンコードができるように、ハフマン符号の統計情報を計算する（DC 係数）。
GetHuffmanStatistics8x8_ACFirst_JPEG	プログレッシブ・エンコードができるように、ハフマン符号の統計情報を計算する（AC 係数、最初の走査）。
GetHuffmanStatistics8x8_ACRefine_JPEG	プログレッシブ・エンコードができるように、ハフマン符号の統計情報を計算する（AC 係数、後続走査）。
EncodeHuffman8x8_DCFirst_JPEG	量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出した DC 係数のプログレッシブ・エンコードを実行する（最初の走査）。
EncodeHuffman8x8_DCRefine_JPEG	量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出した DC 係数のプログレッシブ・エンコードを実行する（後続走査）。

表 15-8　ハフマン・コーデック関数　（続き）

EncodeHuffman8x8_ACFirst_JPEG	量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出したいくつかの AC 係数のプログレッシブ・エンコードを実行する（最初の走査）。
EncodeHuffman8x8_ACRefine_JPEG	量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出したいくつかの AC 係数のプログレッシブ・エンコードを実行する（後続走査）。
デコーダ関数	
DecodeHuffmanSpecGetBufSize_JPEG	IppiDecodeHuffmanSpec 構造体の長さを返す。
DecodeHuffmanSpecInit_JPEG	デコーダに適した形式のハフマン・テーブルを作成する。
DecodeHuffmanSpecInitAlloc_JPEG	メモリを割り当て、デコーダに適した形式のハフマン・テーブルを作成する。
DecodeHuffmanSpecFree_JPEG	ippiDecodeHuffmanSpecInitAlloc_JPEG 関数によって割り当てられたメモリを解放する。
DecodeHuffmanStateGetBufSize_JPEG	IppiDecodeHuffmanState 構造体の長さを返す。
DecodeHuffmanStateInit_JPEG	IppiDecodeHuffmanState 構造体を初期化する。
DecodeHuffmanStateInitAlloc_JPEG	メモリを割り当て、IppiDecodeHuffmanState 構造体を初期化する。
DecodeHuffmanStateFree_JPEG	ippiDecodeHuffmanStateInitAlloc_JPEG 関数によって割り当てられたメモリを解放する。
DecodeHuffman8x8_JPEG	量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックのハフマン・ベースライン・デコードを実行する。
DecodeHuffman8x8_Direct_JPEG	量子化された DCT 係数で構成される 8 × 8 ブロックのハフマン・ベースライン・デコードを直接に実行する。
DecodeHuffman8x8_DCFirst_JPEG	量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出した DC 係数のプログレッシブ・デコードを実行する（最初の走査）。
DecodeHuffman8x8_DCRefine_JPEG	量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出した DC 係数のプログレッシブ・デコードを実行する（後続走査）。
DecodeHuffman8x8_ACFirst_JPEG	量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出したいくつかの AC 係数のプログレッシブ・デコードを実行する（最初の走査）。
DecodeHuffman8x8_ACRefine_JPEG	量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出したいくつかの AC 係数のプログレッシブ・デコードを実行する（後続走査）。

EncodeHuffmanRawTableInit_JPEG

ハフマン符号の統計情報を使用して、未処理のハフマン・テーブルをいくつか作成する。

```
IppStatus ippiEncodeHuffmanRawTableInit_JPEG_8u(const int
    pStatistics[256], Ipp8u* pListBits, Ipp8u* pListVals);
```

引数

pStatistics	ハフマン符号の統計情報が入っているバッファへのポインタ
pListBits	Bits リストへのポインタ
pListVals	Vals リストへのポインタ

説明

関数 ippiEncodeHuffmanRawTableInit_JPEG は、ippj.h ファイルの中で宣言される。この関数は、前に計算されたハフマン符号の統計情報を使用してハフマン・テーブルをいくつか作成する。統計情報用の配列には、ハフマン符号の値でインデックスが付く。この配列内の各値は、当該符号の出現頻度である。

Bits リストと Vals リストについては、[ISO10918], *Annex B, Figure B.7* に規定されている。

Bits リストと Vals リストを生成する際、この関数は [ISO10918], *Annex K.2, A Procedure for Generating the Lists, Which Specify a Huffman Code Table* に記載された手順を使用する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。

EncodeHuffmanSpecGetBufSize_JPEG

IppiEncodeHuffmanSpec 構造体の長さを返す。

```
IppStatus ippiEncodeHuffmanSpecGetBufSize_JPEG_8u(int* size);
```

引数

size	IppiEncodeHuffmanSpec 構造体の長さを取り込む変数へのポインタ

説明

関数 ippiEncodeHuffmanSpecGetBufSize_JPEG は、ippj.h ファイルの中で宣言される。この関数は、IppiEncodeHuffmanSpec 構造体の長さを計算する。

注： 構造体 IppiEncodeHuffmanSpec の各フィールドは、ユーザからは見えない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	当該ポインタが NULL の場合のエラー状態を示す。

EncodeHuffmanSpecInit_JPEG

エンコーダに適した形式のハフマン・テーブルを作成する。

```
IppStatus ippiEncodeHuffmanSpecInit_JPEG_8u(const Ipp8u* pListBits,
    const Ipp8u* pListVals, IppiEncodeHuffmanSpec* pEncHuffSpec);
```

引数

pListBits	Bits リストへのポインタ

pListVals	Vals リストへのポインタ
pEncHuffSpec	エンコーダ用のハフマン・テーブルへのポインタ

説明

関数 ippiEncodeHuffmanSpecInit_JPEG は、ippj.h ファイルの中で宣言される。この関数は、エンコーダに適した形式のハフマン・テーブルを作成する。
Bits リストと Vals リストについては、[ISO10918], *Annex C, Huffman table specificatio* に規定されている。
テーブルを生成する際、この関数は [ISO10918], *Annex C.2, Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths* に記載された手順を使用する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
ippStsJPEGHuffTableErr	当該テーブルの初期化中のエラー状態を示す。未処理のテーブルに許容範囲外の値が含まれていると、この状態になる場合がある。

EncodeHuffmanSpecInitAlloc_JPEG

メモリを割り当て、エンコーダに適した形式のハフマン・テーブルを作成する。

```
IppStatus ippiEncodeHuffmanSpecInitAlloc_JPEG_8u(const Ipp8u*
    pListBits, const Ipp8u* pListVals, IppiEncodeHuffmanSpec**
    ppEncHuffSpec);
```

引数

pListBits	Bits リストへのポインタ
pListVals	Vals リストへのポインタ
ppEncHuffSpec	当該エンコーダ用のハフマン・テーブルを指している戻りポインタへのポインタ

説明

関数 ippiEncodeHuffmanSpecInitAlloc_JPEG は、ippj.h ファイルの中で宣言される。この関数は、メモリを割り当て、エンコーダに適した形式のハフマン・テーブルを作成する。
Bits リストと Vals リストについては、[ISO10918], *Annex C, Huffman table specificatio* に規定されている。
テーブルを生成する際、この関数は [ISO10918], *Annex C.2, Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths* に記載された手順を使用する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
ippStsJPEGHuffTableErr	当該テーブルの初期化中のエラー状態を示す。未処理のテーブルに許容範囲外の値が含まれていると、この状態になる場合がある。

EncodeHuffmanSpecFree_JPEG

ippiEncodeHuffmanSpecInitAlloc_JPEG_8u 関数によって割り当てられたメモリを解放する。

```
IppStatus ippiEncodeHuffmanSpecFree_JPEG_8u(
    IppiEncodeHuffmanSpec*pEncHuffSpec);
```

引数

pEncHuffSpec	エンコーダ用のハフマン・テーブルへのポインタ

説明

関数 ippiEncoderHuffmanSpecFree_JPEG は、ippj.h ファイルの中で宣言される。この関数は、ハフマン・テーブル用として ippiEncoderHuffmanSpecInitAlloc_JPEG_8u 関数によって割り当てられたメモリを解放する。

戻り値

ippStsNoErr	エラーがないことを示す。

EncodeHuffmanStateGetBufSize_JPEG

IppiEncodeHuffmanState 構造体の長さ
を返す。

```
IppStatus ippiEncodeHuffmanStateGetBufSize_JPEG_8u(int* size);
```

引数

size	IppiEncodeHuffmanState 構造体の長さを取り込む変数へのポインタ

説明

関数 ippiEncodeHuffmanStateGetBufSize_JPEG は、ippj.h ファイルの中で宣言される。この関数は、IppiEncodeHuffmanState 構造体の長さ（バイト単位）を計算する。

注： 構造体 IppiEncodeHuffmanState の各フィールドは、ユーザからは見えない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタが NULL の場合のエラー状態を示す。

EncodeHuffmanStateInit_JPEG

IppiEncodeHuffmanState 構造体を初期化する。

```
IppStatus ippiEncodeHuffmanStateInit_JPEG_8u( IppiEncodeHuffmanState*
    pEncHuffState);
```

引数

pEncHuffState	IppiEncodeHuffmanState 構造体へのポインタ

説明

関数 ippiEncodeHuffmanStateInit_JPEG は、ippj.h ファイルの中で宣言される。この関数は、IppiEncodeHuffmanState をその初期状態に初期化する。この関数は、JPEG データ・ストリーム・エンコードを行う前に呼び出さなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。

EncodeHuffmanStateInitAlloc_JPEG

メモリを割り当て、IppiEncodeHuffmanState 構造体を初期化する。

```
IppStatus ippiEncodeHuffmanStateInitAlloc_JPEG_8u(
    IppiEncodeHuffmanState** ppEncHuffState);
```

引数

ppEncHuffState	IppiEncodeHuffmanState 構造体へのポインタ

説明

関数 ippiEncodeHuffmanStateInitAlloc_JPEG は、ippj.h ファイルの中で宣言される。この関数は、メモリを割り当て、IppiEncodeHuffmanState 構造体をその初期状態に初期化する。この関数は、JPEG データ・ストリーム・エンコードを行う前に呼び出さなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。

EncodeHuffmanStateFree_JPEG

ippiEncodeHuffmanStateInitAlloc_JPEG 関数によって割り当てられたメモリを解放する。

```
IppStatus ippiEncodeHuffmanStateFree_JPEG_8u(
    IppiEncodeHuffmanState* pEncHuffState);
```

引数

pEncHuffState	IppiEncodeHuffmanState 構造体へのポインタ

説明

関数 ippiEncodeHuffmanStateFree_JPEG は、ippj.h ファイルの中で宣言される。この関数は、IppiEncodeHuffmanState 構造体に割り当てられたメモリを解放する。

戻り値

ippStsNoErr	エラーがないことを示す。

EncodeHuffman8x8_JPEG

量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックのハフマン・ベースライン・エンコードを実行する。

```
IppStatus ippiEncodeHuffman8x8_JPEG_16s1u_C1(const Ipp16s* pSrc,
    Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, Ipp16s* pLastDC,
    const IppiEncodeHuffmanSpec* pDcTable, const
    IppiEncodeHuffmanSpec* pAcTable,  IppiEncodeHuffmanState*
    pEncHuffState, int bFlushState);
```

引数

pSrc	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pDst	出力ビット・ストリーム用のバッファへのポインタ
dstLenBytes	ビット・ストリーム用のバッファの長さ（バイト単位）
pDstCurrPos	出力バッファに含まれる現在のバイトのシフト量（バイト単位）
pLastDC	前の 8 × 8 ブロックの DC 係数へのポインタ
pDcTable	DC 係数用のハフマン・テーブルへのポインタ
pAcTable	AC 係数用のハフマン・テーブルへのポインタ
pEncHuffState	IppiEncodeHuffmanState 構造体へのポインタ
bFlushState	当該走査のうちの最後の 8 × 8 ブロックに関しては 1 にセットする。

説明

関数 ippiEncodeHuffman8x8_JPEG は、ippj.h ファイルの中で宣言される。この関数は、*pDcTable* テーブルと *pAcTable* テーブルを使用して、量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックをエンコードする。エンコードの手順は、[ISO10918], *Annex F.1.2, Baseline Huffman Encoding Procedures* に規定している。完全なバイトだけが出力バッファに書き込まれる。IppiEncodeHuffmanState 構造体には、各ビットが集まるが、それでは完全なバイトにはならない。IppiEncodeHuffmanState に累積されたいくつかのビットを強制的に出力バッファに追加するには、走査したうちで最後の 8 × 8 ブロックに関してパラメータ *bFlushState* を 1 にセットするか、またはエンコード期間を最初からやり直す。その他の場合は 0 にセットしなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
IppStsSizeErr	*dstLenBytes* の値が 0 または負の場合、あるいは *pDstCurrPos* が *dstLenBytes* の限界を超えた場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数の 1 つが許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。

EncodeHuffman8x8_Direct_JPEG

量子化された DCT 係数で構成される 8 × 8 ブロックのハフマン・ベースライン・エンコードを直接に実行する。

```
IppStatus ippiEncodeHuffman8x8_Direct_JPEG_16s1u_C1(const Ipp16s* pSrc,
    Ipp8u* pDst, int* pDstBitsLen, Ipp16s* pLastDC,
    const IppiEncodeHuffmanSpec* pDcTable,
    const IppiEncodeHuffmanSpec*  pAcTable);
```

引数

pSrc	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pDst	出力ビット・ストリーム用のバッファへのポインタ
pDstBitsLen	ビット・ストリーム用のバッファの長さ（バイト単位）
pLastDC	同じカラー・コンポーネントで構成される前の 8 × 8 ブロックの DC 係数へのポインタ
pDcTable	DC 係数用のハフマン・テーブルへのポインタ
pAcTable	AC 係数用のハフマン・テーブルへのポインタ

説明

関数 ippiEncodeHuffman8x8_Direct_JPEG は、ippi.h ファイルの中で宣言される。この関数は、*pDcTable* テーブルと *pAcTable* テーブルを使用して、量子化された DCT 係数で構成される 8 × 8 ブロックのエンコードを直接に実行する。この

関数は、操作用の追加の構造体を必要としない。
エンコードの手順は、[ISO10918], *Annex F.1.2, Baseline Huffman Encoding Procedures* に規定している。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。

GetHuffmanStatistics8x8_JPEG

ベースライン・エンコードができるように、ハフマン符号の統計情報を計算する。

```
IppStatus ippiGetHuffmanStatistics8x8_JPEG_16s_C1(const Ipp16s* pSrc,
    int pDcStatistics[256], int pAcStatistics[256],  Ipp16s* pLastDC);
```

引数

pSrc	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pDcStatistics	DC 係数用のハフマン符号統計情報バッファへのポインタ
pAcStatistics	AC 係数用のハフマン符号統計情報バッファへのポインタ
pLastDC	前の 8 × 8 ブロックの DC 係数へのポインタ

説明

関数 ippiGetHuffmanStatistics8x8_JPEG は、ippj.h ファイルの中で宣言される。この関数は、ベースライン・エンコードができるように、量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロック用のハフマン符号の統計情報を計算する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。

ippStsJPEGDCTRangeErr	DCT 係数の 1 つが許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。

GetHuffmanStatistics8x8_DCFirst_JPEG

プログレッシブ・エンコードができるように、ハフマン符号の統計情報を計算する（DC 係数）。

```
IppStatus ippiGetHuffmanStatistics8x8_DCFirst_JPEG_16s_C1(const
    Ipp16s* pSrc, int pDcStatistics[256],  Ipp16s* pLastDC, int Al);
```

引数

pSrc	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pDcStatistics	DC 係数用のハフマン符号統計情報バッファへのポインタ
pLastDC	前の 8 × 8 ブロックの DC 係数へのポインタ
Al	逐次近似によるビット位置（low）。実際の位置変換を指定する。

説明

関数 ippiGetHuffmanStatistics8x8_DCFirst_JPEG は、ippj.h ファイルの中で宣言される。この関数は、プログレッシブ・エンコードができるように、量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックを対象にハフマン符号の統計情報を計算する（最初の走査、DC 係数）。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたは両方が NULL の場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数の 1 つが許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。

GetHuffmanStatistics8x8_ACFirst _JPEG

プログレッシブ・エンコードができるように、ハフマン符号の統計情報を計算する（AC 係数、最初の走査）。

```
IppStatus ippiGetHuffmanStatistics8x8_ACFirst_JPEG_16s_C1(const
    Ipp16s* pSrc, int pAcStatistics[256], int Ss,  int Se, int Al,
    IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

引数

pSrc	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pAcStatistics	AC 係数用のハフマン符号統計情報バッファへのポインタ
Ss	スペクトル選択の開始インデックス
Se	スペクトル選択の終了インデックス
Al	逐次近似によるビット位置（low）。実際の位置変換を指定する。
pEncHuffState	IppiEncodeHuffmanState 構造体へのポインタ
bFlushState	当該走査のうちの最後の 8 × 8 ブロックに関しては 1 にセットする。

説明

関数 ippiGetHuffmanStatistics8x8_ACFirst_JPEG は、ippj.h ファイルの中で宣言される。この関数は、プログレッシブ・エンコードができるように、量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックを対象にハフマン符号の統計情報を計算する（最初の走査、AC 係数）。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数の 1 つが許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。

GetHuffmanStatistics8x8_ACRefine_JPEG

プログレッシブ・エンコードができるように、ハフマン符号の統計情報を計算する（AC 係数、後続走査）。

```
IppStatus ippiGetHuffmanStatistics8x8_ACRefine_JPEG_16s_C1(const
    Ipp16s* pSrc, int pAcStatistics[256], int Ss, int Se, int Al,
    IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

引数

pSrc	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pAcStatistics	AC 係数用のハフマン符号統計情報バッファへのポインタ
Ss	スペクトル選択の開始インデックス
Se	スペクトル選択の終了インデックス
Al	逐次近似によるビット位置（low）。実際の位置変換を指定する。
pEncHuffState	IppiEncodeHuffmanState 構造体へのポインタ
bFlushState	当該走査のうちの最後の 8 × 8 ブロックに関しては 1 にセットする。

説明

関数 ippiGetHuffmanStatistics8x8_ACRefine_JPEG は、ippj.h ファイルの中で宣言される。この関数は、プログレッシブ・エンコードができるように、量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックを対象にハフマン符号の統計情報を計算する（後続走査、AC 係数）。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
IppStsJPEGOutOfBufErr	バッファの限界値を超えた場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数の 1 つが許容範囲（-1023..1023）を超えた場合のエラー状態を示す。

EncodeHuffman8x8_DCFirst_JPEG

量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出した DC 係数のプログレッシブ・エンコードを実行する（最初の走査）。

```
IppStatus ippiEncodeHuffman8x8_DCFirst_JPEG_16s1u_C1(const Ipp16s*
    pSrc, Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, Ipp16s*
    pLastDC, int Al, IppiEncodeHuffmanSpec* pDcTable,
    IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

引数

pSrc	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pDst	出力ビット・ストリーム用のバッファへのポインタ
dstLenBytes	ビット・ストリーム用のバッファの長さ（バイト単位）
pDstCurrPos	出力バッファに含まれる現在のバイトのシフト量（バイト単位）
pLastDC	前の 8 × 8 ブロックの DC 係数へのポインタ
Al	逐次近似によるビット位置（low）。実際の位置変換を指定する。
pDcTable	DC 係数用のハフマン・テーブルへのポインタ
pEncHuffState	IppiEncodeHuffmanState 構造体へのポインタ
bFlushState	当該走査のうちの最後の 8 × 8 ブロックに関しては 1 にセットする。

説明

関数 ippiEncodeHuffman8x8_DCFirst_JPEG は、ippj.h ファイルの中で宣言される。この関数は、*pDcTable* テーブルを使用して、量子化されたいくつかの係数から構成する 8 × 8 ブロックから取り出した DC 係数をエンコードする（プログレッシブ・エンコード、最初の走査）。このエンコード手順は、[ISO10918], *Annex G.1.2, Progressive Encoding Procedures with Huffman* に準拠している。
完全なバイトだけが出力バッファに書き込まれる。IppiEncodeHuffmanState 構造体には、各ビットが集まるが、それでは完全なバイトにはならない。IppiEncodeHuffmanState に累積されたいくつかのビットを強制的に出力バッファに追加するには、走査したうちで最後の 8 × 8 ブロックに関してパラメータ *bFlushState* を 1 にセットするか、またはエンコード期間を最初からやり直す。その他の場合は 0 にセットしなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
IppStsJPEGOutOfBufErr	バッファの限界値を超えた場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数が許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。

EncodeHuffman8x8_DCRefine_JPEG

量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出した DC 係数のプログレッシブ・エンコードを実行する（後続走査）。

```
IppStatus ippiEncodeHuffman8x8_DCRefine_JPEG_16s1u_C1(const Ipp16s*
    pSrc, Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, int Al,
    IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

引数

pSrc	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pDst	出力ビット・ストリーム用のバッファへのポインタ

dstLenBytes	ビット・ストリーム用のバッファの長さ（バイト単位）
pDstCurrPos	出力バッファに含まれる現在のバイトのシフト量（バイト単位）
Al	逐次近似によるビット位置（low）。実際の位置変換を指定する。
pEncHuffState	IppiEncodeHuffmanState 構造体へのポインタ
bFlushState	当該走査のうちの最後の 8 × 8 ブロックに関しては 1 にセットする。

説明

関数 ippiEncodeHuffman8x8_DCRefine_JPEG は、ippj.h ファイルの中で宣言される。この関数は、*pDcTable* テーブルを使用して、量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出した DC 係数をエンコードする（プログレッシブ・エンコード、後続走査）。
このエンコード手順は、[ISO10918], *Annex G.1.2, Progressive Encoding Procedures with Huffman* に準拠している。
完全なバイトだけが出力バッファに書き込まれる。IppiEncodeHuffmanState 構造体には、各ビットが集まるが、それでは完全なバイトにはならない。IppiEncodeHuffmanState に累積されたいくつかのビットを強制的に出力バッファに追加するには、走査したうちで最後の 8 × 8 ブロックに関してパラメータ *bFlushState* を 1 にセットするか、またはエンコード期間を最初からやり直す。その他の場合は 0 にセットしなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
IppStsJPEGOutOfBufErr	バッファの限界値を超えた場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数が許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。

EncodeHuffman8x8_ACFirst_JPEG

量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出したいくつかの AC 係数のプログレッシブ・エンコードを実行する（最初の走査）。

```
IppStatus ippiEncodeHuffman8x8_ACFirst_JPEG_16s1u_C1(const Ipp16s*
    pSrc, Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, int Ss, int Se,
    int Al, IppiEncodeHuffmanSpec* pAcTable, IppiEncodeHuffmanState*
    pEncHuffState, int bFlushState);
```

引数

pSrc	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pDst	出力ビット・ストリーム用のバッファへのポインタ
dstLenBytes	ビット・ストリーム用のバッファの長さ（バイト単位）
pDstCurrPos	出力バッファに含まれる現在のバイトのシフト量（バイト単位）
Ss	スペクトル選択の開始インデックス
Se	スペクトル選択の終了インデックス
Al	逐次近似によるビット位置（low）。実際の位置変換を指定する。
pAcTable	AC 係数用のハフマン・テーブルへのポインタ
pEncHuffState	IppiEncodeHuffmanState 構造体へのポインタ
bFlushState	当該走査のうちの最後の 8 × 8 ブロックに関しては 1 にセットする。

説明

関数 ippiEncodeHuffman8x8_ACFirst_JPEG は、ippj.h ファイルの中で宣言される。この関数は、*pAcTable* テーブルを使用して、量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出したいくつかの AC 係数をエンコードする（プログレッシブ・エンコード、最初の走査）。
このエンコード手順は、[ISO10918], *Annex G.1.2, Progressive Encoding Procedures with Huffman* に準拠している。
完全なバイトだけが出力バッファに書き込まれる。IppiEncodeHuffmanState 構造体には、各ビットが集まるが、それでは完全なバイトにはならない。IppiEncodeHuffmanState に累積されたいくつかのビットを強制的に出力バッファに追加するには、走査したうちで最後の 8 × 8 ブロックに関してパラメータ *bFlushState* を 1 にセットするか、またはエンコード期間を最初からやり直す。その他の場合は 0 にセットしなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
IppStsJPEGOutOfBufErr	バッファの限界値を超えた場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数が許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。

EncodeHuffman8x8_ACRefine_JPEG

量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出したいくつかの AC 係数のプログレッシブ・エンコードを実行する。

```
IppStatus ippiEncodeHuffman8x8_ACRefine_JPEG_16s1u_C1(const Ipp16s*
    pSrc, Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, int Ss, int
    Se, int Al, IppiEncodeHuffmanSpec*  pAcTable,
    IppiEncodeHuffmanState* pEncHuffState,  int bFlushState);
```

引数

pSrc	量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックへのポインタ

pDst	出力ビット・ストリーム用のバッファへのポインタ
dstLenBytes	ビット・ストリーム用のバッファの長さ（バイト単位）
pDstCurrPos	出力バッファに含まれる現在のバイトのシフト量（バイト単位）
Ss	スペクトル選択の開始インデックス
Se	スペクトル選択の終了インデックス
Al	逐次近似によるビット位置（low）。実際の位置変換を指定する。
pAcTable	AC 係数用のハフマン・テーブルへのポインタ
pEncHuffState	IppiEncodeHuffmanState 構造体へのポインタ
bFlushState	当該走査のうちの最後の 8 × 8 ブロックに関しては 1 にセットする。

説明

関数 ippiEncodeHuffman8x8_ACRefine_JPEG は、ippj.h ファイルの中で宣言される。この関数は、*pAcTable* テーブルを使用して、量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出した AC 係数をエンコードする（プログレッシブ・エンコード、後続走査）。
このエンコード手順は、[ISO10918], *Annex G.1.2, Progressive Encoding Procedures with Huffman* に準拠している。
完全なバイトだけが出力バッファに書き込まれる。IppiEncodeHuffmanState 構造体には、各ビットが集まるが、それでは完全なバイトにはならない。IppiEncodeHuffmanState に累積されたいくつかのビットを強制的に出力バッファに追加するには、走査したうちで最後の 8 × 8 ブロックに関してパラメータ *bFlushState* を 1 にセットするか、またはエンコード期間を最初からやり直す。その他の場合は 0 にセットしなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
IppStsJPEGOutOfBufErr	バッファの限界値を超えた場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数が許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。

DecodeHuffmanSpecGetBufSize_JPEG

IppiDecodeHuffmanSpec 構造体の長さを返す。

```
IppStatus ippiDecodeHuffmanSpecGetBufSize_JPEG_8u(int* size);
```

引数

Size	IppiDecodeHuffmanSpec 構造体の長さを取り込む変数へのポインタ

説明

関数 ippiDecodeHuffmanSpecGetBufSize_JPEG は、ippj.h ファイルの中で宣言される。この関数は、IppiDecodeHuffmanSpec 構造体の長さを返す。

注： 構造体 IppiDecodeHuffmanSpec の各フィールドは、ユーザからは見えない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	当該ポインタが NULL の場合のエラー状態を示す。

DecodeHuffmanSpecInit_JPEG

デコーダに適した形式のハフマン・テーブルを作成する。

```
ippiDecodeHuffmanSpecInit_JPEG_8u(const Ipp8u* pListBits,
    const Ipp8u* pListVals, IppiDecodeHuffmanSpec* pDecHuffSpec);
```

引数

pListBits	Bits リストへのポインタ
pListVals	Vals リストへのポインタ
pDecHuffSpec	デコーダ用のハフマン・テーブルへのポインタ

説明

関数 ippiDecodeHuffmanSpecInit_JPEG は、ippj.h ファイルの中で宣言される。この関数は、デコーダに適した形式のハフマン・テーブルを作成する。
Bits リストと Vals リストは、[ISO10918], *Annex C, Huffman Table Specification* に規定されている。
テーブルを生成する際、この関数は [ISO10918], *Annex C.2, Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths* に記載した手順を使用する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
ippStsJPEGHuffTableErr	当該テーブルの初期化中のエラー状態を示す。未処理のテーブルに許容範囲外の値が含まれていると、この状態になる場合がある。

DecodeHuffmanSpecInitAlloc_JPEG

メモリを割り当て、デコーダに適した形式のハフマン・テーブルを作成する。

```
IppStatus ippiDecodeHuffmanSpecInitAlloc_JPEG_8u(const Ipp8u*
    pListBits, const Ipp8u* pListVals,  IppiDecodeHuffmanSpec**
    ppDecHuffSpec);
```

引数

pListBits	Bits リストへのポインタ
pListVals	Vals リストへのポインタ

ppDecHuffSpec	デコーダ用のハフマン・テーブルを指している戻りポインタへのポインタ

説明

関数 ippiDecodeHuffmanSpecInitAlloc_JPEG は、ippj.h ファイルの中で宣言される。この関数は、メモリを割り当て、デコーダに適した形式のハフマン・テーブルを作成する。
この関数を使用するには、[ISO10918], *Annex C, Huffman Table Specification* に規定されている形式で Bits リストと Vals リストを提供する必要がある。
テーブルを生成する際、この関数は [ISO10918], *Annex C.2, Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths* に記載された手順を使用する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
ippStsJPEGHuffTableErr	当該テーブルの初期化中のエラー状態を示す。未処理のテーブルに許容範囲外の値が含まれていると、この状態になる場合がある。

DecodeHuffmanSpecFree_JPEG

ippiDecodeHuffmanSpecInitAlloc_JPEG 関数によって割り当てられたメモリを解放する。

```
IppStatus ippiDecodeHuffmanSpecFree_JPEG_8u(IppiDecodeHuffmanSpec*
    pDecHuffSpec);
```

引数

pDecHuffSpec	デコーダ用のハフマン・テーブルへのポインタ

説明

関数 ippiDecodeHuffmanSpecFree_JPEG_8U は、ippj.h ファイルの中で宣言される。この関数は、ハフマン・テーブル用として ippiDecodeHuffmanSpecInitAlloc_JPEG_8u 関数によって割り当てられたメモリを解放する。

戻り値

ippStsNoErr	エラーがないことを示す。

DecodeHuffmanStateGetBufSize_JPEG

IppiDecodeHuffmanState 構造体の長さを返す。

```
IppStatus ippiDecodeHuffmanStateGetBufSize_JPEG_8u(int* size);
```

引数

size	IppiDecodeHuffmanState 構造体の長さを取り込む変数へのポインタ

説明

関数 ippiDecodeHuffmanStateGetBufSize_JPEG は、ippj.h ファイルの中で宣言される。この関数は、IppiDecodeHuffmanState 構造体の長さ（バイト単位）を返す。

注： 構造体 IppiDecodeHuffmanState の各フィールドは、ユーザからは見えない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定したポインタが NULL の場合のエラー状態を示す。

DecodeHuffmanStateInit_JPEG

IppiDecodeHuffmanState 構造体を初期化する。

```
IppStatus ippiDecodeHuffmanStateInit_JPEG_8u(
    IppiDecodeHuffmanState* pDecHuffState);
```

引数

pDecHuffState	IppiDecodeHuffmanState 構造体へのポインタ

説明

関数 ippiDecodeHuffmanStateInit_JPEG は、ippj.h ファイルの中で宣言される。この関数は、IppiDecoderHuffmanState 構造体をその初期状態に初期化する。この関数は、JPEG ビット・ストリーム・デコードを行う前に呼び出さなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。

DecodeHuffmanStateInitAlloc_JPEG

メモリを割り当て、IppiDecodeHuffmanState 構造体を初期化する。

```
IppStatus ippiDecodeHuffmanStateInitAlloc_JPEG_8u(
    IppiDecodeHuffmanState** ppDecHuffState);
```

引数

ppDecHuffState	IppiDecodeHuffmanState 構造体へのポインタ

説明

関数 ippiDecodeHuffmanStateInit_JPEG は、ippj.h ファイルの中で宣言される。この関数は、メモリを割り当て、IppiDecodeHuffmanState 構造体をその初期状態に初期化する。この関数は、JPEG ビット・ストリーム・デコードを行う前に呼び出さなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。

DecodeHuffmanStateFree_JPEG

ippiDecodeHuffmanStateInitAlloc_JPEG 関数によって割り当てられたメモリを解放する。

```
IppStatus ippiDecodeHuffmanStateFree_JPEG_8u(
    IppiDecodeHuffmanState* pDecHuffState);
```

引数

pDecHuffState	IppiDecodeHuffmanState 構造体へのポインタ

説明

関数 ippiDecodeHuffmanStateFree_JPEG は、ippj.h ファイルの中で宣言される。この関数は、IppiDecodeHuffmanState 構造体用として ippiDecodeHuffmanStateInitAlloc_JPEG_8u 関数によって割り当てられたメモリを解放する。

戻り値

ippStsNoErr	エラーがないことを示す。

DecodeHuffman8x8_JPEG

量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックのハフマン・ベースライン・デコードを実行する。

```
IppStatus ippiDecodeHuffman8x8_JPEG_1u16s_C1(const Ipp8u* pSrc, int
    srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, Ipp16s*  pLastDC, int*
    pMarker, const IppiDecodeHuffmanSpec*  pDcTable, const
    IppiDecodeHuffmanSpec* pAcTable,  IppiDecodeHuffmanState*
    pDecHuffState);
```

引数

pSrc	JPEG ビット・ストリームを含むソース・バッファへのポインタ
srcLenBytes	ビット・ストリーム用のバッファの長さ（バイト単位）
pSrcCurrPos	バッファに含まれる現在のバイトのシフト量（バイト単位）
pDst	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pLastDC	前の 8 × 8 ブロックの DC 係数へのポインタ
pMarker	デコード中に検出された JPEG マーカを取り込む変数へのポインタ
pDcTable	DC 係数用のハフマン・テーブルへのポインタ
pAcTable	AC 係数用のハフマン・テーブルへのポインタ
pDecHuffState	IppiDecodeHuffmanState 構造体へのポインタ

説明

関数 ippiDecodeHuffman8x8_JPEG は、ippj.h ファイルの中で宣言される。この関数は、*pDcTable* テーブルと *pAcTable* テーブルを使用して、量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックをデコードする。
このデコード手順は、[ISO10918], *Annex F.2.2, Baseline Huffman Decoding Procedures* に準拠している。
デコード中に JPEG マーカが検出されると、デコードが停止し、そのマーカが *pMarker* で指定された位置に書き込まれる。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
IppStsSizeErr	*srcLenBytes* の値が 0 または負の場合、あるいは *pSrcCurrPos* が *srcLenBytes* の限界値を超えた場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数が許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。
IppStsJPEGOutOfBufErr	バッファの限界値を超えた場合のエラー状態を示す。
IppStsJPEGMarkerWarn	JPEG マーカが検出された場合の警告を示す。

DecodeHuffman8x8_Direct_JPEG

量子化された DCT 係数で構成される 8 × 8 ブロックのハフマン・ベースライン・デコードを直接に実行する。

```
IppStatus ippiDecodeHuffman8x8_Direct_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int* pSrcBitsLen, Ipp16s* pDst, Ipp16s* pLastDC, int* pMarker,
    Ipp32u* pPrefetchedBits, int* pNumValidPrefetchedBits,
    const IppiDecodeHuffmanSpec* pDcTable,
    const IppiDecodeHuffmanSpec* pAcTable);
```

引数

pSrc	JPEG ビット・ストリームを含むソース・バッファへのポインタ
pSrcBitsLen	ビット・ストリーム用のバッファの長さ（バイト単位）
pDst	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pLastDC	同じカラー・コンポーネントで構成される前の 8 × 8 ブロックの DC 係数へのポインタ
pMarker	デコード中に検出された JPEG マーカを取り込む変数へのポインタ

pPrefetchedBits	以前にデコードされたビットを格納するプリフェッチ・バッファへのポインタ
pNumValidPrefetchedBits	プリフェッチ・バッファ内の有効ビット数
pDcTable	DC 係数用のハフマン・テーブルへのポインタ
pAcTable	AC 係数用のハフマン・テーブルへのポインタ

説明

関数 ippiDecodeHuffman8x8_Direct_JPEG は、ippi.h ファイルの中で宣言される。
この関数は、*pDcTable* テーブルと *pAcTable* テーブルを使用して、量子化された DCT 係数で構成される 8 × 8 ブロックを直接デコードする。この関数は、操作用の追加の構造体を必要としない。
このデコード手順は、[ISO10918], *Annex F.2.2, Baseline Huffman Decoding Procedures* に準拠している。
デコード中に JPEG マーカが検出されると、デコードが停止し、そのマーカが *pMarker* で指定された位置に書き込まれる。

pLastDC は、関数の初期化中または各リスタート間隔の後、0 に設定される。
pMarker は、関数の初期化中または検出されたマーカの処理後、0 に設定される。
pNumValidPrefetchedBits は、1) 関数の初期化中、2) 各リスタート間隔の後、3) 検出された各マーカの処理後、0 に設定される。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
IppStsSizeErr	*pSrcBitsLen* または *pNumValidPrefetchedBits* の値が負の場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数が許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。
IppStsJPEGOutOfBufErr	バッファの限界値を超えた場合のエラー状態を示す。

DecodeHuffman8x8_DCFirst_JPEG

量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出した DC 係数のプログレッシブ・デコードを実行する（最初の走査）。

```
IppStatus ippiDecodeHuffman8x8_DCFirst_JPEG_1u16s_C1(const Ipp8u*
    pSrc, int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, Ipp16s*
    pLastDC, int* pMarker, int Al, IppiDecodeHuffmanSpec* pDcTable,
    IppiDecodeHuffmanState* pDecHuffState);
```

引数

pSrc	JPEG ビット・ストリームを含むバッファへのポインタ
srcLenBytes	ビット・ストリーム用のバッファの長さ（バイト単位）
pSrcCurrPos	バッファに含まれる現在のバイトのシフト量（バイト単位）
pDst	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pLastDC	前の 8 × 8 ブロックの DC 係数へのポインタ
pMarker	デコード中に検出された JPEG マーカを取り込む変数へのポインタ
Al	逐次近似によるビット位置（low）。実際の位置変換を指定する。
pDcTable	DC 係数用のハフマン・テーブルへのポインタ
pDecHuffState	IppiDecodeHuffmanState 構造体へのポインタ

説明

関数 ippiDecodeHuffman8x8_DCFirst_JPEG は、ippj.h ファイルの中で宣言される。この関数は、*pDcTable* テーブルを使用して、量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出した DC 係数をデコードする（プログレッシブ・モード、最初の走査）。

このデコード手順は、[ISO10918], *Annex G.2, Progressive Decoding of the DCT* に準拠している。

デコード中に JPEG マーカが検出されると、デコードが停止し、そのマーカが *pMarker* で指定された位置に書き込まれる。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数が許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。
IppStsJPEGOutOfBufErr	バッファの限界値を超えた場合のエラー状態を示す。
IppStsJPEGMarkerWarn	JPEG マーカが検出された場合の警告を示す。

DecodeHuffman8x8_DCRefine_JPEG

量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出した DC 係数のプログレッシブ・デコードを実行する。

```
IppStatus ippiDecodeHuffman8x8_DCRefine_JPEG_1u16s_C1(const Ipp8u*
    pSrc, int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int* pMarker,
    int Al, IppiDecodeHuffmanState* pDecHuffState);
```

引数

pSrc	JPEG ビット・ストリームを含むバッファへのポインタ
srcLenBytes	ビット・ストリーム用のバッファの長さ（バイト単位）
pSrcCurrPos	バッファに含まれる現在のバイトのシフト量（バイト単位）
pDst	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pMarker	デコード中に検出された JPEG マーカを取り込む変数へのポインタ
Al	逐次近似によるビット位置（low）。実際の位置変換を指定する。
pDecHuffState	IppiDecodeHuffmanState 構造体へのポインタ

説明

関数 ippiDecodeHuffman8x8_DCRefine_JPEG は、ippj.h ファイルの中で宣言される。この関数は、*pDcTable* テーブルを使用して、量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出した DC 係数をデコードする（プログレッシブ・モード、後続走査）。
このデコード手順は、[ISO10918], *Annex G.2, Progressive Decoding of the DCT* に準拠している。
デコード中に JPEG マーカが検出されると、デコードが停止し、そのマーカが *pMarker* で指定された位置に書き込まれる。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数が許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。
IppStsJPEGOutOfBufErr	バッファの限界値を超えた場合のエラー状態を示す。
IppStsJPEGMarkerWarn	JPEG マーカが検出された場合の警告を示す。

DecodeHuffman8x8_ACFirst_JPEG

量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出したいくつかの AC 係数のプログレッシブ・デコードを実行する（最初の走査）。

```
IppStatus ippiDecodeHuffman8x8_ACFirst_JPEG_1u16s_C1(const Ipp8u*
    pSrc, int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst,  int*
    pMarker, int Ss, int Se, int Al,  IppiDecodeHuffmanSpec* pAcTable,
    IppiDecodeHuffmanState* pDecHuffState);
```

引数

pSrc	JPEG ビット・ストリームを含むバッファへのポインタ
srcLenBytes	ビット・ストリーム用のバッファの長さ（バイト単位）
pSrcCurrPos	バッファに含まれる現在のバイトのシフト量（バイト単位）

pDst	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pMarker	デコード中に検出された JPEG マーカを取り込む変数へのポインタ
Ss	スペクトル選択の開始インデックス
Se	スペクトル選択の終了インデックス
Al	逐次近似によるビット位置（low）。実際の位置変換を指定する。
pAcTable	AC 係数用のハフマン・テーブルへのポインタ
pDecHuffState	IppiDecodeHuffmanState 構造体へのポインタ

説明

関数 ippiDecodeHuffman8x8_ACFirst_JPEG は、ippj.h ファイルの中で宣言される。この関数は、*pAcTable* テーブルを使用して、量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出したいくつかの AC 係数をデコードする（プログレッシブ・モード、最初の走査）。
このデコード手順は、[ISO10918], *Annex G.2, Progressive Decoding of the DCT* に準拠している。
デコード中に JPEG マーカが検出されると、デコードが停止し、そのマーカが *pMarker* で指定された位置に書き込まれる。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数が許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。
IppStsJPEGOutOfBufErr	バッファの限界値を超えた場合のエラー状態を示す。
IppStsJPEGMarkerWarn	JPEG マーカが検出された場合の警告を示す。

DecodeHuffman8x8_ACRefine_JPEG

量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出したいくつかの AC 係数のプログレッシブ・デコードを実行する（後続走査）。

```
IppStatus ippiDecodeHuffman8x8_ACRefine_JPEG_1u16s_C1(const Ipp8u*
    pSrc, int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int*
    pMarker, int Ss, int Se, int Al, IppiDecodeHuffmanSpec* pAcTable,
    IppiDecodeHuffmanState* pDecHuffState);
```

引数

pSrc	JPEG ビット・ストリームを含むバッファへのポインタ
srcLenBytes	ビット・ストリーム用のバッファの長さ（バイト単位）
pSrcCurrPos	バッファに含まれる現在のバイトのシフト量（バイト単位）
pDst	量子化されたいくつかの DCT 係数から構成する 8×8 ブロックへのポインタ
pMarker	デコード中に検出された JPEG マーカを取り込む変数へのポインタ
Ss	スペクトル選択の開始インデックス
Se	スペクトル選択の終了インデックス
Al	逐次近似によるビット位置（low）。実際の位置変換を指定する。
pAcTable	AC 係数用のハフマン・テーブルへのポインタ
pDecHuffState	IppiDecodeHuffmanState 構造体へのポインタ

説明

関数 ippiDecodeHuffman8x8_ACRefine_JPEG は、ippj.h ファイルの中で宣言される。この関数は、*pAcTable* テーブルを使用して、量子化されたいくつかの DCT 係数から構成する 8 × 8 ブロックから取り出したいくつかの AC 係数をデコードする（プログレッシブ・モード、後続走査）。このデコード手順は、[ISO10918], *Annex G.2, Progressive Decoding of the DCT* に準拠している。

デコード中に JPEG マーカが検出されると、デコードが停止し、そのマーカが *pMarker* で指定された位置に書き込まれる。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つまたはすべてが NULL の場合のエラー状態を示す。
ippStsJPEGDCTRangeErr	DCT 係数が許容範囲 (-1023..1023) を超えた場合のエラー状態を示す。
IppStsJPEGOutOfBufErr	バッファの限界値を超えた場合のエラー状態を示す。
IppStsJPEGMarkerWarn	JPEG マーカが検出された場合の警告を示す。

ウェーブレット変換関数

本節では、JPEG 2000 画像符号化システム（[ISO15444] を参照）専用のウェーブレット変換関数について説明する。表 15-9 に、これらの関数の一覧を示す。

表 15-9　ウェーブレット変換関数

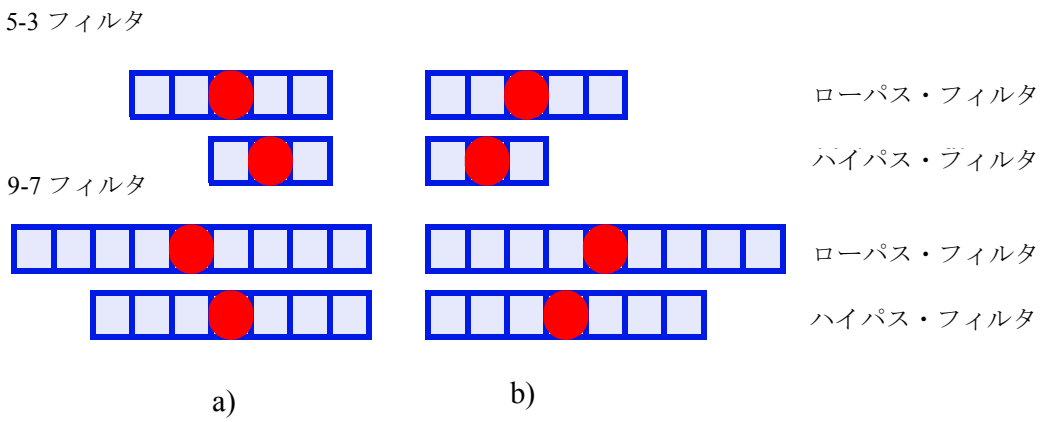
関数の基本名	説明
可逆変換	
WTFwdRow_B53_JPEG2K	行優先の順方向ウェーブレット変換を実行する。
WTInvRow_B53_JPEG2K	行優先の逆方向ウェーブレット変換を実行する。
WTFwdCol_B53_JPEG2K	列優先の順方向ウェーブレット変換を実行する。
WTInvCol_B53_JPEG2K	列優先の逆方向ウェーブレット変換を実行する。
非可逆変換	
WTFwdRow_D97_JPEG2K	行優先の順方向ウェーブレット変換を実行する。
WTInvRow_D97_JPEG2K	行優先の逆方向ウェーブレット変換を実行する。
WTFwdCol_D97_JPEG2K	列優先の順方向ウェーブレット変換を実行する。
WTInvRow_D97_JPEG2K	列優先の逆方向ウェーブレット変換を実行する。

ウェーブレット変換関数は、[ISO15444], *Annex F, Discrete Wavelet Transformation of Tile-Components* に従って、1 次元の可逆 （_B53 修飾子）および非可逆 （_D97 修飾子）ウェーブレット変換を実行する。可逆ウェーブレット変換はデータ損失がない圧縮に使用され、非可逆ウェーブレット変換はデータ損失がある圧縮に使用される。これらの変換は、5-3 可逆ウェーブレット・フィルタおよび 9-7 非可逆ウェーブレット・フィルタによるフィルタリングを、リフティング・ベースで実行する。これらのフィルタは奇数の長さになり、ローパス・フィルタの長さは 5 （9）、ハイパス・フィルタの長さは 3 （7）である。図 15-2 に示すように、ローパス・フィルタとハイパス・フィルタには、ISO15444 規格で認められた 2 種類の相対位置がある。この相対位置は、*phase* 引数によって指定される。この引数の値は、次の 2 つのいずれかである。

`ippWTFilterFirstLow` は、位置 a）に対応する。
`ippWTFilterFirstHigh` は、位置 b）に対応する。

ウェーブレット変換関数は、ソース・イメージ ROI に対して、固定ボーダ拡張 （対称、ラップアラウンドなど）を適用しない。代わりに、ウェーブレット変換関数は、ソース・イメージ ROI の外側にアクセス可能で有効なボーダ・データを要求する。

図 15-2　フィルタの可能な相対位置



WTFwdRow_B53_JPEG2K

画像の行の順方向ウェーブレット変換を実行する。

```
IppStatus ippiWTFwdRow_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc,
    int srcStep, Ipp16s* pDstLow,  int dstLowStep, Ipp16s* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
IppStatus ippiWTFwdRow_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc,
    int srcStep, Ipp32s* pDstLow,  int dstLowStep, Ipp32s* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

引数

pSrc	ソース・イメージ ROI へのポインタ
srcStep	ソース・イメージの次の行へのステップ（バイト単位）
pDstLow	デスティネーション・イメージの低周波数成分の ROI へのポインタ
dstLowStep	デスティネーション・イメージの低周波数成分の次の行へのステップ（バイト単位）
pDstHigh	デスティネーション・イメージの高周波数成分の ROI へのポインタ
dstHighStep	デスティネーション・イメージの高周波数成分の次の行へのステップ（バイト単位）
dstRoiSize	デスティネーション・イメージ ROI のサイズ
phase	ハイパス・フィルタとローパス・フィルタの相対位置（15-92 ページを参照）

説明

関数 ippiWTFwdRow_B53_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、5-3 可逆フィルタを使用して、ソース・イメージ行のウェーブレット分解を実行する。両方のデスティネーション ROI は同じサイズ *dstRoiSize* であるが、ソース・イメージ ROI のサイズは次の関係によって個別に指定される。

```
srcRoiSize.width = 2 * dstRoiSize.width;

srcRoiSize.height = dstRoiSize.height
```

この関数は、適切に動作するために、ソース・イメージ ROI の外側に以下の有効なデータを必要とする。

phase 引数が ippWTFilterFirstLow に等しい場合、この関数は、処理される各行について、(ROI ボーダの外側に) 左側に 2 ピクセル、右側に 1 ピクセルの追加を必要とする。

phase 引数が ippWTFilterFirstHigh に等しい場合、この関数は、処理される各行について、(ROI ボーダの外側に) 左側に 1 ピクセル、右側に 2 ピクセルの追加を必要とする。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsSizeErr	ROI の値の幅また高さが 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

WTInvRow_B53_JPEG2K

画像の行の逆方向ウェーブレット変換を実行する。

```
IppStatus ippiWTInvRow_B53_JPEG2K_16s_C1R(const Ipp16s* pSrcLow,
    int srcLowStep, const Ipp16s* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp16s* pDst, int dstStep,
    IppiWTFilterFirst phase);
IppStatus ippiWTInvRow_B53_JPEG2K_32s_C1R(const Ipp32s* pSrcLow,
    int srcLowStep, const Ipp32s* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp32s* pDst, int dstStep,
    IppiWTFilterFirst phase);
```

引数

pSrcLow	ソース・イメージの低周波数成分の ROI へのポインタ
srcLowStep	ソース・イメージの低周波数成分の次の行へのステップ（バイト単位）

pSrcHigh	ソース・イメージの高周波数成分の ROI へのポインタ
srcHighStep	ソース・イメージの高周波数成分の次の行へのステップ（バイト単位）
srcRoiSize	ソース・イメージ ROI のサイズ
pDst	デスティネーション・イメージの ROI へのポインタ
dstStep	デスティネーション・イメージの次の行へのステップ（バイト単位）
phase	ハイパス・フィルタとローパス・フィルタの相対位置（15-92 ページを参照）

説明

関数 ippiWTInvRow_B53_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、5-3 可逆フィルタを使用して、ソース・イメージ行のウェーブレット分解を実行する。両方のソース ROI は同じサイズ *srcRoiSize* であるが、デスティネーション・イメージ ROI のサイズは次の関係によって個別に指定される。

dstRoiSize.width = 2 * *srcRoiSize*.width;

dstRoiSize.height = *srcRoiSize*.height

この関数は、適切に動作するために、ソース・イメージ ROI の外側に以下の有効なデータを必要とする。

低周波数成分の場合 :

phase 引数が ippWTFilterFirstLow に等しい場合、この関数は、処理される各行について、(ROI ボーダの外側に) 右側に 1 ピクセルの追加を必要とする。

phase 引数が to ippWTFilterFirstHigh に等しい場合、この関数は、処理される各行について、(ROI ボーダの外側に) 左側に 1 ピクセルの追加を必要とする。

高周波数成分の場合、この関数は、処理される各行について、(ROI ボーダの外側に) 左側に 1 ピクセル、右側に 1 ピクセルの追加を常に必要とする。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。

ippStsSizeErr	ROI の値の幅また高さが 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

WTFwdCol_B53_JPEG2K

画像の列の順方向ウェーブレット変換を実行する。

```
IppStatus ippiWTFwdCol_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc,
    int srcStep, Ipp16s* pDstLow, int dstLowStep, Ipp16s* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
IppStatus ippiWTFwdCol_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc,
    int srcStep, Ipp32s* pDstLow, int dstLowStep, Ipp32s* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

引数

pSrc	ソース・イメージ ROI へのポインタ
srcStep	ソース・イメージの次の行へのステップ（バイト単位）
pDstLow	デスティネーション・イメージの低周波数成分の ROI へのポインタ
dstLowStep	デスティネーション・イメージの低周波数成分の次の行へのステップ（バイト単位）
pDstHigh	デスティネーション・イメージの高周波数成分の ROI へのポインタ
dstHighStep	デスティネーション・イメージの高周波数成分の次の行へのステップ（バイト単位）
dstRoiSize	デスティネーション・イメージ ROI のサイズ
phase	ハイパス・フィルタとローパス・フィルタの相対位置（15-92 ページを参照）

説明

関数 ippiWTFwdCol_B53_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、5-3 可逆フィルタを使用して、ソース・イメージ列のウェーブレット分解を実行する。両方のデスティネーション ROI は同じサイズ *dstRoiSize* であるが、ソース・イメージ ROI のサイズは次の関係によって個別に指定される。

```
srcRoiSize.width = dstRoiSize.width;

srcRoiSize.height = 2 * dstRoiSize.height
```

この関数は、適切に動作するために、ソース・イメージ ROI の外側に以下の有効なデータを必要とする。

phase 引数が ippWTFilterFirstLow に等しい場合、この関数は、処理される各行について、(ROI ボーダの外側に) 上側に 2 ピクセル、下側に 1 ピクセルの追加を必要とする。

phase 引数が ippWTFilterFirstHigh に等しい場合、この関数は、処理される各行について、(ROI ボーダの外側に) 上側に 1 ピクセル、下側に 2 ピクセルの追加を必要とする。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsSizeErr	ROI の値の幅また高さが 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

WTInvCol_B53_JPEG2K

画像の列の逆方向ウェーブレット変換を実行する。

```
IppStatus ippiWTInvCol_B53_JPEG2K_16s_C1R(const Ipp16s* pSrcLow,
    int srcLowStep, const Ipp16s* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp16s* pDst, int dstStep,
    IppiWTFilterFirst phase);
IppStatus ippiWTInvCol_B53_JPEG2K_32s_C1R(const Ipp32s* pSrcLow,
    int srcLowStep, const Ipp32s* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp32s* pDst, int dstStep,
    IppiWTFilterFirst phase);
```

引数

pSrcLow	ソース・イメージの低周波数成分の ROI へのポインタ
srcLowStep	ソース・イメージの低周波数成分の次の行へのステップ（バイト単位）
pSrcHigh	ソース・イメージの高周波数成分の ROI へのポインタ
srcHighStep	ソース・イメージの高周波数成分の次の行へのステップ（バイト単位）
srcRoiSize	ソース・イメージ ROI のサイズ
pDst	デスティネーション・イメージの ROI へのポインタ
dstStep	デスティネーション・イメージの次の行へのステップ（バイト単位）
phase	ハイパス・フィルタとローパス・フィルタの相対位置（15-92 ページを参照）

説明

関数 ippiWTInvCol_B53_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、5-3 可逆フィルタを使用して、ソース・イメージ列のウェーブレット再構成を実行する。両方のソース・イメージ ROI は同じサイズ *srcRoiSize* であるが、デスティネーション・イメージ ROI のサイズは次の関係によって個別に指定される。

```
dstRoiSize.width = srcRoiSize.width;

dstRoiSize.height = 2 * srcRoiSize.height
```

この関数は、適切に動作するために、ソース・イメージ ROI の外側に以下の有効なデータを必要とする。

低周波数成分の場合 :

phase 引数が ippWTFilterFirstLow に等しい場合、この関数は、処理される各列について、(ROI ボーダの外側に) 下側に 1 ピクセルの追加を必要とする。

phase 引数が ippWTFilterFirstHigh に等しい場合、この関数は、処理される各列ついて、(ROI ボーダの外側に) 上側に 1 ピクセルの追加を必要とする。

高周波数成分の場合、この関数は、処理される各列について、(ROI ボーダの外側に) 上側に 1 ピクセル、下側に 1 ピクセルの追加を常に必要とする。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsSizeErr	ROI の値の幅また高さが 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

WTFwdRow_D97_JPEG2K

画像の行の順方向ウェーブレット変換を実行する。

```
IppStatus ippiWTFwdRow_D97_JPEG2K_32f_C1R(const Ipp32f* pSrc,
    int srcStep, Ipp32f* pDstLow,  int dstLowStep, Ipp32f* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

引数

pSrc	ソース・イメージ ROI へのポインタ
srcStep	ソース・イメージの次の行へのステップ (バイト単位)
pDstLow	デスティネーション・イメージの低周波数成分の ROI へのポインタ

dstLowStep	デスティネーション・イメージの低周波数成分の次の行へのステップ（バイト単位）
pDstHigh	デスティネーション・イメージの高周波数成分の ROI へのポインタ
dstHighStep	デスティネーション・イメージの高周波数成分の次の行へのステップ（バイト単位）
dstRoiSize	デスティネーション・イメージ ROI のサイズ
phase	ハイパス・フィルタとローパス・フィルタの相対位置（15-92 ページを参照）

説明

ippiWTFwdRow_D97_JPEG2K 関数は、9-7 非可逆フィルタを使用して、ソース・イメージ行のウェーブレット分解を実行する。両方のデスティネーション ROI は同じサイズ *dstRoiSize* であるが、ソース・イメージ ROI のサイズは次の関係によって個別に指定される。

```
srcRoiSize.width = 2 * dstRoiSize.width;

srcRoiSize.height = dstRoiSize.height
```

この関数は、適切に動作するために、ソース・イメージ ROI の外側に以下の有効なデータを必要とする。

phase 引数が ippWTFilterFirstLow に等しい場合、この関数は、処理される各行について、（ROI ボーダの外側に）左側に 4 ピクセル、右側に 3 ピクセルの追加を必要とする。

phase 引数が ippWTFilterFirstHigh に等しい場合、この関数は、処理される各行について、（ROI ボーダの外側に）左側に 3 ピクセル、右側に 4 ピクセルの追加を必要とする。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsSizeErr	ROI の値の幅また高さが 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

WTInvRow_D97_JPEG2K

画像の行の逆方向ウェーブレット変換を実行する。

```
IppStatus ippiWTInvRow_D97_JPEG2K_32f_C1R(const Ipp32f* pSrcLow,
    int srcLowStep, const Ipp32f* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp32f* pDst, int dstStep,
    IppiWTFilterFirst phase);
```

引数

pSrcLow	ソース・イメージの低周波数成分の ROI へのポインタ
srcLowStep	ソース・イメージの低周波数成分の次の行へのステップ（バイト単位）
pSrcHigh	ソース・イメージの高周波数成分の ROI へのポインタ
srcHighStep	ソース・イメージの高周波数成分の次の行へのステップ（バイト単位）
srcRoiSize	ソース・イメージ ROI のサイズ
pDst	デスティネーショション・イメージの ROI へのポインタ
dstStep	デスティネーショション・イメージの次の行へのステップ（バイト単位）
phase	ハイパス・フィルタとローパス・フィルタの相対位置（15-92 ページを参照）

説明

関数 ippiWTInvRow_D97_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、9-7 非可逆フィルタを使用して、ソース・イメージ行のウェーブレット再構成を実行する。両方のソース ROI は同じサイズ *srcRoiSize* であるが、デスティネーション・イメージ ROI のサイズは次の関係によって個別に指定される。

```
dstRoiSize.width = 2 * srcRoiSize.width;
```

```
dstRoiSize.height = srcRoiSize.height
```

この関数は、適切に動作するために、ソース・イメージ ROI の外側に以下の有効なデータを必要とする。

低周波数成分の場合 :

phase 引数が ippWTFilterFirstLow に等しい場合、この関数は、処理される各行について、(ROI ボーダの外側に) 左側に 1 ピクセル、右側に 2 ピクセルの追加を必要とする。

phase 引数が ippWTFilterFirstHigh に等しい場合、この関数は、処理される各行について、(ROI ボーダの外側に) 左側に 2 ピクセル、右側に 1 ピクセルの追加を必要とする。

高周波数成分の場合、この関数は、処理される各行について、(ROI ボーダの外側に) 左側に 2 ピクセル、右側に 2 ピクセルの追加を常に必要とする。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsSizeErr	ROI の値の幅また高さが 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

WTFwdCol_D97_JPEG2K

画像の列の順方向ウェーブレット変換を実行する。

```
IppStatus ippiWTFwdCol_D97_JPEG2K_32f_C1R(const Ipp32f* pSrc,
    int srcStep, Ipp32f* pDstLow, int dstLowStep, Ipp32f* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

引数

pSrc	ソース・イメージ ROI へのポインタ
srcStep	ソース・イメージの次の行へのステップ (バイト単位)
pDstLow	デスティネーション・イメージの低周波数成分の ROI へのポインタ

dstLowStep	デスティネーション・イメージの低周波数成分の次の行へのステップ（バイト単位）
pDstHigh	デスティネーション・イメージの高周波数成分の ROI へのポインタ
dstHighStep	デスティネーション・イメージの高周波数成分の次の行へのステップ（バイト単位）
dstRoiSize	デスティネーション・イメージ ROI のサイズ
phase	ハイパス・フィルタとローパス・フィルタの相対位置（15-92 ページを参照）

説明

関数 ippiWTFwdCol_D97_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、9-7 非可逆フィルタを使用して、ソース・イメージ列のウェーブレット分解を実行する。両方のデスティネーション ROI は同じサイズ *dstRoiSize* であるが、ソース・イメージ ROI のサイズは次の関係によって個別に指定される。

```
srcRoiSize.width = dstRoiSize.width;

srcRoiSize.height = 2 * dstRoiSize.height
```

この関数は、適切に動作するために、ソース・イメージ ROI の外側に以下の有効なデータを必要とする。

phase 引数が ippWTFilterFirstLow に等しい場合、この関数は、処理される各列について、（ROI ボーダの外側に）上側に 4 ピクセル、下側に 3 ピクセルの追加を必要とする。

phase 引数が ippWTFilterFirstHigh に等しい場合、この関数は、処理される各列について、（ROI ボーダの外側に）上側に 3 ピクセル、下側に 4 ピクセルの追加を必要とする。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsSizeErr	ROI の値の幅また高さが 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

WTInvCol_D97_JPEG2K

画像の列の逆方向ウェーブレット変換を実行する。

```
IppStatus ippiWTInvCol_D97_JPEG2K_32f_C1R(const Ipp32f* pSrcLow,
    int srcLowStep, const Ipp32f* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp32f* pDst, int dstStep,
    IppiWTFilterFirst phase);
```

引数

pSrcLow	ソース・イメージの低周波数成分の ROI へのポインタ
srcLowStep	ソース・イメージの低周波数成分の次の行へのステップ（バイト単位）
pSrcHigh	ソース・イメージの高周波数成分の ROI へのポインタ
srcHighStep	ソース・イメージの高周波数成分の次の行へのステップ（バイト単位）
srcRoiSize	ソース・イメージ ROI のサイズ
pDst	デスティネーショション・イメージの ROI へのポインタ
dstStep	デスティネーショション・イメージの次の行へのステップ（バイト単位）
phase	ハイパス・フィルタとローパス・フィルタの相対位置（15-92 ページを参照）

説明

関数 ippiWTInvCol_D97_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、9-7 非可逆フィルタを使用して、ソース・イメージ列のウェーブレット再構成を実行する。両方のソース・イメージ ROI は同じサイズ *srcRoiSize* であるが、デスティネーション・イメージ ROI のサイズは次の関係によって個別に指定される。

```
dstRoiSize.width = srcRoiSize.width;

dstRoiSize.height = 2 * srcRoiSize.height
```

この関数は、適切に動作するために、ソース・イメージ ROI の外側に以下の有効なデータを必要とする。

低周波数成分の場合 :

phase 引数が ippWTFilterFirstLow に等しい場合、この関数は、処理される各列について、(ROI ボーダの外側に) 上側に 1 ピクセル、下側に 2 ピクセルの追加を必要とする。

phase 引数が ippWTFilterFirstHigh に等しい場合、この関数は、処理される各列について、(ROI ボーダの外側に) 上側に 2 ピクセル、下側に 1 ピクセルの追加を必要とする。

高周波数成分の場合、この関数は、処理される各列について、(ROI ボーダの外側に) 上側に 2 ピクセル、下側に 2 ピクセルの追加を常に必要とする。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsSizeErr	ROI の値の幅また高さが 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

JPEG2000 エントロピー・コード化およびデコード関数

本節では、算術コード化およびデコード手順を実行する、JPEG 2000 画像符号化システム専用の画像処理関数について説明する。表 15-10 に、これらの関数の一覧を示す。

表 15-10 エントロピー・コーダ関数

関数の基本名	説明
EncodeInitAlloc_JPEG2K	エントロピー・エンコーダのステート構造体にメモリを割り当て、初期化する。
EncodeFree_JPEG2K	エントロピー・エンコード・ステート構造体に割り当てられたメモリを解放する。
EncodeLoadCodeBlock_JPEG2K	コード・ブロックをロードし、エントロピー・エンコード用のデータを準備する。
EncodeStoreBits_JPEG2K	出力コードストリームを生成する。
EncodeGetTermPassLen_JPEG2K	指定された終了したコード化パスの長さを返す。
EncodeGetRate_JPEG2K	指定されたコード化パスの目標ビット・レートの推定値を返す。
EncodeGetDist_JPEG2K	指定されたコード化パスの画像の歪みの推定値を返す。
DecodeGetBufSize_JPEG2K	デコード・ルーチン用の作業バッファのサイズを計算する。

表 15-10　エントロピー・コーダ関数

関数の基本名	説明
DecodeCodeBlock_JPEG2K	圧縮されたコード・ブロック・データをデコードする。

実装されたアダプティブ算術エンコードは、2 つの連続的な手順で実行される。最初に、準備されたコード・ブロックを関数 ippiEncodeLoadCodeBlock_JPEG2K によって前処理する。その後、関数 ippiEncodeStoreBits_JPEG2K によってエンコードの最終段階を実行し、出力コードストリームを生成する。エンコードの過程で、ステート構造体 *pState* が使用される。この構造体は、関数 ippiEncodeInitAlloc_JPEG2K によって割り当てられ、初期化される。

EncodeInitAlloc_JPEG2K

エントロピー・エンコーダのステート構造体にメモリを割り当て、初期化する。

```
IppStatus ippiEncodeInitAlloc_JPEG2K(IppiEncodeState_JPEG2K** pState,
     IppiSize codeBlockMaxSize);
```

引数

pState	割り当てられ、初期化されたエンコーダ・ステート構造体へのポインタを返す変数へのポインタ
codeBlockMaxSize	コード・ブロックの最大サイズ（ピクセル単位）

説明

関数 ippiEncodeInitAlloc_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、エンコード手順で使用されるエントロピー・エンコーダのステート構造体の割り当てと初期化を行う。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsSizeErr	コード・ブロックの幅または高さの値が 0 または負の場合のエラー状態を示す。
ippStsMemAllocErr	メモリの割り当てに失敗した場合のエラー状態を示す。

EncodeFree_JPEG2K

エントロピー・エンコード・ステート構造体に割り当てられたメモリを解放する。

```
IppStatus ippiEncodeFree_JPEG2K(IppiEncodeState_JPEG2K* pState);
```

引数

pState	割り当てられ、初期化されたエンコーダ・ステート構造体へのポインタ

説明

関数 ippiEncodeFree_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、関数 ippiEncodeInitAlloc_JPEG2K によってエントロピー・エンコーダのステート構造体に割り当てられたメモリを解放する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ステート構造体への *pState* ポインタが NULL の場合のエラー状態を示す。
ippStsContextMatchErr	ステート構造体への無効なポインタが渡された場合のエラー状態を示す。

EncodeLoadCodeBlock_JPEG2K

コード・ブロックをロードし、エントロピー・エンコード用のデータを準備する。

```
IppStatus ippiEncodeLoadCodeBlock_JPEG2K_32s_C1R(const Ipp32s* pSrc,
    int srcStep, IppiSize codeBlockSize, IppiEncodeState_JPEG2K* pState,
    IppiWTSubband subband, int magnBits, IppiMQTermination mqTermType,
    IppiMQRateAppr mqRateAppr, int codeStyleFlags, int* pSfBits,
    int* pNOfPasses, int* pNOfTermPasses);
```

引数

pSrc	ソース・コード・ブロックへのポインタ
srcStep	コード・ブロックの次の行へのステップ（バイト単位）
codeBlockSize	コード・ブロックのサイズ
pState	割り当てられ、初期化されたエンコーダ・ステート構造体へのポインタ
subband	特定のコード・ブロックに含まれるウェーブレット変換サブバンド。表 15-11 に、可能な値の一覧を示す。
magnBits	整数表現の非小数部分ビット数
codeStyleFlags	エンコード手順のオプションを指定する。表 15-12 に、可能な値の一覧を示す。
mqTermType	MQ コーダの終了モード。表 15-13 に、可能な値の一覧を示す。
mqRateAppr	ビット・レート推定モデルを指定する。現在は、非最適近似モデルに設定する 1 つの値 ippMQRateApprGood だけが利用可能である。
pSfBits	有効ビット・プレーンの数を返す変数へのポインタ
pNOfPasses	コード化パスの数を返す変数へのポインタ
pNOfTermPasses	終了したコード化パスの数を返す変数へのポインタ

説明

関数 ippiEncodeLoadCodeBlock_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、第 1 のエンコード手順を実行する。つまり、この関数は、指定されたコード・ブロックをロードして、MQ コーダの必要なパラメータを計算し、第 2 のエンコード手順に使用されるデータを準備する。
ウェーブレット変換サブバンドは、*subband* パラメータで指定される。表 15-11 に、可能な値の一覧を示す。

表 15-11　JPEG2000 エントロピー・コーダ関数のサブバンド・パラメータ

関数の基本名	説明
ippWTSubbandLxLy	x 方向および y 方向のローパス・フィルタリングによって得られるサブバンド
ippWTSubbandLxHy	x 方向のローパス・フィルタリングと y 方向のハイパス・フィルタリングによって得られるサブバンド
ippWTSubbandHxLy	x 方向のハイパス・フィルタリングと y 方向のローパス・フィルタリングによって得られるサブバンド
ippWTSubbandHxHy	x 方向および y 方向のハイパス・フィルタリングによって得られるサブバンド

表 15-12　JPEG2000 エントロピー・コーダ関数の CodeStyleFlag 引数

関数の基本名	説明
IPP_JPEG2K_VERTICALLY_CAUSAL_CONTEXT	垂直的因果コンテキストを作成するように設定する。
IPP_JPEG2K_SELECTIVE_MQ_BYPASS	選択的 MQ エンコード・バイパスを設定する。これはいくつかのコード化パス用の生のエンコードである。
IPP_JPEG2K_TERMINATE_ON_EVERY_PASS	各コード化パスの終了後に MQ コーダを終了するように設定する。
IPP_JPEG2K_RESETCTX_ON_EVERY_PASS	各コード化パスの終了後に MQ コーダのコンテキストをリセットするように設定する。
IPP_JPEG2K_USE_SEGMENTATION_SYMBOLS	エラーに対する安全性のためにセグメント化記号シーケンスを使用する。
IPP_JPEG2K_LOSSLESS_MODE	レート歪みの推定に可逆ウェーブレット変換を使用するように指定する。
IPP_JPEG2K_DEC_CONCEAL_ERRORS	最後にエラーが検出されたビット・プレーン内のエラー隠蔽を許可する。
IPP_JPEG2K_DEC_DO_NOT_CLEAR_CB	デコードの前にコード・ブロック・データをクリアしない。
IPP_JPEG2K_DEC_DO_NOT_RESET_LOW_BITS	デコードの後に下位ビットをリセットしない。
IPP_JPEG2K_DEC_DO_NOT_CLEAR_SFBUFFER	デコードの前にバッファをクリアせず、前の有効性ステートを保持する。
IPP_JPEG2K_DEC_CHECK_PRED_TERM	デコード中に、予測可能な終了が適切かどうか確認する。適切でない場合は、「コード・ブロックの損傷」エラー・コードが生成される。 この設定は、予測可能終了モードで使用される。

表 15-13　JPEG2000 エントロピー・コーダ関数の MqTermType 引数

関数の基本名	説明
ippMQTermSimple	単純終了 - 若干の追加バイトが追加される。
ippMQTermNearOptimal	準最適終了モード
ippMQTermPredictable	予測可能エラーに対する安全性のための終了モード

変数 *sfBits* は、有効ビット・プレーンの数だけを返す。例えば、すべてのソース・ピクセルが負でなく、ソース・ピクセルの最大値が 0xA （2 進数の 1010）である場合、*sfBits* は 4 有効ビットを返す。より上位のビットはコード化されない。

整数表現では、有効な非小数部分ビットだけがコード化される。
magnBits は、非小数部分ビットの数を指定する。例えば、*magnBits* = 11 の場合、最下位 20 （31 ～ 11）ビットはコード化されない。

コード・ブロック内の負の整数は、最上位ビット （最下位ビットのインデックスがゼロの場合、ゼロベースの記数法で 31 番目）に符号を指定して直接コード内で表現される。このコードは、専用の関数 ippiComplement を使用して、効率よく作成できる。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラーを示す。
ippStsSizeErr	コード・ブロックの幅または高さの値が 0 または負の場合のエラー状態を示す。
ippStsContextMatchErr	ステート構造体への無効なポインタが渡された場合のエラー状態を示す。
ippStsStepErr	*srcStep* の値が 0 または負の場合のエラー状態を示す。

EncodeStoreBits_JPEG2K

出力コードストリームを生成する。

```
IppStatus ippiEncodeStoreBits_JPEG2K_1u(Ipp8u* pDst, int* pDstLen,
     IppiEncodeState_JPEG2K* pState, int* pIsNotFinish);
```

引数

pDst	デスティネーション・データ・バッファへのポインタ
pDstLen	デスティネーション・バッファの長さへのポインタ
pState	割り当てられ、初期化されたエンコーダ・ステート構造体へのポインタ
pIsNotFinish	エンコードの完了を示す変数へのポインタ

説明

関数 ippiEncodeStoreBits_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、第 2 のエンコード手順を実行し、出力コードストリームを生成する。

アプリケーション・ノート

パラメータ *dstLen* は、読み出しと書き込みの両方に使用される。このパラメータは有効なバッファ長さを持つ関数に渡され、使用された（充填された）バッファの長さを返す。

例 15-1 は、この関数の使用例を示している。

例 15-1　ippiEncodeStoreBits_JPEG2K 関数の使用例

```
int isNotFinish = 1;
while(isNotFinish)
{
    int len = BUFFER_LEN;
    ippiEncodeStoreBits_JPEG2K_1u(buffer, &len, state, &isNotFinish);
   // You can write compressed data to the output stream, for example:
    // fwrite(buffer, sizeof(Ipp8u), len, file);

}
```

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラーを示す。
ippStsSizeErr	デスティネーション・バッファの値の幅また高さが 0 または負の場合のエラー状態を示す。
ippStsContextMatchErr	無効なステート構造体へのポインタが渡された場合のエラー状態を示す。

EncodeGetTermPassLen_JPEG2K

指定された終了したコード化パスの長さを返す。

```
IppStatus ippiEncodeGetTermPassLen_JPEG2K(IppiEncodeState_JPEG2K*
     pState, int passNumber, int* pPassLen);
```

引数

pState	割り当てられ、初期化されたエンコーダ・ステート構造体へのポインタ

passNumber	終了したコード化パスの番号
pPassLen	終了したコード化パスの長さを返す変数へのポインタ

説明

関数 ippiEncodeGetTermPassLen_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、*pState* ステート構造体に格納されている、特定の番号 *passNumber* の終了したコード化パスの長さ *pPassLen* を返す。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラーを示す。
ippStsContextMatchErr	ステート構造体への無効なポインタが渡された場合のエラー状態を示す。
ippStsJPEG2KBadPassNumber	*passNumber* の値が許容境界を超える場合のエラー状態を示す。

EncodeGetRate_JPEG2K

指定されたコード化パスのビット・レートの推定値を返す。

```
IppStatus ippiEncodeGetRate_JPEG2K(IppiEncodeState_JPEG2K* pState,
     int passNumber, int* pRate);
```

引数

pState	割り当てられ、初期化されたエンコーダ・ステート構造体へのポインタ
passNumber	指定されたコード化パスの番号
pRate	指定されたコード化パスのビット・レートの推定値を返す変数へのポインタ

説明

関数 ippiEncodeGetRate_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、特定の番号 *passNumber* のコード化パスの目標ビット・レート *pRate* の推定値を返す。ビット・レートの推定値は、エンコード手順の実行中に計算され、*pState* ステート構造体に格納される。したがって、この値は、両方のエンコード手順の完了後にのみ利用可能になる。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラーを示す。
ippStsContextMatchErr	ステート構造体への無効なポインタが渡された場合のエラー状態を示す。
ippStsJPEG2KBadPassNumber	*passNumber* の値が許容境界を超える場合のエラー状態を示す。

EncodeGetDist_JPEG2K

指定されたコード化パスの画像の歪みの推定値を返す。

```
IppStatus ippiEncodeGetDist_JPEG2K(IppiEncodeState_JPEG2K* pState,
     int passNumber, Ipp64f* pDist);
```

引数

pState	割り当てられ、初期化されたエンコーダ・ステート構造体へのポインタ
passNumber	指定されたコード化パスの番号
pDist	歪みの推定値を返す変数へのポインタ

説明

関数 ippiEncodeGetDistReduction_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、特定の番号 *passNumber* のコード化パスの再構成された画像の推定される歪み *pDist* の測定値を返す。この歪みは、[ISO15444], *Annex J, section*

J.14, Rate Control に従ってエンコード手順の実行中に計算され、*pState* 構造体に格納される。したがって、この値は、両方のエンコード手順の完了後にのみ利用可能になる。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラーを示す。
ippStsContextMatchErr	ステート構造体への無効なポインタが渡された場合のエラー状態を示す。
ippStsJPEG2KBadPassNumber	*passNumber* の値が許容境界を超える場合のエラー状態を示す。

DecodeGetBufSize_JPEG2K

デコード・ルーチン用の作業バッファのサイズを計算する。

```
IppStatus ippiDecodeGetBufSize_JPEG2K(IppiSize codeBlockMaxSize,
      int* pSize);
```

引数

codeBlockMaxSize	コード・ブロックの最大サイズ（ピクセル単位）
pSize	作業バッファのサイズを返す変数へのポインタ

説明

関数 ippiDecodeGetBufSize_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、コード・ブロックの指定された最大サイズ（ピクセル単位）に対応する、作業バッファのサイズ（バイト単位）を計算する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラーを示す。

ippStsSizeErr	コード・ブロックの幅または高さの値が 0 または負の場合のエラー状態を示す。

DecodeCodeBlock_JPEG2K

圧縮されたコード・ブロック・データをデコードする。

```
IppStatus ippiDecodeCodeBlock_JPEG2K_1u32s_C1R(const Ipp8u* pSrc,
    Ipp32s* pDst, int dstStep, IppiSize codeBlockSize,
    IppiWTSubband subband, int sfBits, int nOfPasses,
    const int* pTermPassLen, int nOfTermPasses, int codeStyleFlags,
    int* pErrorBitPlane, Ipp8u* pBuffer);
```

引数

pSrc	圧縮されたソース・コード・ブロックへのポインタ
pDst	デコードされたデータのデスティネーションへのポインタ
dstStep	デスティネーション・バッファの近傍線間のステップ（バイト単位）
codeBlockSize	コード・ブロックのサイズ
subband	特定のコード・ブロックに含まれるウェーブレット変換サブバンド。表 15-11 に、可能な値の一覧を示す。
sfBits	有効ビット・プレーンの数
nOfPasses	コード化パスの数
pTermPassLen	終了した各コード化パスの長さを格納する配列へのポインタ
nOfTermPasses	終了したコード化パスの数
codeStyleFlags	デコード手順のオプション。表 15-12 に、可能な値の一覧を示す。
pErrorBitPlane	損傷コード・ブロックの発生時に最初に返されたエラーを格納するビット・プレーンへのポインタ
pBuffer	作業バッファへのポインタ

説明

関数 ippiDecodeCodeBlock_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、アダプティブ算術デコード手順を使用して、データの圧縮されたコード・ブロック *pSrc* をデコードし、デコードされたデータを *pDst* バッファに格納する。コード・ブロック内の負の整数は、最上位ビット （最下位ビットのインデックスがゼロの場合、ゼロベースの記数法で 31 番目）に符号を指定して直接コード内で表現される。このコードは、専用の関数 ippiComplement を使用して、効率よく作成できる。

注： errorBitPlane に対応する情報が不要な場合は、*errorBitPlane* を NULL に設定する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラーを示す。
ippStsSizeErr	コード・ブロックの幅または高さの値が 0 または負の場合のエラー状態を示す。
ippStsJPEG2KDamagedCodeBlock	コード・ブロックに損傷したデータが含まれている場合のエラー状態を示す。
ippStsStepErr	*dstStep* の値が 0 または負の場合のエラー状態を示す。

コンポーネント変換関数

本節では、コンポーネント変換を実行する、JPEG 2000 画像符号化システム専用の関数について説明する。すべての関数は、[ISO15444], *Annex G, DC Level Shifting and Multiple Component Transformations* の定義に従って、コンポーネント変換を実行する。

表 15-14 に、コンポーネント変換関数の一覧を示す。これらの関数については、本節後半で詳しく説明する。

表 15-14 コンポーネント変換関数

関数の基本名	説明
RCTFwd_JPEG2K	順方向の RCT を実行する。
RCTInv_JPEG2K	逆方向の RCT を実行する。

RCTFwd_JPEG2K

順方向の可逆コンポーネント変換を実行する。

事例 1：ピクセル順序データの操作

```
IppStatus ippiRCTFwd_JPEG2K_32s_C3P3R(const Ipp32s* pSrc, int srcStep,
     Ipp32s*  pDst[3], int dstStep, IppiSize roiSize);
```

事例 2：プレーン・データのインプレース操作

```
IppStatus ippiRCTFwd_JPEG2K_32s_P3IR(Ipp32s* pSrcDst[3], int srcDstStep,
     IppiSize roiSize);
```

引数

pSrc	ソース・データへのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDst	デスティネーション・データへのポインタの配列へのポインタ。
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタの配列へのポインタ
dstSrcStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRCTFwd_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、イメージ・バッファ *pSrc* の順方向の RCT（Reversible Component Transform）を実行する。RCT は、すべての画像コンポーネント・サンプル I_0、I_1、I_2（それぞれ第 1、第 2、第 3 のコンポーネントに対応する）に適用され、以下の公式に従って、変換サンプル Y_0、Y_1、Y_2 を生成する。

$$Y_0 = \left\lfloor \frac{I_0 + 2I_1 + I_2}{4} \right\rfloor$$

$$Y_1 = I_2 - I_1$$

$$Y_2 = I_0 - I_1$$

$\lfloor x \rfloor$ の表記は、フロア関数（つまり、x を超えない範囲で最大の整数）を指定する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラーを示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

RCTInv_JPEG2K

逆方向の可逆コンポーネント変換を実行する。

事例 1：プレーン・データの操作

```
IppStatus ippiRCTInv_JPEG2K_32s_P3C3R(const Ipp32s* pSrc[3], int srcStep,
     Ipp32s* pDst, int dstStep, IppiSize roiSize);
```

事例 2：プレーン・データのインプレース操作

```
IppStatus ippiRCTInv_JPEG2K_32s_P3IR(Ipp32s* pSrcDst[3], int srcDstStep,
     IppiSize roiSize);
```

引数

pSrc	ソース・データへのポインタの配列へのポインタ
srcStep	ソース・イメージ内の近傍線間のステップ（バイト単位）
pDst	デスティネーション・データへのポインタ
dstStep	デスティネーション・イメージ内の近傍線間のステップ（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・バッファへのポインタの配列へのポインタ
dstSrcStep	（インプレース操作の場合）ソースおよびデスティネーション・イメージ・バッファ内のステップ（バイト単位）
roiSize	ソース ROI とデスティネーション ROI のサイズ（ピクセル単位）

説明

関数 ippiRCTFwd_JPEG2K は、ippi.h ファイルの中で宣言される。この関数は、逆方向のウェーブレット変換後のイメージ・バッファ *pSrc* の逆方向の RCT を実行する。逆方向の RCT は、変換された画像コンポーネント・サンプル Y_0、Y_1、Y_2 に適用され、以下の公式に従って、対応するサンプル I_0、I_1、I_2 を生成する。

$$I_1 = Y_0 - \left\lfloor \frac{Y_2 + Y_1}{4} \right\rfloor$$

$$I_0 = Y_2 + I_1$$

$$I_2 = Y_1 + I_1$$

$\lfloor x \rfloor$ の表記は、フロア関数（つまり、x を超えない範囲で最大の整数）を指定する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのいずれかが NULL の場合のエラーを示す。
ippStsSizeErr	*roiSize* フィールドの値が 0 または負の場合のエラー状態を示す。
ippStsStepErr	指定されたバッファのステップ値のいずれかが 0 または負の場合のエラー状態を示す。

H.263 ビデオ・デコーダ 16

本章では、汎用ビデオ処理、ITU-T 勧告 H.263（[ITUH263] を参照、付録のデコーダ部分（通常は「H263+ デコーダ」の略語で呼ばれる）をサポートするインテル ® IPP 関数について説明する。

実装されているインテル IPP 関数は、H.263+ デコーダの以下の要素を対象とする。

- 動きベクトルのデコード
- 逆方向の量子化、逆方向のジグザグ・ポジショニング、再構成、IDCT、「コンパクト」および「プレーン」バージョン
- ブロック・レイヤ係数のデコード（ビットストリーム解析を含む）、VLC デコード、逆方向の量子化、逆方向のジグザグ・ポジショニング、IDCT（各ステップで適切なクリッピングを実行）
- 非制限動きベクトル・モード（Annex D/H.263+）
- 高度予測モード（Annex F/H.263+）
- PB フレーム・モード（Annex G/H.263+）
- 高度 INTRA コード化モード（Annex I/H.263+）
- デブロッキング・フィルタ・モード（Annex J/H.263+）
- 修正量子化モード（Annex T/H.263+）

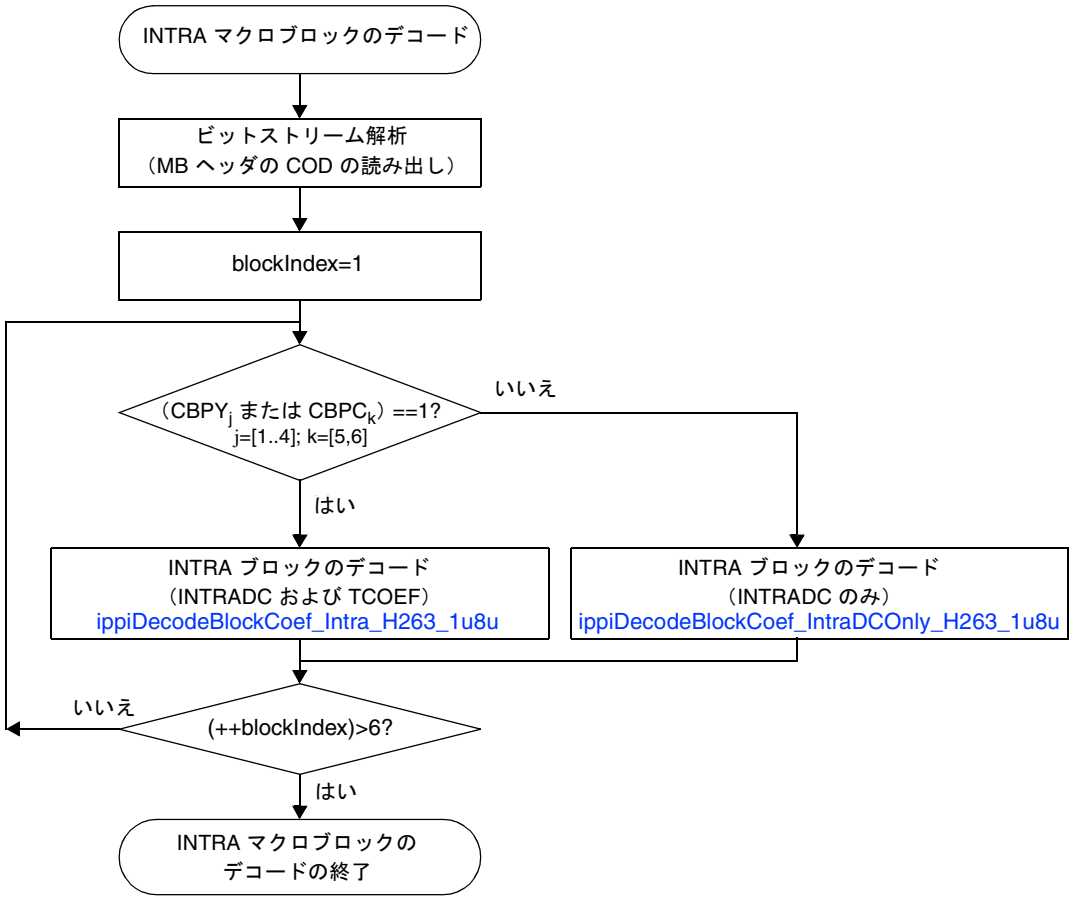
本章の各節では、H.263+ デコーダ関数階層の概要と、この関数に使用されるマクロ / データ構造体を示した後、個々の関数について詳しく説明する。

概要

INTRA マクロブロックと INTER マクロブロックのデコード

図 16-1 に、INTRA マクロブロックのデコードのプロセスを示す。

図 16-1　INTRA マクロブロックのデコード



ただし、`ippiDecodeBlockCoef_Intra_H263_1u8u` は中レベル関数である。図 16-2 に示すように、この関数の主なセグメントは、低レベルのインテル IPP 関数によって実行できる。

図 16-2　インテル® IPP 低レベル関数を使用した INTRA ブロックのデコード

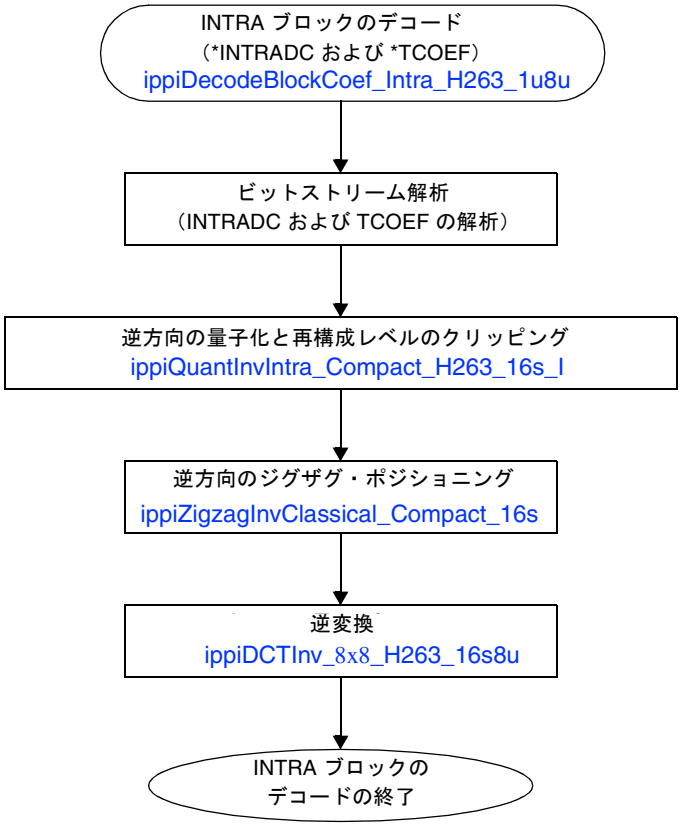
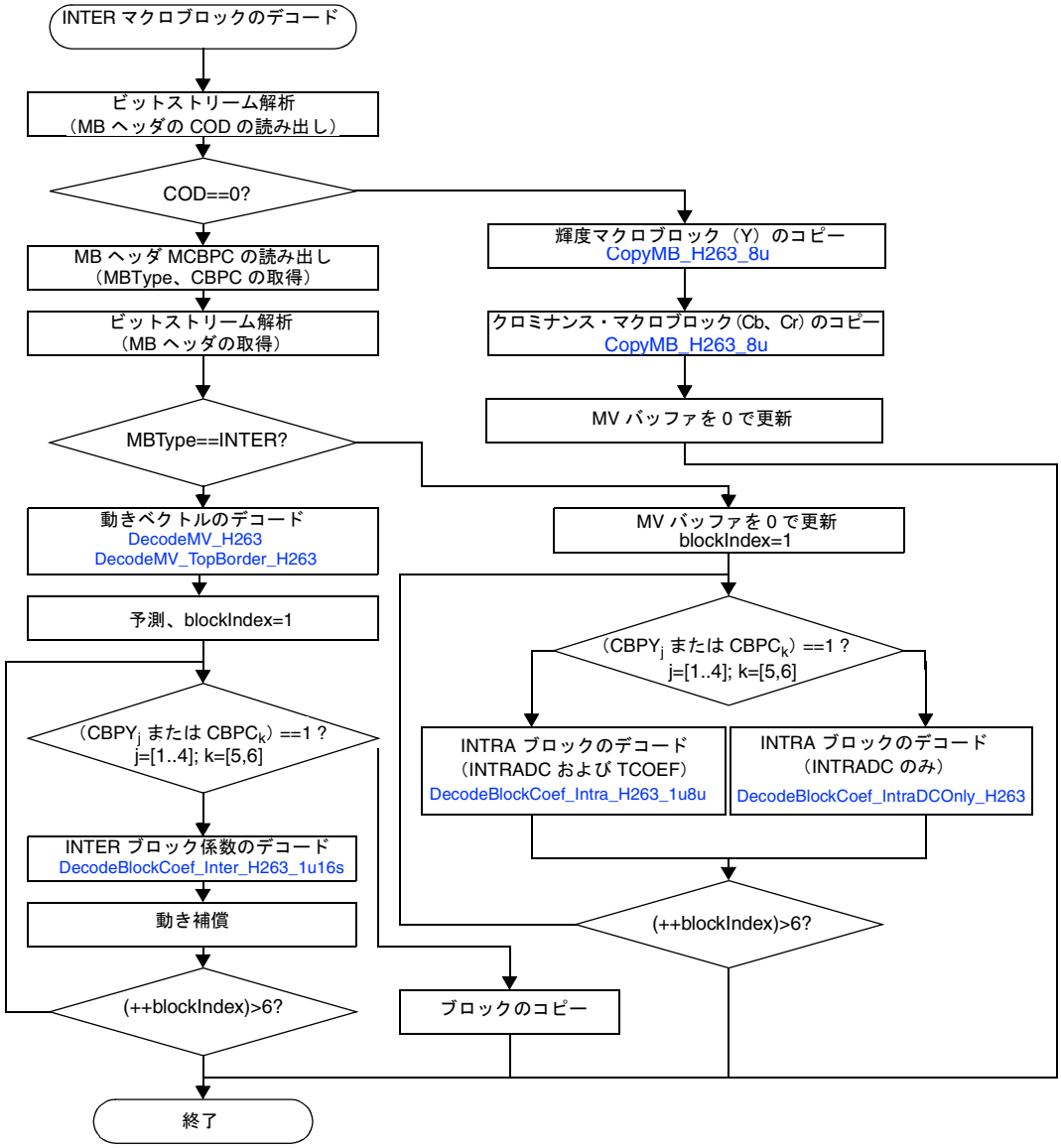


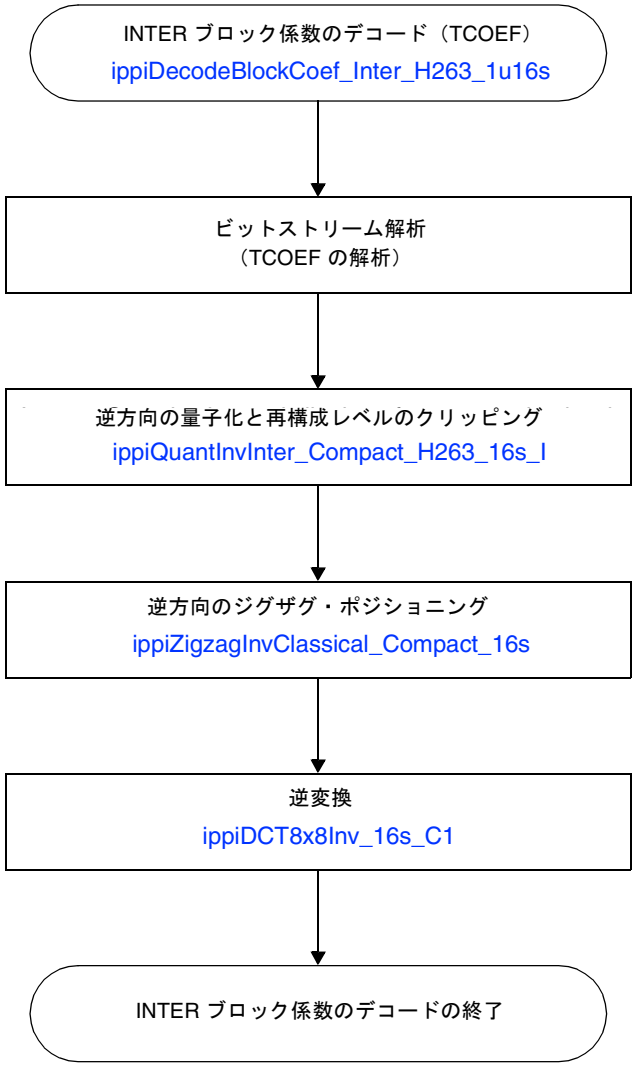
図 16-3 に、INTER マクロブロックのデコードのプロセスを示す。

図 16-3　INTER マクロブロックのデコード



ただし、ippiDecodeBlockCoef_Inter_H263_1u8u は中レベル関数である。図 16-4 に示すように、この関数の主な部分は、低レベルのインテル IPP 関数によって実行できる。

図 16-4　インテル® IPP 低レベル関数を使用した INTER ブロックのデコード



構造体とマクロの定義

動きベクトル

構造体 IppMotionVector は、H.263 ビデオ処理関数と MPEG-4 ビデオ処理関数の両方で使用できる。この構造体は、次のように定義される。

```
typedef struct {
    Ipp16s dx;
    Ipp16s dy;
} IppMotionVector;
```

ステップ

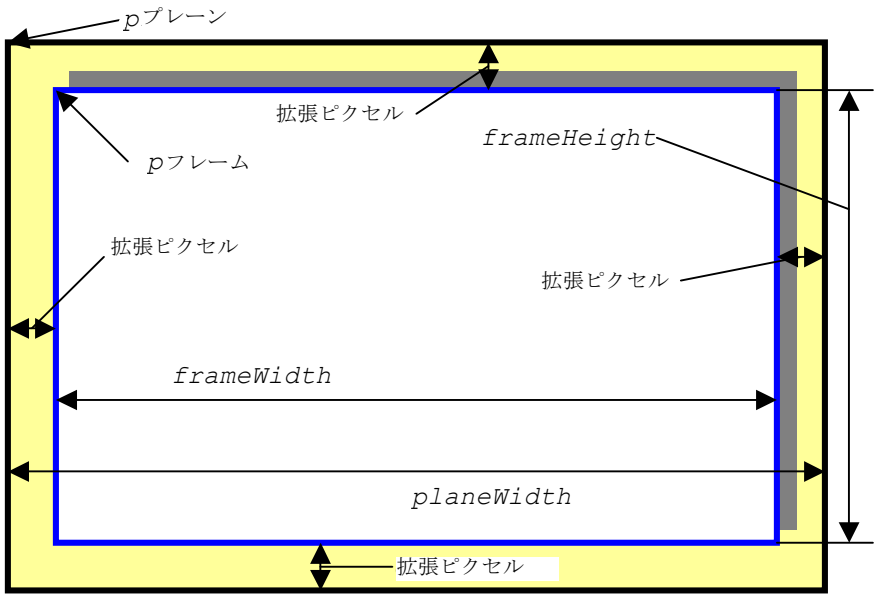
図 16-5 は、プレーン、フレーム、拡張ピクセルの概念の間の関係を示している。*Step*、*frameWidth*、*frameHeight*、*expandPels* パラメータは、次のように定義される。

- *Step* は、プレーンの幅 （ピクセル単位）である。
- *Step* >= *planeWidth* = *frameWidth* + 2 * *expandPels*;
- *frameWidth* は、フレームの幅である。
- *frameHeight* は、フレームの高さである。
- *expandPels* は、一方向に拡張されるピクセルの数である。

H.263 のデフォルトの予測モードでは、動きベクトルは、動きベクトルによって参照されるすべてのピクセルがコード化されるピクチャ領域内に入るように制限される。したがって、*expandPels* の値は 0（つまり、*Step* = *frameWidth*）の場合がある。

動きベクトルがピクチャの外側を参照できる場合は（例えば、Annex D/H.263+）、プレーンに合わせてフレームを拡張しなければならない（つまり、次の関係が成り立つ）。*Step* = *frameWidth* + 2 * *expandPels*

図 16-5　プレーン、フレーム、拡張ピクセル

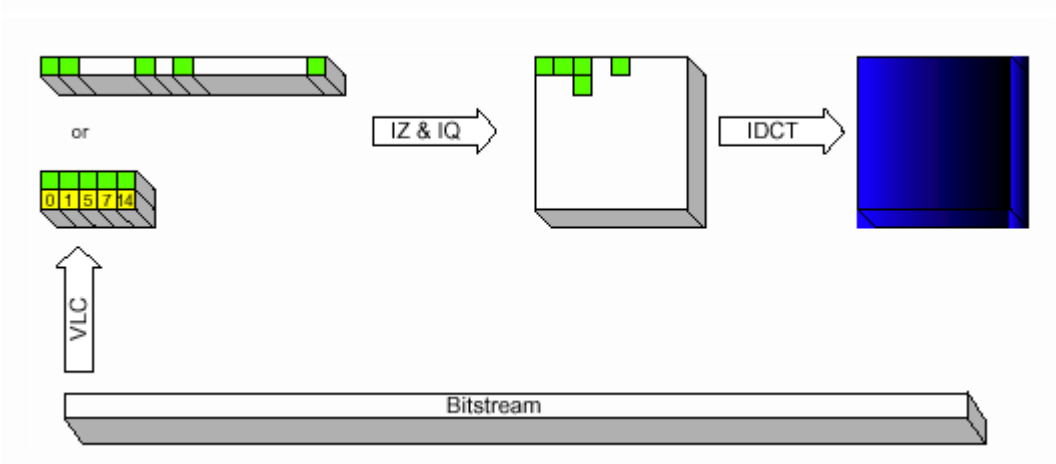


コンパクト・バッファ

H.263 規格は、トランスフォーム・コーディングを使用して、空間的な冗長性を低減する。したがって、ビットストリームから解析されるブロックの疎密度を利用して、逆方向のジグザグ・ポジショニングおよび逆方向の量子化の計算量を減らせる。0 でない最後の係数のインデックスは、可変長デコードから求められる。この場合、逆方向のジグザグ・ポジショニングおよび逆方向の量子化用のバッファの長さは 64 よりはるかに小さくなるため、このバッファは「コンパクト・バッファ」と呼ばれる。図 16-6 は、コンパクト・バッファとフル・バッファの概念を示している。

さらに、中レベルの関数では、可変長デコード、逆方向のジグザグ・ポジショニング、逆方向の量子化が、1 つのステップに統合される。0 でない 2 つの係数の間の 0 は圧縮されるため、バッファはさらにコンパクトになる。

図 16-6　コンパクト・バッファ



H.263 ビデオ・デコーダ関数

H.263 ビデオ・デコーダに関連するすべての関数の説明は、以下の節に分けられる。

汎用ビデオ処理関数と H.263 デコーダ関数

H.263+　関数

H.263+　中レベル関数

汎用ビデオ処理関数と H.263 デコーダ関数

本節では、すべての汎用ビデオ処理関数と H.263 基本デコーダ関数について説明する。汎用ビデオ処理関数は関数名の修飾子に H263 サフィックスを含まないが、すべての H.263 デコーダ関数は関数名に H263 サフィックスを含むため、2 種類の関数は簡単に区別できる。

表 16-1 に、本節で詳しく説明する関数の一覧を示す。

表 16-1　汎用ビデオ処理関数と H.263 デコーダ関数

関数の基本名	説明
DecodeMV_H263, DecodeMV_TopBorder_H263	動きベクトルをデコードする。
CopyMB_H263, CopyBlock_H263	リファレンス・マクロブロック / ブロックを現在のマクロブロック / ブロックにコピーする。
CopyApproxHMB_H263, CopyApproxHBlock_H263	水平方向の近似を使用して、リファレンス・マクロブロック / ブロックを現在のマクロブロック / ブロックにコピーする。
CopyApproxVMB_H263, CopyApproxVBlock_H263	垂直方向の近似を使用して、リファレンス・マクロブロック / ブロックを現在のマクロブロック / ブロックにコピーする。
CopyApproxHVMB_H263, CopyApproxHVBlock_H263	水平方向の近似と垂直方向の近似を使用して、リファレンス・マクロブロック / ブロックを現在のマクロブロック / ブロックにコピーする。
QuantInvIntra_Compact_H263, QuantInvInter_Compact_H263	コンパクト・バッファに格納された、コード化された INTRA または INTER ブロック上で、逆方向の量子化を実行する。
ZigzagInvClassical_Compact, ZigzagInvHorizontal_Compact, ZigzagInvVertical_Compact	コンパクト・バッファに格納されたブロック上で、標準、水平、または垂直の逆方向ジグザグ・スキャンを実行する。
ZigzagInv_Horizontal ZigzagInv_Vertical	ブロック上で水平または垂直の逆方向ジグザグ・スキャンを実行する。
DCTInv_8x8	8x8 の 2 次元逆コサイン変換を実行する。
ReconMB_H263, ReconBlock_H263	予測値と逆変換の結果 （残差）の和を求めて、INTER マクロブロック / ブロックを再構成する。

DecodeMV_H263, DecodeMV_TopBorder_H263

動きベクトルをデコードする。

```
IppStatus ippiDecodeMV_H263(Ipp8u** ppBitStream, int* pBitOffset,
      IppMotionVector* pSrcDstMV);
IppStatus ippiDecodeMV_TopBorder_H263(Ipp8u** ppBitStream,
      int* pBitOffset, IppMotionVector* pSrcDstMV);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ (**ppBitStream* はブロックのデコード後に更新される)。
pBitOffset	**ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。 **pBitOffset* はブロックのデコード後に更新される。
pSrcDstMV	動きベクトル・バッファ内の現在のマクロブロックの左側の動きベクトルへのポインタ。 **(pSrcDstMV + 1)* は、デコードされた動きベクトルに合わせて更新される。この値を使用して、次の動きベクトルを予測し、動き補償を実行する。

説明

関数 ippiDecodeMV_H263 と ippiDecodeMV_TopBorder_H263 は、ippmp.h ファイルの中で宣言される。これらの関数は、周囲の 3 つの 動きベクトル（MV）に従って現在の MV を予測し、ビットストリームから解析される差分動きベクトルに加算することによって、動きベクトルをデコードする。デコードされた動きベクトルは、動きベクトル・バッファに保存される。このバッファのサイズ（バイト単位）は、以下の式に等しい。
[number_of_macroblocks_per_row + 2]* sizeof (IppMotionVector)

この関数の呼び出しの前に、**ppBitStream* と **pBitOffset* は、ビットストリーム内の MVD の最初のビットに置かれる必要がある。

ビットストリーム・バッファの境界チェックは行われない。
詳細は、DecodeBlockCoef_Inter_H263 関数の項目で、動きベクトルのデコードの説明を参照のこと。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。

CopyMB_H263, CopyBlock_H263

リファレンス・マクロブロック / ブロックを現在のマクロブロック / ブロックにコピーする。

```
IppStatus ippiCopyMB_H263_8u(const Ipp8u* pSrc, Ipp8u* pDst,
      int step);
IppStatus ippiCopyBlock_H263_8u(const Ipp8u* pSrc, Ipp8u* pDst,
      int step);
```

引数

pSrc	現在のフレーム内のマクロブロック / ブロックに空間的に対応する、リファレンス・フレーム内のマクロブロック / ブロックへのポインタ
pDst	コピー先のマクロブロック / ブロックへのポインタ
step	ソース・プレーンとデスティネーション・プレーンの幅。

説明

関数 ippiCopyMB_H263 と ippiCopyBlock_H263 は、ippmp.h ファイルの中で宣言される。これらの関数は、リファレンス・マクロブロック / ブロックを現在のマクロブロック / ブロックにコピーする。COD（Coded Macroblock Indication）が 1 に設定されている場合は、そのマクロブロックはコード化されていない。このマクロブロックは、ブロック全体の値 0 の動きベクトルを持ち、係数データを持たない INTER マクロブロックとして扱われる。

この場合、H.263 基本仕様デコーダ内でマクロブロックのコピーだけが実行される。このマクロブロックのコピーは、次のように定義される。

*pDst[i*step + j] = pSrc[i*step + j]*

ここで、*i,j = [0,15]*

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsH263MBStepErr	*step* の値が 16 より小さい場合のエラー状態を示す。
ippStsH263BlockStepErr	*step* の値が 8 より小さい場合のエラー状態を示す。

QuantInvIntra_Compact_H263, QuantInvInter_Compact_H263

コンパクト・バッファに格納された、コード化された INTRA または INTER ブロック上で、逆方向の量子化を実行する。

```
IppStatus ippiQuantInvIntra_Compact_H263_16s_I(Ipp16s* pSrcDst,
      int len, int QP);
IppStatus ippiQuantInvInter_Compact_H263_16s_I(Ipp16s* pSrcDst,
      int len, int QP);
```

引数

pSrcDst	入力の量子化されたブロックと出力の再構成されたブロックへのポインタ
len	入力および出力コンパクト・バッファの長さ
QP	量子化パラメータ

説明

関数 ippiQuantInvIntra_Compact_H263 と ippiQuantInvInter_Compact_H263 は、ippmp.h ファイルの中で宣言される。これらの関数は、コンパクト・バッファに格納された、コード化された INTRA または INTER ブロック上で、逆方向の量子化を実行する。逆方向の量子化は、次のように定義される。

$$pDst\,[i] = \begin{cases} 0, & pSrc\,[i] = 0 \\ Sign\,(pSrc\,[i]) * QP * (2 * |pSrc\,[i]| + 1) & pSrc\,[i] \neq 0、\text{QP は奇数} \\ Sign\,(pSrc\,[i]) * (QP * (2 * |pSrc\,[i]| + 1) - 1) & pSrc\,[i] \neq 0、\text{QP は偶数} \end{cases}$$

ここで、$i = \begin{cases} 0, 1, ..., \text{len} - 1 & \text{INTER モード} \\ 1, 2, ..., \text{len} - 1 & \text{INTRA モード} \end{cases}$ 、かつ

$$pDst\,[0] = \begin{cases} 1024、 & pSrc\,[0] = 255、\text{かつ INTRA モード} \\ 8 * pSrc\,[0]、 & pSrc\,[0] = 255 \text{ 以外の場合、かつ INTRA モード} \end{cases}$$

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	*pSrcDst* ポインタが NULL の場合のエラー状態を示す。
ippStsH263QuantErr	量子化器の値が、0 か負の場合、または 31 より大きい場合のエラー状態を示す。

ZigzagInvClassical_Compact, ZigzagInvHorizontal_Compact, ZigzagInvVertical_Compact

コンパクト・バッファに格納されたブロック上で、標準、水平、または垂直の逆方向ジグザグ・スキャンを実行する。

```
IppStatus ippiZigzagInvClassical_Compact_16s(const Ipp16s* pSrc,
      int len, Ipp16s* pDst);
IppStatus ippiZigzagInvHorizontal_Compact_16s(const Ipp16s* pSrc,
      int len, Ipp16s* pDst);
IppStatus ippiZigzagInvVertical_Compact_16s(const Ipp16s* pSrc,
      int len, Ipp16s* pDst);
```

引数

pSrc	（ジグザグ・スキャンされる）入力ブロックへのポインタ
pDst	（通常の方法でスキャンされる）出力ブロックへのポインタ。出力は、64 個の要素を格納するフル・バッファに置かれる。
len	入力および出力コンパクト・バッファの長さ

説明

関数 ippiZigzagInvClassical_Compact、ippiZigzagInvHorizontal_Compact、ippiZigzagInvVertical_Compact は、ippmp.h ファイルの中で宣言される。これらの関数は、コンパクト・バッファに格納されたブロック上で、標準、水平、または垂直の逆方向ジグザグ・スキャンの順序を（それぞれ通常の順序に）変更する。以下の図 16-7 と図 16-8 に、3 種類のジグザグ・スキャン・パターンと通常のスキャン・パターンを示す（セル内の番号はスキャンの順序を示す）。

図 16-7　ジグザグ・スキャン・パターン

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

標準ジグザグ・スキャン

0	1	2	3	10	11	12	13
4	5	8	9	17	16	15	14
6	7	19	18	26	27	28	29
20	21	24	25	30	31	32	33
22	23	34	35	42	43	44	45
36	37	40	41	46	47	48	49
38	39	50	51	56	57	58	59
52	53	54	55	60	61	62	63

水平ジグザグ・スキャン

0	4	6	20	22	36	38	52
1	5	7	21	23	37	39	53
2	8	19	24	34	40	50	54
3	9	18	25	35	41	51	55
10	17	26	30	42	46	56	60
11	16	27	31	43	47	57	61
12	15	28	32	44	48	58	62
13	14	29	33	45	49	59	63

垂直ジグザグ・スキャン

図 16-8　通常のスキャン・パターン

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

逆方向スキャンとは、通常のスキャン・パターンから 1 つのジグザグ・スキャン・パターンへのマッピングである。

例えば、ブロック上で標準ジグザグ・スキャンを実行した後は、*pDst*[3] = *pSrc*[6] になる。

注： この関数の呼び出しの前に、出力バッファ pDst を 0 にする必要がある。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsH263ZigzagLenErr	*Len* の値が 64 より大きい場合のエラー状態を示す。

ZigzagInv_Horizontal ZigzagInv_Vertical

ブロック上で水平または垂直の逆方向ジグザグ・スキャンを実行する。

```
IppStatus ippiZigzagInv_Horizontal_16s(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiZigzagInv_Vertical_16s(const Ipp16s* pSrc, Ipp16s* pDst);
```

引数

pSrc	（ジグザグ・スキャンされる）入力ブロックへのポインタ
pDst	（通常の方法でスキャンされる）出力ブロックへのポインタ

説明

関数 ippiZigzagInv_Horizontal と ippiZigzagInv_Vertical は、ippmp.h ファイルの中で宣言される。これらの関数は、入力ブロック *pSrc* 上でそれぞれ水平または垂直の逆方向ジグザグ・スキャンを実行し、結果を通常のスキャン順序のブロック *pDst* に格納する。図 16-7 と図 16-8 に、水平および垂直のジグザグ・スキャン・パターンと通常のスキャン・パターンを示す（セル内の番号はスキャンの順序を示す）。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。

CopyApproxHMB_H263, CopyApproxHBlock_H263

水平方向の近似を使用して、リファレンス・マクロブロック / ブロックを現在のマクロブロック / ブロックにコピーする。

```
IppStatus ippiCopyApproxHMB_H263_8u(const Ipp8u* pSrc, Ipp8u* pDst,
      int step);
IppStatus ippiCopyApproxHBlock_H263_8u(const Ipp8u* pSrc, Ipp8u* pDst,
      int step);
```

引数

pSrc	現在のフレーム内のマクロブロック / ブロックに空間的に対応する、リファレンス・フレーム内のマクロブロック / ブロックへのポインタ
pDst	デスティネーション・マクロブロック / ブロックへのポインタ
step	ソース・プレーンとデスティネーション・プレーンの幅。

説明

関数 ippiCopyApproxHMB_H263 と ippiCopyApproxH Block_H263 は、ippmp.h ファイルの中で宣言される。これらの関数は、水平方向の近似を使用して、リファレンス・マクロブロック / ブロック *pSrc* を現在のマクロブロック / ブロック *pDst* にコピーする。デスティネーション・マクロブロック / ブロック内の各ピクセルは、ソース・マクロブロック / ブロック内の水平に並んだ 2 つのピクセルの平均値になる。この操作は丸めなしで実行され、次のように定義される。

*pDst[i*step + j] = pSrc[((i*step + j)*+(i*step + (j + 1) + 1))/2]*、
マクロブロックの場合は *i,j = [0, 15]* ブロックの場合は *i,j = [0, 7]*

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsH263MBStepErr	*step* の値が 16 より小さい場合のエラー状態を示す。
ippStsH263BlockStepErr	*step* の値が 8 より小さい場合のエラー状態を示す。

CopyApproxVMB_H263, CopyApproxVBlock_H263

垂直方向の近似を使用して、リファレンス・マクロブロック / ブロックを現在のマクロブロック / ブロックにコピーする。

```
IppStatus ippiCopyApproxVMB_H263_8u(const Ipp8u* pSrc, Ipp8u* pDst,
      int step);
IppStatus ippiCopyApproxVBlock_H263_8u(const Ipp8u* pSrc, Ipp8u* pDst,
      int step);
```

引数

pSrc	現在のフレーム内のマクロブロック / ブロックに空間的に対応する、リファレンス・フレーム内のマクロブロック / ブロックへのポインタ
pDst	デスティネーション・マクロブロック / ブロックへのポインタ
step	ソース・プレーンとデスティネーション・プレーンの幅。

説明

関数 ippiCopyApproxVMB_H263 と ippiCopyApproxVBlock_H263 は、ippmp.h ファイルの中で宣言される。これらの関数は、垂直方向の近似を使用して、リファレンス・マクロブロック / ブロック *pSrc* を現在のマクロブロック / ブロック *pDst* にコピーする。デスティネーション・マクロブロック / ブロック内の各ピクセルは、ソース・マクロブロック / ブロック内の垂直に並んだ 2 つのピクセルの平均値になる。この操作は丸めなしで実行され、次のように定義される。

*pDst[i*step + j] = pSrc[((i*step + j)*+((i + 1)*step + j) + 1)/2],*

ここで、マクロブロックの場合は *i,j = [0, 15]* ブロックの場合は *i,j = [0, 7]*

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsH263MBStepErr	*step* の値が 16 より小さい場合のエラー状態を示す。
ippStsH263BlockStepErr	*step* の値が 8 より小さい場合のエラー状態を示す。

CopyApproxHVMB_H263, CopyApproxHVBlock_H263

両方向の近似を使用して、リファレンス・マクロブロック / ブロックを現在のマクロブロック / ブロックにコピーする。

```
IppStatus ippiCopyApproxHVMB_H263_8u(const Ipp8u* pSrc, Ipp8u* pDst,
      int step);
IppStatus ippiCopyApproxHVBlock_H263_8u(const Ipp8u* pSrc, Ipp8u* pDst,
      int step);
```

引数

pSrc	現在のフレーム内のマクロブロック / ブロックに空間的に対応する、リファレンス・フレーム内のマクロブロック / ブロックへのポインタ
pDst	デスティネーション・マクロブロック / ブロックへのポインタ
step	ソース・プレーンとデスティネーション・プレーンの幅。

説明

関数 ippiCopyApproxHVMB_H263 と ippiCopyApproxHVBlock_H263 は、ippmp.h ファイルの中で宣言される。これらの関数は、垂直方向の近似と水平方向の近似を使用して、リファレンス・マクロブロック / ブロック *pSrc* を現在のマクロブロック / ブロック *pDst* にコピーする。デスティネーション・マクロブロック / ブロック内の各ピクセルは、ソース・マクロブロック / ブロック内の対応する 4 つのピクセルの平均値になる。この操作は丸めなしで実行され、次のように定義される。

```
pDst[i*step + j] = pSrc[((i*step+j) + (i*step+(j+1)) + ((i+1)*step)+j)+
                        +((i+1)*step+(j+1))+ 2)/4],
```

ここで、マクロブロックの場合は *i,j = [0, 15]*、ブロックの場合は *i,j = [0, 7]*

戻り値

ippStsNoErr	エラーがないことを示す。

ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsH263MBStepErr	*step* の値が 16 より小さい場合のエラー状態を示す。
ippStsH263BlockStepErr	*step* の値が 8 より小さい場合のエラー状態を示す。

DCTInv_8x8

8 × 8 の 2 次元逆コサイン変換を実行する。

```
IppStatus ippiDCTInv_8x8_16s8u(const Ipp16s* pSrc, Ipp8u* pDst,
      int dstStep);
```

引数

pSrc	入力 DCT 係数へのポインタ
pDst	デスティネーション・プレーン内のブロックへのポインタ
dstStep	デスティネーション・プレーンの幅

説明

関数 ippiDCTInv_8x8 は、ippmp.h ファイルの中で宣言される。この関数は、8 × 8 の 2 次元逆離散コサイン変換を実行し、結果をデスティネーション・プレーンに格納する。
2D（8 × 8）逆 DCT（正規化）は、次の式によって定義される。

$$x_{n,m} = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} c_u c_v X_{u,v} \cos\left[\frac{(2n+1)u\pi}{16}\right]\cos\left[\frac{(2m+1)v\pi}{16}\right]$$

$u,v = 0, 1, \ldots, 7$

ここで、n, m = ピクセル領域内の空間座標、

u, v = 変換領域内の座標

$$c_l = \begin{cases} 1/\sqrt{2}, & l = 0 \\ 1, & l \neq 0 \end{cases}$$

出力は、[0, 255] の範囲でクリッピングされる。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsH263BlockStepErr	*step* の値が 8 より小さい場合のエラー状態を示す。

ReconMB_H263, ReconBlock_H263

予測値と逆変換の結果（残差）の和を求めて、INTER マクロブロック / ブロックを再構成する。

```
IppStatus ippiReconMB_H263(const Ipp8u* pSrc,
      const Ipp16s* pSrcResidual, Ipp8u* pDst, int step);
IppStatus ippiReconMB_H263_I(Ipp8u* pSrcDst,
      const Ipp16s* pSrcResidual, int step);
IppStatus ippiReconBlock_H263(const Ipp8u* pSrc,
      const Ipp16s* pSrcResidual, Ipp8u* pDst, int step);
IppStatus ippiReconBlock_H263_I(Ipp8u* pSrcDst,
      const Ipp16s* pSrcResidual, int step);
```

引数

pSrc	リファレンス・プレーン内の入力予測値へのポインタ
pSrcDst	リファレンス・プレーン内の入力予測値とデスティネーション・プレーン内の再構成されたマクロブロック / ブロックへのポインタ（インプレース操作の場合）
pSrcResidual	残差バッファ（64 個の要素）内の逆変換の結果へのポインタ
pDst	デスティネーション・プレーン内の再構成されたマクロブロック / ブロックへのポインタ
step	ソース・プレーンとデスティネーション・プレーンの幅。

説明

関数 ippiReconMB_H263 と ippiReconBlock_H263 は、ippmp.h ファイルの中で宣言される。これらの関数は、予測値と逆変換の結果（残差）の和を求めて、INTER マクロブロック / ブロックを再構成する。エンコーダ・ループおよびデコーダ・ループ内で、変換係数の振幅の量子化歪みによって算術オーバーフローが発生しないように、クリッピング関数が挿入される。クリッピング範囲は [0, 255] である。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsH263MBStepErr	*step* の値が 16 より小さい場合のエラー状態を示す。
ippStsH263BlockStepErr	*step* の値が 8 より小さい場合のエラー状態を示す。

H.263+ 関数

本節では、H263 ＋デコーダのオプション・モードをサポートする関数について説明する。表 16-2 に、H.263 ＋関数の一覧を示す。

表 16-2 H.263 ＋関数

関数の基本名	説明
ExpandFrame_H263	プレーンに合わせてフレームを拡張する。
PredictBlock_OBMC	リファレンス・フレームから現在のブロックを予測する。
FilterDeblocking_HorEdge_H263, FilterDeblocking_VerEdge_H263	再構成されたフレーム上で 1 つのブロック・エッジのブロック解除フィルタリングを実行する。
DecodeMCBPC_Intra_H263 DecodeMCBPC_Inter_H263	MCBPC コードワードをデコードする。
DecodeMODB_H263	MODB コードワードをデコードする。
DecodeCBPY_H263	CBPY コードワードをデコードする。
UpdateQuant_MQ_H263	量子化器の情報を更新する。

ExpandFrame_H263

プレーンに合わせてフレームを拡張する。

```
IppStatus ippiExpandFrame_H263_8u(Ipp8u* pSrcDstPlane, int frameWidth,
     int frameHeight, int expandPels, int step);
```

引数

pSrcDstPlane	プレーンへのポインタ
frameWidth	フレームの幅
frameHeight	フレームの高さ
expandPels	一方向に拡張されるピクセルの数
step	ステップ値、 *step* ≥ planeWidth = *frameWidth* + 2 * *expandPels*

説明

関数 ippiExpandFrame_H263 は、ippmp.h ファイルの中で宣言される。この関数は、ピクチャ境界機能上で動きベクトルをイネーブルにするために、プレーンに合わせてフレームを拡張する。H.263+ の Annex D、F、J をサポートする場合は、この機能をイネーブルにする必要がある。動きベクトルによって参照されるピクセルがコード化されるピクチャ領域の外側にある場合は、代わりにエッジ・ピクセルが使用される。
この関数の呼び出しの前に、ピクチャ・フレームが再構成されている必要がある。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	*pSrcDstPlane* ポインタが NULL の場合のエラー状態を示す。
ippStsH263FrameWidthErr	*frameWidth* の値が 8 より小さい場合のエラー状態を示す。
ippStsH263FrameHeightErr	*frameWidth* の値が 0 または負の場合のエラー状態を示す。

ippStsH263ExpandPelsErr
: *expandPels* の値が 8 より小さい場合のエラー状態を示す。

ippStsH263BlockStepErr
: *step* の値が 8 より小さい場合のエラー状態を示す。

ippStsH263PlaneStepErr
: *step* の値がプレーンの幅より小さい場合のエラー状態を示す。

PredictBlock_OBMC

リファレンス・フレームから現在のブロックを予測する。

```
IppStatus ippiPredictBlock_OBMC_8u(const Ipp8u* pSrcRef, Ipp8u* pDst,
    int step, IppMotionVector* pMVCur, IppMotionVector* pMVLeft,
    IppMotionVector* pMVRight, IppMotionVector* pMVAbove,
    IppMotionVector* pMVBelow);
```

引数

pSrcRef
: 現在のフレーム内の予測されるブロックに空間的に対応する、リファレンス（ソース）プレーン内のブロックへのポインタ

pMVCur
: 現在のブロックへのポインタ

pMVLeft
: 現在のブロックの左側のブロックへのポインタ

pMVRight
: 現在のブロックの右側のブロックへのポインタ

pMVAbove
: 現在のブロックの上側のブロックへのポインタ

pMVBelow
: 現在のブロックの下側のブロックへのポインタ

pDst
: デスティネーション・プレーン内の補正されたブロックへのポインタ

step
: ソース・プレーンとデスティネーション・プレーンの幅。

説明

関数 ippiPredictBlock_OBMC は、ippmp.h ファイルの中で宣言される。この関数は、重複ブロック動き補償（OBMC）を使用して、リファレンス・フレームから現在のブロックを予測する。

動きベクトルによって参照されるサンプルがデコード・ピクチャ領域の外側にある場合は、この関数の呼び出しの前に、この関数に渡される各動きベクトルを調整する必要がある。

周囲のブロックのうち 1 つがコード化されていない場合は、それに対応するリモート動きベクトルへのポインタは、0 動きベクトルに設定される。

周囲のブロックのうち 1 つが INTRA モードでコード化されているか、ピクチャ境界の外側にある場合は、それに対応するリモート動きベクトルへのポインタは、現在の動きベクトルへのポインタ（*pMVCur* に渡されるポインタと同じポインタ）に設定される。

現在のブロックがマクロブロックの最下部にある場合は、現在のマクロブロックの下側のマクロブロック内の 8 × 8 輝度ブロックに対応するリモート動きベクトルは、（上で説明したように）現在のブロックに対応する動きベクトルで置き換えられる。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。

FilterDeblocking_HorEdge_H263, FilterDeblocking_VerEdge_H263

再構成されたフレーム上で 1 つのブロック・エッジのブロック解除フィルタリングを実行する。

```
IppStatus ippiFilterDeblocking_HorEdge_H263_8u_I(Ipp8u* pSrcDst,
      int step, int QP);
IppStatus ippiFilterDeblocking_VerEdge_H263_8u_I(Ipp8u* pSrcDst,
      int step, int QP);
```

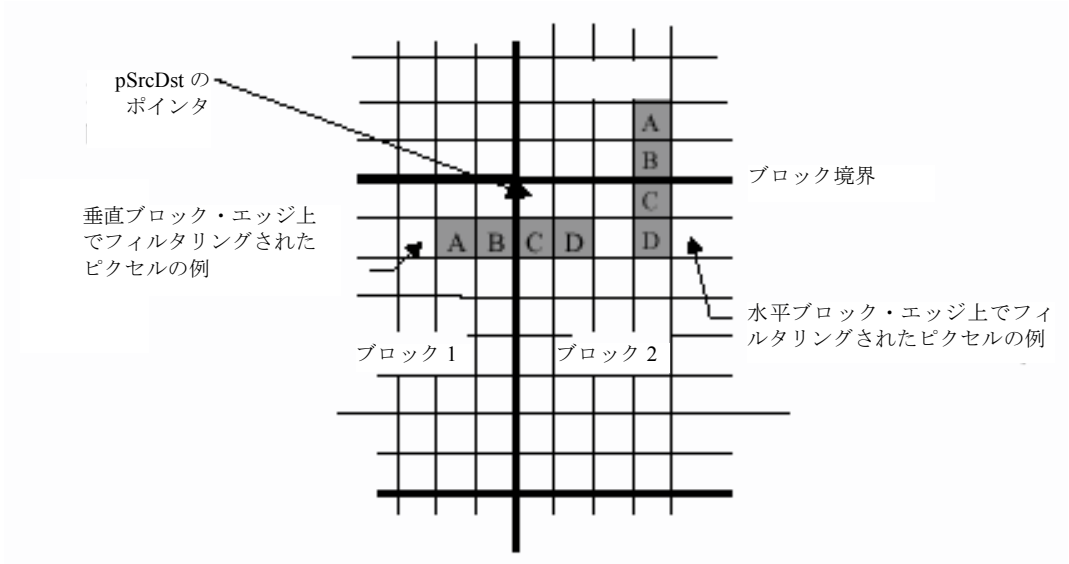
引数

pSrcDst	適用される 2 つのブロックのうち 2 番目のブロック（ブロック 2）の最初のピクセルへのポインタ
step	ソース・プレーンとデスティネーション・プレーンの幅。
QP	量子化パラメータ。*QP* の値については、H.263+ の Annex J の J.3 節の説明に従って検出される。

説明

関数 ippiFilterDeblocking_HorEdge_H263 と ippiFilterDeblocking_VerEdge_H263 は、ippmp.h ファイルの中で宣言される。これらの関数は、再構成されたフレーム上で 1 つのブロック・エッジ（それぞれ水平または垂直）のブロック解除フィルタリングを実行する。図 16-9 に示すように、ポインタ *pSrcDst* は、ブロック 2 の最初のピクセルを指す。

図 16-9　ブロック解除フィルタリングのレイアウト



戻り値

`ippStsNoErr`	エラーがないことを示す。
`ippStsNullPtrErr`	*pSrcDst* ポインタが NULL の場合のエラー状態を示す。
`ippStsH263BlockStepErr`	*step* の値が 8 より小さい場合のエラー状態を示す。
`ippStsH263QuantErr`	量子化器の値が、0 か負の場合、または 31 より大きい場合のエラー状態を示す。

DecodeMCBPC_Intra_H263
DecodeMCBPC_Inter_H263

MCBPC コードワードをデコードする。

```
IppStatus ippiDecodeMCBPC_Intra_H263(Ipp8u** ppBitStream,
      int* pBitOffset, Ipp8u* pDstVal);
IppStatus ippiDecodeMCBPC_Inter_H263(Ipp8u** ppBitStream,
      int* pBitOffset, Ipp8u* pDstVal);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ（**ppBitStream* はデータのデコード後に更新される）。
pBitOffset	**ppBitStream* によって指定されるバイト内のビット位置へのポインタ
pDstVal	デコードされた MCBPC 値へのポインタ

説明

関数 ippiDecodeMCBPC_Intra_H263 と ippiDecodeMCBPC_Inter_H263 は、ippmp.h ファイルの中で宣言される。これらの関数は、それぞれ INTRA マクロブロックと INTER マクロブロックについて、マクロブロック・レイヤ内の MCBPC（Macroblock type and Coded Block Pattern for Chrominance）コードワードをデコードする。コード化されたマクロブロックには、MCBPC が常に含まれている。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsH263MCBPCIntraCodeErr	MCBPC INTRA ストリームの処理中に無効なハフマン・コードが発生した場合のエラー状態を示す。
ippStsH263MCBPCInterCodeErr	MCBPC INTER ストリームの処理中に無効なハフマン・コードが発生した場合のエラー状態を示す。

DecodeMODB_H263

MODB コードワードをデコードする。

```
IppStatus ippDecodeMODB_H263(Ipp8u** ppBitStream, int* pBitOffset,
      Ipp8u* pDstVal);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ（**ppBitStream* はデータのデコード後に更新される）。
pBitOffset	**ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効（**pBitOffset* はブロックのデコード後に更新される）。
pDstVal	デコードされた MODB 値へのポインタ

説明

関数 ippiDecodeMODB_H263 は、ippmp.h ファイルの中で宣言される。この関数は、マクロブロック・レイヤ内の MODB （Macroblock mode for B-blocks）コードワードをデコードする（PB フレーム・モードの場合）。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。

DecodeCBPY_H263

CBPY コードワードをデコードする。

```
IppStatus ippDecodeCBPY_H263(Ipp8u** ppBitStream, int* pBitOffset,
      Ipp8u* pDstVal);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ。**ppBitStream* はデータのデコード後に更新される。
pBitOffset	**ppBitStream* によって指定されるバイト内のビット位置へのポインタ 0 ～ 7 の範囲内で有効。**pBitOffset* はブロックのデコード後に更新される。

pDstVal	デコードされた MODB 値へのポインタ

説明

関数 ippiDecodeMODB_H263 は、ippmp.h ファイルの中で宣言される。この関数は、マクロブロック・レイヤ内の CBPY（Coded Block Pattern for Luminance）コードワードをデコードする。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsH263CBPYCodeErr	CBPY ストリームの処理中に無効なハフマン・コードが発生した場合のエラー状態を示す。

UpdateQuant_MQ_H263

量子化器の情報を更新する。

```
IppStatus ippiUpdateQuant_MQ_H263_1u32s_I(Ipp8u** ppBitStream,
      int* pBitOffset, Ipp32s* pSrcDstQP);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ。
pBitOffset	**ppBitStream* によって指定されるバイト内のビット位置へのポインタ
pSrcDstQP	量子化パラメータの更新された値へのポインタ

説明

関数 ippiUpdateQuant_MQ_H263 は、ippmp.h ファイルの中で宣言される。この関数は、量子化器の情報を含む QUANT コードワードを更新する（修正量子化モードの場合）。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。

H.263+　中レベル関数

本節では、H.263+ デコードに使用される中レベルのインテル IPP 関数について説明する。これらの関数は、通常は複数の低レベル関数を必要とするタスクを実行するように設計されている。通常、中レベル関数の呼び出しは、同等な複数の低レベル関数の呼び出しと比較して、消費する CPU サイクル数が少ない。表 16-3 に、H.263+　中レベル関数の一覧を示す。

表 16-3　H.263 ＋中レベル関数

関数の基本名	説明
DecodeBlockCoef_Intra_H263	INTRA ブロック係数をデコードする。
DecodeBlockCoef_Inter_H263	INTER ブロック係数をデコードする。
DecodeBlockCoef_AdvIntra_H263	高度 INTRA コード化モードで INTRA ブロック係数をデコードする。
DecodeBlockCoef_IntraDCOnly_H263	INTRA ブロックの DC only 係数をデコードする。

DecodeBlockCoef_Intra_H263

INTRA ブロック係数をデコードする。

```
IppStatus ippiDecodeBlockCoef_Intra_H263_1u8u(Ipp8u** ppBitStream,
     int pBitOffset, Ipp8u* pDst, int step, int QP);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ (**ppBitStream* はブロックのデコード後に更新される)。
pBitOffset	**ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。 **pBitOffset* はブロックのデコード後に更新される。

pDst	デスティネーション・プレーン内のブロックへのポインタ
step	デスティネーション・プレーンの幅
QP	量子化パラメータ（非 INTRADC 係数の場合）

説明

関数 ippiDecodeBlockCoef_Intra_H263 は、ippmp.h ファイルの中で宣言される。この関数は、INTRA ブロック係数をデコードする。INTRA ブロック係数に対して、逆方向の量子化、逆方向ジグザグ・ポジショニング （標準形式）、IDCT が実行され、各ステップで適切なクリッピングが行われる。結果は、出力フレーム / プレーンにピクセルベースで格納される。

ビットストリーム・バッファの境界チェックは行われない。

INTRA ブロックの場合、出力値は [0, 255] の範囲でクリッピングされ、デスティネーション・プレーン内の現在のフレームに書き込まれる。

この関数は、現在のブロックの 0 でない AC 係数が、ビットストリーム内に少なくとも 1 つ存在する場合にのみ使用される。

注意： VLC テーブル内で検索できない無効なコードがビットストリーム内で検出された場合は、ステータス・エラーが発生する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsH263BlockStepErr	step の値が 8 より小さい場合のエラー状態を示す。
ippStsH263QuantErr	量子化器の値が、0 か負の場合、または 31 より大きい場合のエラー状態を示す。

DecodeBlockCoef_Inter_H263

INTER ブロック係数をデコードする。

```
IppStatus ippiDecodeBlockCoef_Inter_H263_1u16s(Ipp8u** ppBitStream,
      int* pBitOffset, Ipp16s* pDst, int QP);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ (**ppBitStream* はブロックのデコード後に更新される)。
pBitOffset	**ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。 **pBitOffset* はブロックのデコード後に更新される。
pDst	デコードされた残差バッファ（64 個の short 型整数の連続した配列）へのポインタ
QP	量子化パラメータ

説明

関数 ippiDecodeBlockCoef_Inter_H263 は、ippmp.h ファイルの中で宣言される。この関数は、INTER ブロック係数をデコードする。INTER ブロック係数に対して、逆方向の量子化、逆方向ジグザグ・ポジショニング（標準形式）、DCT が実行され、各ステップで適切なクリッピングが行われる。結果（残差）は、64 個の short 型整数の連続した配列に格納される。

ビットストリーム・バッファの境界チェックは行われない。

INTER ブロックの場合、出力バッファは、さらに再構成を行うための残差を保持する。

動きベクトルが保存されるバッファのサイズ（バイト単位）は、[number_of_macroblocks_per_row ＋ 2]* sizeof （IppMotionVector）に等しくなる。このバッファ内の最初の動きベクトルと最後の動きベクトルは、常に（0,0）である。このバッファ内の n 番目のユニット（最初のユニットと最後のユニットを除く）は、一列に並んだ（n-1）番目のマクロブロックの動きベクトルである。

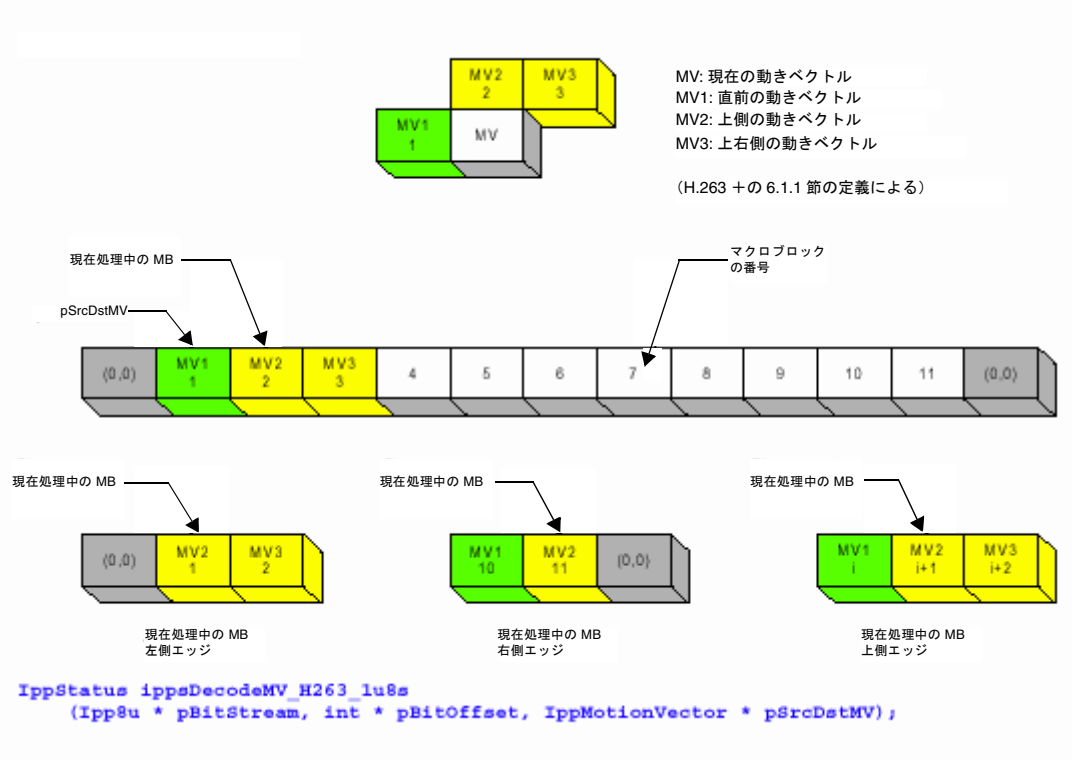
対応するマクロブロックがピクチャの最上部にあるか、（現在のブロックのグループ（GOB）の GOB ヘッダが空白でないときに）ブロックのグループの最上部にある場合は、関数 ippiDecodeMV_TopBorder_H263 が使用される。候補になる予測子は、バッファ内の現在の予測子の左側に格納される。

これらの条件に該当しない場合、H263+ 基本仕様では、ippiDecodeMV_H263 が使用される。3 つの候補になる予測子（MV1、MV2、MV3）は、図 16-10 に示す方法で格納される。

このバッファは、関数が呼び出された後に更新される。候補になる予測子 MV2 は、将来の使用に備えて、デコードされた動きベクトルで置き換えられる。

VLC テーブル内で検索できない無効なコードがビットストリーム内で検出された場合は、ステータス・エラーが発生する。

図 16-10　動きベクトルのデコード




```
IppStatus ippsDecodeMV_H263_1u8s
    (Ipp8u * pBitStream, int * pBitOffset, IppMotionVector * pSrcDstMV);
```

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsH263QuantErr	量子化器の値が、0 か負の場合、または 31 より大きい場合のエラー状態を示す。

DecodeBlockCoef_AdvIntra_H263

高度 INTRA コード化モードで INTRA ブロック係数をデコードする。

```
IppStatus ippiDecodeBlockCoef_AdvIntra_H263_1u8u(Ipp8u** ppBitStream,
     int* pBitOffset, Ipp8u* pDst, int dstStep, Ipp16s*
     pSrcDstCoefRow, Ipp16s* pSrcDstCoefCol, int QP, int predictMode);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ（**ppBitStream* はブロックのデコード後に更新される）。
pBitOffset	**ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。**pBitOffset* はブロックのデコード後に更新される。
pDst	デスティネーション・プレーン内のブロックへのポインタ
pSrcDstCoefRow	現在のブロックの上側のブロックの係数バッファへのポインタ。このバッファは、予測に使用される最初の行を格納し、ブロックのデコード後に更新される。
pSrcDstCoefCol	現在のブロックの左側のブロックの係数バッファへのポインタ。このバッファは、予測に使用される最初の列を格納し、ブロックのデコード後に更新される。
dstStep	デスティネーション・プレーンの幅
QP	量子化パラメータ（非 INTRADC 係数の場合）
predictMode	係数予測モード。次のいずれかの値になる。 IPP_VIDEO_DCONLY, IPP_VIDEO_VERTICAL, IPP_VIDEO_HORIZONTAL

説明

関数 ippiDecodeBlockCoef_AdvIntra_H263 は、ippmp.h ファイルの中で宣言される。
この関数は、高度 INTRA コード化モードで INTRA ブロック係数をデコードする。INTRA ブロック係数に対して、逆方向の量子化、(*predictMode* の値に基づいて、標準、垂直、または水平の）逆方向ジグザグ・ポジショニング、IDCT が実行され、各ステップで適切なクリッピングが行われる。結果は、出力フレーム / プレーンにピクセルベースで格納される（H.263 の付録 I)。

ビットストリーム・バッファの境界チェックは行われない。

INTRA ブロックの場合、出力値は [0, 255] の範囲でクリッピングされ、デスティネーション・プレーン内の現在のフレームに書き込まれる。

この関数は、現在のブロックの 0 でない AC 係数が、ビットストリーム内に少なくとも 1 つ存在する場合にのみ使用される。

VLC テーブル内で検索できない無効なコードがビットストリーム内で検出された場合は、ステータス・エラーが発生する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsH263BlockStepErr	*step* の値が 8 より小さい場合のエラー状態を示す。
ippStsH263QuantErr	量子化器の値が、0 か負の場合、または 31 より大きい場合のエラー状態を示す。
ippStsH263PredModeErr	*predictMode* の値が無効な場合のエラー状態を示す。

DecodeBlockCoef_IntraDCOnly_H263

INTRA ブロックの DC only 係数をデコードする。

```
IppStatus ippiDecodeBlockCoef_IntraDCOnly_H263_1u8u(Ipp8u** ppBitStream,
      int pBitOffset, Ipp8u* pDst, int step);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ （**ppBitStream* はブロックのデコード後に更新される）。
pBitOffset	**ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。 **pBitOffset* はブロックのデコード後に更新される。
pDst	デスティネーション・プレーン内のブロックへのポインタ
step	デスティネーション・プレーンの幅

説明

関数 ippiDecodeBlockCoef_IntraDCOnly_H263 は、ippmp.h ファイルの中で宣言される。この関数は、INTRA ブロックの DC only 係数をデコードする。結果は、出力フレーム / プレーンにピクセルベースで格納される。

ビットストリーム・バッファの境界チェックは行われない。

INTRA ブロックの場合、出力値は [0, 255] の範囲でクリッピングされ、デスティネーション・プレーン内の現在のフレームに書き込まれる。

この関数は、現在のブロックの DC only 係数がビットストリーム内に存在する場合にのみ使用される。

VLC テーブル内で検索できない無効なコードがビットストリーム内で検出された場合は、ステータス・エラーが発生する。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsH263BlockStepErr	step の値が 8 より小さい場合のエラー状態を示す。

MPEG-4 ビデオ・デコーダ 17

本章では、ISO/IEC 14496-2 MPEG-4 ビデオ・デコーダをサポートするために開発されたインテル® IPP 関数について説明する。MPEG-4（[ISO14496] を参照）は、デジタル記憶媒体、インターネット、さまざまな形態の有線 / 無線通信など、各種のアプリケーションのビデオ信号用に広く使用されているコード化手法である。
これらの関数は、MPEG-4 デコーダの以下の要素を対象とする。

- プログレッシブ・ノンスケーラブル・テクスチャ・デコードおよびシェープ（グレースケール）でコード
- マクロブロック・ベースの反復パディングと拡張パディング
- ブロック・ベースの可変長コード （VLC）デコードと逆ジグザグ・スキャン
- 動きベクトルのデコードとパディング
- Intra DC/AC 予測
- オーバーラップ・ブロック動き補償
- ブロック・レイヤ係数デコード。これには、ビットストリーム解析、VLC デコード、Intra DC/AC 予測 （イントラ・ブロックの場合）、逆量子化、逆ジグザグ・スキャン、IDCT （各ステップで適切なクリッピングを実行）が含まれる。
- 動き補償
- デブロック・フィルタやデリンギング・フィルタなど、ノイズ・リダクションのコード化のための後処理

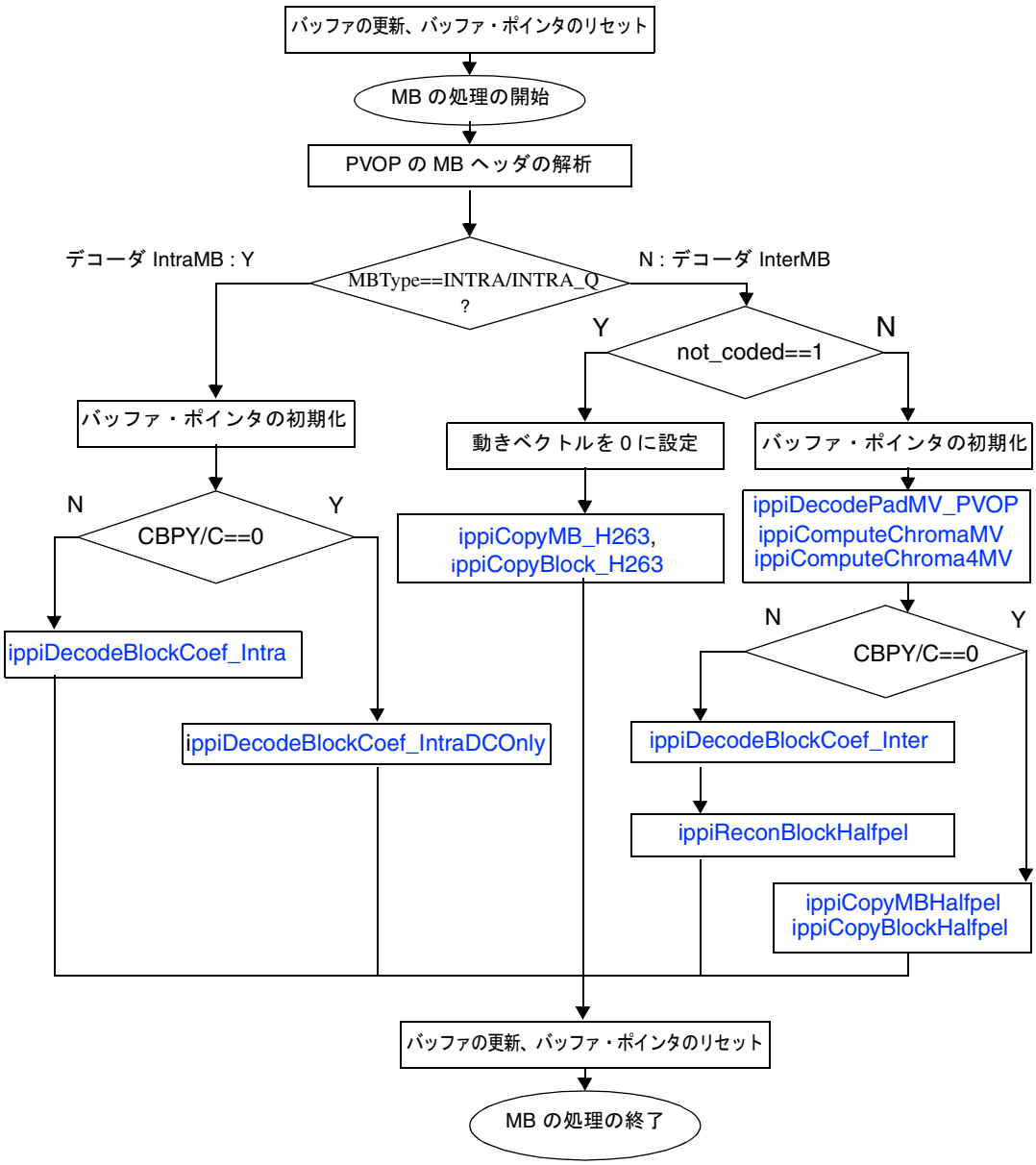
注： すべての MPEG-4 関数は、`short_video_header` の値が 0 の場合に適用可能である。`short_video_header` の値が 1 の場合は、H.263 デコーダ関数を使用すること。

本章では、MPEG-4 関数の使用法を高い視点から把握した説明と、インテル IPP マクロ / データ構造の定義を示した後、個々のインテル IPP 関数について詳しく説明する。

概要

図 17-1 は、P-VOP （Predictive coded Video Object Plane）内のマクロブロック （MB）のデコードに必要な一般的手順と、関連するインテル IPP 関数を示している。

図 17-1　P-VOP 内の MB のデコードのフローチャート



データ・タイプとデータ構造

ビデオ・コンポーネント

ビデオ・コンポーネントは、次のように定義される。

```
typedef enum {
    IPP_VIDEO_LUMINANCE,      /* Luminance component   */
    IPP_VIDEO_CHROMINANCE,    /* Chrominance component */
    IPP_VIDEO_ALPHA            /* Alpha component       */
} IppVideoComponent;
```

ピクセル・プレーンとアルファ・プレーン

デコーダ出力は、Y プレーン （輝度成分）、Cb プレーン、Cr プレーン （クロミナンス成分）と呼ばれる 3 つのピクセル・プレーンと、1 つの A プレーン （アルファ信号）に格納される。

Y プレーンのサイズは、VOP を拡張した結果として、ビデオ・オブジェクト・レイヤ （VOL）のサイズに基づくが、VOL と同じにはならない。輝度 VOP は 4 方向のそれぞれに 16 ピクセルだけ拡張 （およびパディング）されるため、Y プレーンの幅と高さは、VOL の幅と高さよりそれぞれ 32 ピクセル大きくなる。

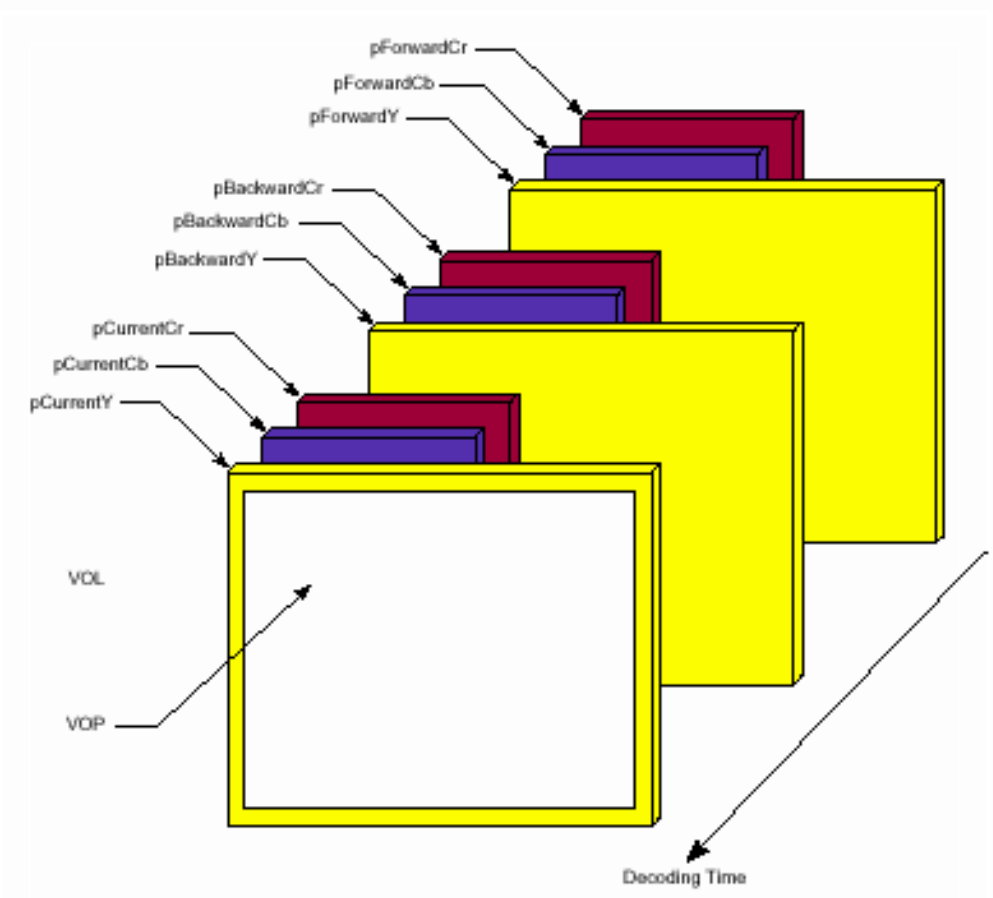
クロミナンス VOP は各方向に 8 ピクセルだけ拡張されるため、Cb または Cr プレーンのサイズ （幅、高さ）は、Y プレーンのサイズの 1/2 になる。

A プレーンは Y プレーンと同じサイズである。

図 17-2 は、ピクセル・プレーン、VOL、VOP の関係を示している。

3 組のピクセル・プレーン （各組が Y プレーン、Cb プレーン、Cr プレーンで構成される）は、ユーザ割り当てによって 32 ビット・ワードにアライメントされていなければならない。これらのピクセル・プレーンは、現在のピクセル・プレーン、順方向のピクセル・プレーン、逆方向のピクセル・プレーンと呼ばれる。MPEG-4 Simple Profile は B-VOP をサポートしていないため、MPEG-4 Simple Profile には逆方向のピクセル・プレーン・セットは不要である。

図 17-2　ピクセル・プレーン、VOL、VOP



マクロブロックのタイプ

Intra コード化（I-）VOP、予測コード化（P-）VOP、双方向予測コード化（B-）VOP 内のマクロブロックのタイプは、次のように定義される。

```
typedef enum {
    IPP_VIDEO_INTER         = 0,   /* P picture or P-VOP */
    IPP_VIDEO_INTER_Q       = 1,   /* P picture or P-VOP */
    IPP_VIDEO_INTER4V       = 2,   /* P picture or P-VOP */
    IPP_VIDEO_INTRA         = 3,  /* I and P picture, or I- and P-VOP */
    IPP_VIDEO_INTRA_Q       = 4,  /* I and P picture, or I- and P-VOP */
    IPP_VIDEO_INTER4V_Q     = 5,   /* P picture or P-VOP (H.263)*/
```

```
    IPP_VIDEO_DIRECT           = 6,  /* B picture or B-VOP (MPEG-4 only) */
    IPP_VIDEO_INTERPOLATE      = 7,   /* B picture or B-VOP */
    IPP_VIDEO_BACKWARD         = 8,   /* B picture or B-VOP */
    IPP_VIDEO_FORWARD          = 9    /* B picture or B-VOP */
} IppMacroblockType;
```

動きベクトル

構造体 IppMotionVector は、MPEG-4 ビデオ処理関数と H.263 ビデオ処理関数の両方に使用できる。この構造体は、次のように定義される。

```
typedef struct {
    Ipp16s dx;
    Ipp16s dy;
} IppMotionVector;
```

1 つのマクロブロック （MB）内には、マクロブロックのタイプに基づいて最大 8 つまでの有効な動きベクトル（MV）が存在する。インテル IPP MPEG-4 コーデックで採用しているベクトル操作方式を以下に示す。pMVForward と pMVBackward は、各マクロブロックに割り当てられる 2 つのベクトル・バッファである。

- P-VOP または B-VOP 内では、1 マクロブロックにつき 2 つのバッファが使用される。これには、pMVForward[4] と pMVBackward[4] が含まれる。
- 要素はブロック・ベースであり、各バッファごとに連続的に格納される。
- P-VOP 内では、pMVForward[] だけが有効なバッファとして使用される。pMVBackward[] は使用されない。
- マクロブロックのタイプが"IPP_VIDEO_INTER"または"IPP_VIDEO_INTER_Q"で、透過でない場合は、pMVForward[0]~[3] には同一のデコードされた MV が格納される。
- マクロブロックが INTRA コード化されているか、スキップされる場合は、pMVForward[0]~[3] は 0 MV でパディングされる。
- B-VOP 内では、pMVForward[] と pMVBackward[] は、使用されることも使用されないこともある。これはマクロブロックのタイプによって異なる。
- B_VOP では、マクロブロックのタイプが "IPP_VIDEO_DIRECT" でない場合は、pMVForward[1]~[3] と pMVBackward[1]~[3] は使用されない。

座標値は、[ISO14496]、図 7-19 に示す絶対座標系に基づいている。

透過ステータス

透過ステータスは、1 バイト（Ipp8u）の 3 ステート値である。3 つの可能なステートは、次のように定義される。

```
enum {
    IPP_VIDEO_TRANSPARENT  = 0,    /* Wholly transparent */
    IPP_VIDEO_PARTIAL      = 1,    /* Partially transparent */
    IPP_VIDEO_OPAQUE       = 2     /* Opaque */
};
```

MPEG-4 では、透過ステータスはブロック・ベースのステータスである。したがって、以下の条件が適用される。

- 1 マクロブロックにつき 1 つのバッファがある。
- 要素はブロック・ベースであり、連続的に格納される。
- 各要素は 1 バイト（Ipp8u）で 1 つのブロックのステータスを表す。
- 1 マクロブロックにつき 4 つの要素がある。
- 最初の要素 （ブロック 0 に対応）は、32 ビットにアライメントされていなければならない（この条件はユーザが保証する必要がある）。
- マクロブロックの透過ステータスは、ワード全体の値を評価することによって決定される。

量子化パラメータ

空間的に右側または下側、あるいはその両方（存在する場合）にあるイントラ・コード化されたマクロブロックの DC および AC 予測を実行するには、イントラ・コード化されたマクロブロックの量子化パラメータ（QP）が格納されている必要がある。量子化パラメータのストレージ・バッファには、以下の条件が適用される。

- I-VOP および P-VOP につき 1 つの行バッファがある。
- このバッファは係数の予測に使用される。
- 「イントラ」または「intra+q」マクロブロックのデコードの前に、このバッファは、上のマクロブロック行の MB の QP を保存する。
- 「イントラ」または「intra+q」マクロブロックのデコード後に、（空間的に上側にあるマクロブロックの）対応する QP は、現在の QP で更新される。
- 各要素は 1 バイト（Ipp8u）で 1 つのマクロブロックを表す。

方向

DC/AC 予測とジグザグ・スキャンを実行する場合は、方向が重要である。以下の列挙子を使用して方向を表す。

```
enum {
    IPP_VIDEO_NONE          = 0,
    IPP_VIDEO_HORIZONTAL    = 1,
    IPP_VIDEO_VERTICAL      = 2
};
```

矩形プレーン

インテル IPP では、矩形を表す以下の構造体が定義されている。

```
    typedef structure _IppiRect {
        int x;
        int y;
        int width;
        int height;
}
```

バッファ

本節では、インテル IPP に必要なバッファとそのレイアウトについて説明する。ユーザは、以下の仕様に従って、バッファの割り当てまたは初期化、あるいはその両方を実行する必要がある。

ビデオ・プレーン・バッファ。シェープ・コード化をサポートしている場合は、テクスチャ成分（Y/Cb/Cr）とアルファ成分（バイナリ / グレースケール）で構成される、デコードされたピクチャ（H.263）またはビデオ・オブジェクト（MPEG-4）を格納するためのバッファを割り当てる必要がある。各成分の有無は、video_object_layer_shape で指定される VOL シェープ・タイプによって異なる。以下の表に、この依存関係を詳しく示す。「x」は必要、「-」は不要を示す。

表 17-1　ビデオ・バッファの割り当ての必要条件

VOL シェープ・タイプ	テクスチャ Y/Cb/Cr プレーン	バイナリ・アルファ・プレーン	グレースケール・アルファ・プレーン
矩形	x	-	-
バイナリ	x	x	-
バイナリのみ	-	x	-
グレースケール	x	x	x

2 組のビデオ・プレーン・バッファを利用できる必要がある。1 組は現在のピクチャまたはビデオ・オブジェクトを格納し、もう 1 組は直前の （順方向参照）ピクチャまたはビデオ・オブジェクトを格納する。双方向予測 （B-VOP）をサポートしている場合は、さらにもう 1 組のバッファを割り当てる必要がある。ビデオ・プレーンの詳細は、本章のピクセル・プレーンとアルファ・プレーンの節を参照のこと。

動きベクトル・バッファ。動きベクトル・バッファは、IppMotionVector データの 4 つの要素を格納する。要素はブロック・ベースであり、各バッファごとに連続的に格納される。P-VOP の場合、1 マクロブロックにつき 1 つの MV バッファを割り当てる必要がある。B-VOP をサポートしている場合は、双方向予測のために 2 つの MV バッファを利用できる必要がある。

マクロブロックのタイプが IPP_VIDEO_INTER または IPP_VIDEO_INTER_Q であり、IPP_VIDEO_INTER4V ではなく、透過でない要素には同一の MV が格納される。

マクロブロックが INTRA コード化されているか、スキップされる場合は、4 つの要素は 0 MV でパディングされる。

B-VOP 内では、順方向または逆方向の MV バッファは、使用されることも使用されないこともある。これはマクロブロックのタイプによって異なる。

B-VOP 内で、マクロブロックのタイプが IPP_VIDEO_DIRECT でない場合は、両方のバッファの要素 [1]~[3] は使用されず、要素 [0] だけが使用される。

透過ステータス・バッファ。1 つのマクロブロックに対して 1 つの透過ステータス・バッファを割り当てる必要がある。Ipp8u 型の要素はブロック・ベースであり、連続的に格納される。1 つのマクロブロックに対して 4 つの要素があり、16 × 16 の領域に対応する。

最初の要素 （ブロック 0 に対応）は、32 ビットにアライメントされていなければならない （この条件はユーザが保証する必要がある）。MB 透過ステータスは、ワード全体の値を評価することによって検出される。

量子化パラメータ・バッファ。P-VOP の量子化パラメータ（QP）を格納する、各要素につき 1 バイト （Ipp8u）を含む 1 つの「行バッファ」を割り当てる必要がある。このバッファは係数の予測に使用される。INTRA コード化されたマクロブロックのデコードの前に、QP バッファ内の対応する要素は、上の MB 行の MB の QP を保存する。マクロブロックのデコード後に、対応する要素 （上側の MB の QP を格納している要素）は、現在のマクロブロックの QP で更新される。

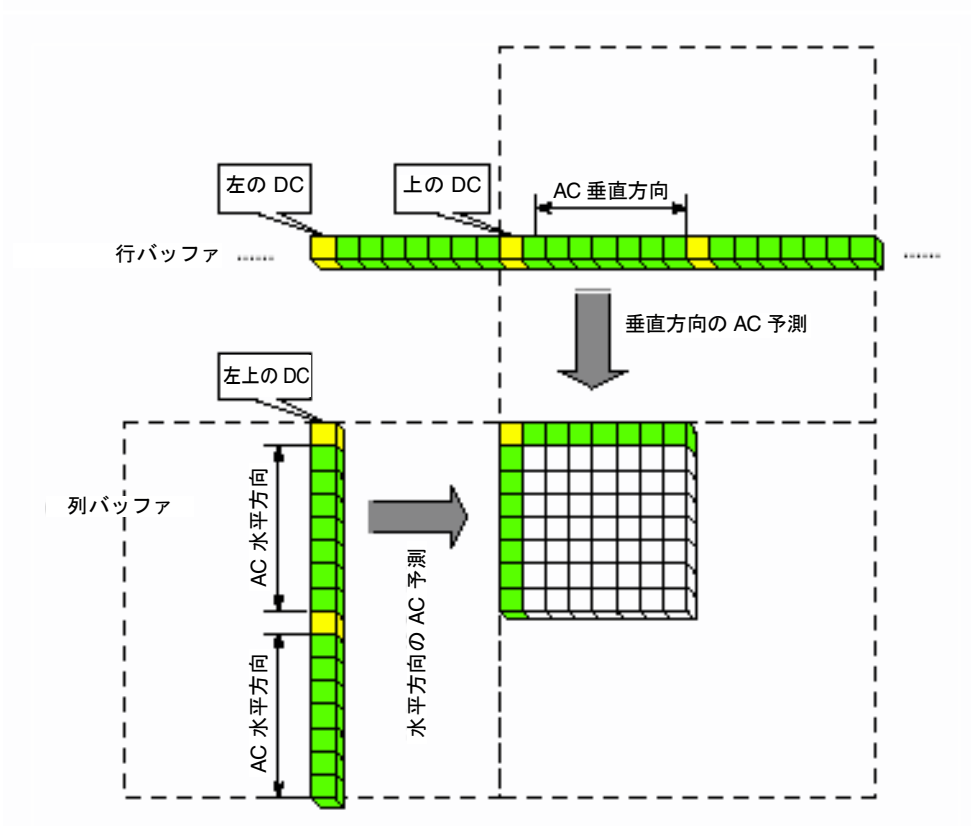
係数バッファ。 Intra DC/AC 予測では、行バッファと列バッファの 2 つの係数バッファを割り当てる必要がある。行バッファは、`Ipp16s` 型の （（`mb_num_per_row` * 2 + 1）* 8）個の要素を格納する。列バッファは、`Ipp16s` 型の 16 個の要素を格納する。

行バッファと列バッファの各 8 要素と、行バッファ内の 8 単位先の 1 つの要素を使用して、マクロブロック内の INTRA コード化されたブロックの DC/AC 予測を実行する。バッファ要素の各グループが、現在デコードされているブロックに対して空間的に上側または左側にある隣接ブロックの係数予測子を格納する。各 8 要素の中で、最初の要素は DC 係数を格納し、その他の要素は量子化された AC 係数を格納する。DC 係数の値が負の場合は、この隣接ブロックが INTRA コード化されていない（したがって、DC 係数も AC 係数も有効でない）ことを示す。

各 VOP をデコードする前に、行バッファ内のすべての DC 要素を -1 に初期化する必要がある。また、各マクロブロック行をデコードする前に、列バッファ内の 2 つの DC 要素も -1 に初期化する必要がある。

図 17-3 に、係数バッファの詳しいレイアウトを示す。

図 17-3　係数バッファのレイアウト図




```
DC Coefficient Prediction:
    QFx[0][0] = PQFx[0][0] + Fp[0][0] //dc_scaler
AC Coefficient Prediction:
    QFx[v][0] = PQFx[v][0] + (QFp[v][0] * QPp)//QPx v = 1 to 7
 or:
    QFx[0][u] = PQFx[0][u] + (QFp[0][u] * QPp)//QPx u = 1 to 7
API:
    IppStatus ippiPredictReconCoefIntra_MPEG4_16s(Ipp16s* pSrcDst,
    Ipp16s* pPredBufRow, Ipp16s* pPredBufCol, int curQP, int predQP,
    int predDir, int ACPredFlag, IppVideoComponent videoComp);
```

MPEG4 ビデオ・デコーダ関数

表 17-2 に、すべての MPEG4 ビデオ・デコーダ関数の一覧を示す。これらの関数については、本節後半で詳しく説明する。

表 17-2　MPEG4 デコーダ関数

関数の基本名	説明
DecodePadMV_PVOP_MPEG4	P-VOP 内の非イントラ・マクロブロックの 4 つの動きベクトルをデコードし、パディングする。
DecodeMV_BVOP_Forward_MPEG4	B-VOP 順方向モードでマクロブロックの動きベクトルをデコードする。
DecodeMV_BVOP_Backward_MPEG4	B-VOP 逆方向モードでマクロブロックの動きベクトルをデコードする。
DecodeMV_BVOP_Interpolate_MPEG4	B-VOP 補間モードでマクロブロックの動きベクトルをデコードする。
DecodeMV_BVOP_Direct_MPEG4	直接モードを使用して、B-VOP 内のマクロブロックの動きベクトルをデコードする。
DecodeMV_BVOP_DirectSkip_MPEG4	現在のマクロブロックがスキップされる場合、直接モードを使用して、B-VOP 内のマクロブロックの動きベクトルをデコードする。
PadMV_MPEG4	透過でないマクロブロックのベクトル・パディングを実行する。
ComputeChromaMV_MPEG4	マクロブロックが各輝度ブロックに対して同じ動きベクトルを持つ場合に、クロミナンス・ブロックの動きベクトルを計算する。
ComputeChroma4MV_MPEG4	マクロブロックが各輝度ブロックに対して個別の動きベクトルを持つ場合に、クロミナンス・ブロックの動きベクトルを計算する。
LimitMVToRect_MPEG4	現在のブロック / マクロブロックの動きベクトルを、拡張された境界矩形内に制限する。
PadCurrent_16x16_MPEG4, PadCurrent_8x8_MPEG4	輝度 / アルファ・マクロブロックまたはクロミナンス・ブロック上で、水平方向および垂直方向の反復パディング・プロセスを実行する。
PadMBHorizontal_MPEG4	境界マクロブロックに隣接する外側のマクロブロック上で、水平方向の拡張パディング・プロセスを実行する。
PadMBVertical_MPEG4	境界マクロブロックに隣接する外側のマクロブロック上で、垂直方向の拡張パディング・プロセスを実行する。
PadMBVertical_MPEG4	境界マクロブロックに隣接しない外側のマクロブロック内のグレー値を充填する。
PadMBGray_MPEG4	境界マクロブロックに隣接しない外側のマクロブロック内のグレー値を充填する。
DecodeVLCZigzag_IntraDCVLC_MPEG4, DecodeVLCZigzag_IntraACVLC_MPEG4	1 つのイントラ・コード化ブロックの VLC デコードと逆ジグザグ・スキャンを実行する。

表 17-2　MPEG4 デコーダ関数（続き）

関数の基本名	説明
DecodeVLC_IntraDCVLC_MPEG4	1 つのイントラ・コード化ブロックの DC 係数の VLC デコードだけを実行する。
DecodeVLCZigzag_Inter_MPEG4	1 つのインター・コード化ブロックの VLC デコードと逆ジグザグ・スキャンを実行する。
PredictReconCoefIntra_MPEG4	イントラ・ブロックの適応型 DC/AC 係数予測を実行する。
QuantInvIntraFirst_MPEG4, QuantInvInterFirst_MPEG4	第 1 の手法によって、イントラ / インター・コード化ブロックの逆量子化を実行する。
QuantInvIntraSecond_MPEG4, QuantInvInterSecond_MPEG4	第 2 の手法によって、イントラ / インター・コード化ブロックの逆量子化を実行する。
QuantInvIntra_MPEG4, QuantInvInter_MPEG4	イントラ / インター・コード化ブロックの逆量子化を実行する。
DecodeBlockCoef_Intra_MPEG4	INTRA ブロック係数をデコードする。
DecodeBlockCoef_IntraDCOnly_MPEG4	DC 係数だけを含む INTRA ブロックをデコードする。
DecodeBlockCoef_Inter_MPEG4	INTER ブロック係数をデコードする。
FilterDeblocking_HorEdge_MPEG4, FilterDeblocking_VerEdge_MPEG4	水平エッジまたは垂直エッジ上でデブロック・フィルタリングを実行する。
FilterDeringingThresholdMB_MPEG4	マクロブロックのデリンギング・フィルタリング用のしきい値を計算する。
FilterDeringingSmoothBlock_MPEG4	ブロックのデリンギング・フィルタリングを実行する。
AverageBlock_MPEG4, AverageMB_MPEG4	2 つのブロック / マクロブロックの平均を求める。
CopyBlockHalfpel_MPEG4, CopyMBHalfpel_MPEG4	ハーフ・ピクセル精度でブロック / マクロブロックをコピーする。
ReconBlockHalfpel_MPEG4	ハーフ・ピクセル精度でブロックを再構成する。
OBMCHalfpel_MPEG4	重複ブロック動き補償を実行する。

DecodePadMV_PVOP_MPEG4

P-VOP 内の非イントラ・マクロブロックの 4 つの動きベクトルをデコードし、パディングする。

```
IppStatus ippiDecodePadMV_PVOP_MPEG4(Ipp8u** ppBitStream,
      int* pBitOffset, IppMotionVector pSrcMVLeftMB[4],
      IppMotionVector pSrcMVUpperMB[4],
      IppMotionVector pSrcMVUpperRightMB[4],
      IppMotionVector pDstMVCurMB[4],
      Ipp8u pTranspLeftMB[4], Ipp8u pTranspUpperMB[4],
      Ipp8u pTranspUpperRightMB[4], Ipp8u pTranspCurMB[4],
      int fcodeForward, IppMacroblockType MBType);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ。**ppBitStream* は動きベクトルのデコード後に更新される。
pBitOffset	*ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。**pBitOffset* は動きベクトルのデコード後に更新される。
pSrcMVLeftMB	現在のマクロブロックに対して空間的に左側にあるマクロブロックの動きベクトル・バッファへのポインタ
pSrcMVUpperMB	現在のマクロブロックに対して空間的に上側にあるマクロブロックの動きベクトル・バッファへのポインタ
pSrcMVUpperRightMB	現在のマクロブロックに対して空間的に右上側にあるマクロブロックの動きベクトル・バッファへのポインタ
pDstMVCurMB	4 つのデコードされた動きベクトルを格納する、現在のマクロブロックのデスティネーション動きベクトル・バッファへのポインタ
pTranspLeftMB	現在のマクロブロックに対して空間的に左側にあるマクロブロックの透過ステータス・バッファへのポインタ

pTranspUpperMB	現在のマクロブロックに対して空間的に上側にあるマクロブロックの透過ステータス・バッファへのポインタ
pTranspUpperRightMB	現在のマクロブロックに対して空間的に右上側にあるマクロブロックの透過ステータス・バッファへのポインタ
pTranspCurMB	現在のマクロブロックの透過ステータス・バッファへのポインタ
fcodeForward	MPEG-4 ビットストリーム構文中の vop_fcode_forward と同じコード
MBType	現在のマクロブロックのタイプ

説明

関数 ippiDecodePadMV_PVOP_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、P-VOP 内の非イントラ・マクロブロックの 4 つの動きベクトルをデコードし、パディングする。

動きベクトルのデコード・プロセスは、[ISO14496]、副項 7.6.3 に規定されている。動きベクトルの予測は、[ISO14496]、副項 7.6.5 に規定されている。動きベクトルのパディング・プロセスは、[ISO14496]、副項 7.6.1.6 に規定されている。

対応するマクロブロックが VOP の外側にある場合は、*pTranspLeftMB*、*pTranspUpperMB*、*pTranspUpperRightMB* は、IPP_VIDEO_TRANSPARENT に設定する必要がある。

MBType が IPP_VIDEO_INTER4V でない場合でも、デスティネーション動きベクトル・バッファには、同一のデコードされたベクトルが格納される。

注： 透過ステータス・バッファへのポインタは、32 ビットにアライメントされていなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *ppBitStream*、**ppBitStream*、*pBitOffset*、*pTranspLeftMB*、*pTranspUpperMB*、*pTranspUpperRightMB*、*pTranspCurMB*、*pDstMVCurMB* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4BitOffsetErr	**pBitOffset* が [0, 7] の範囲から外れている場合のエラー状態を示す。
ippStsMP4FcodeErr	*fcodeForward* が [1, 7] の範囲から外れている場合のエラー状態を示す。
ippStsMP4AlignErr	ポインタ *pTranspLeftMB*、*pTranspUpperMB*、*pTranspUpperRightMB*、pTranspCurMB のうち少なくとも 1 つが 32 ビットにアライメントされていない場合のエラー状態を示す。
ippStsMP4MVCodeErr	MV ストリームの処理中に無効なハフマン・コードが検出された場合のエラー状態を示す。

DecodeMV_BVOP_Forward_MPEG4

B-VOP 順方向モードでマクロブロックの動きベクトルをデコードする。

```
IppStatus ippiDecodeMV_BVOP_Forward_MPEG4(Ipp8u** ppBitStream,
      int* pBitOffset, IppMotionVector pSrcDstMVF[4],
      int fcodeForward);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ。**ppBitStream* は動きベクトルのデコード後に更新される。
pBitOffset	*ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。**pBitOffset* は動きベクトルのデコード後に更新される。

pSrcDstMVF	入力の順方向動きベクトル予測子へのポインタ。出力の現在のマクロブロックの順方向動きベクトルへのポインタ
fcodeForward	MPEG-4 ビットストリーム構文中の *vop_fcode_forward* と同じコード

説明

関数 ppiDecodeMV_BVOP_Forward_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、B-VOP 順方向モードでマクロブロックの動きベクトルをデコードする。動きベクトルのデコード・プロセスは、[ISO14496]、副項 7.6.8 に規定されている。

順方向モードのマクロブロックのデコード後は、順方向予測子だけが、デコードされた順方向ベクトルに設定される。順方向動きベクトル予測子は、各マクロブロック行の始まりで 0 にリセットする必要がある。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *ppBitStream*、**ppBitStream*、*pBitOffset*、*pSrcDstMVF* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4BitOffsetErr	**pBitOffset* が [0, 7] の範囲から外れている場合のエラー状態を示す。
ippStsMP4FcodeErr	*fcodeForward* が [1,7] の範囲を外れている場合のエラー状態を示す。
ippStsMP4MVCodeErr	MV ストリームの処理中に無効なハフマン・コードが検出された場合のエラー状態を示す。

DecodeMV_BVOP_Backward_MPEG4

B-VOP 逆方向モードでマクロブロックの動きベクトルをデコードする。

```
IppStatus ippiDecodeMV_BVOP_Backward_MPEG4(Ipp8u** ppBitStream,
      int* pBitOffset, IppMotionVector pSrcDstMVB[4],
      int fcodeBackward);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ。**ppBitStream* は動きベクトルのデコード後に更新される。
pBitOffset	*ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。**pBitOffset* は動きベクトルのデコード後に更新される。
pSrcDstMVB	入力の逆方向動きベクトル予測子へのポインタ。出力の現在のマクロブロックの逆方向動きベクトルへのポインタ
fcodeBackward	MPEG-4 ビットストリーム構文中の vop_fcode_backward と同じコード

説明

関数 ippiDecodeMV_BVOP_Backward_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、B-VOP 逆方向モードでマクロブロックの動きベクトルをデコードする。動きベクトルのデコード・プロセスは、[ISO14496]、副項 7.6.8 に規定されている。

逆方向モードのマクロブロックのデコード後は、逆方向予測子だけが、デコードされた逆方向ベクトルに設定される。逆方向動きベクトル予測子は、各マクロブロック行の始まりで 0 にリセットする必要がある。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *ppBitStream*、**ppBitStream*、*pBitOffset*、*pSrcDstMVB* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4BitOffsetErr	**pBitOffset* が [0, 7] の範囲から外れている場合のエラー状態を示す。
ippStsMP4FcodeErr	*FcodeBackward* が [1, 7] の範囲から外れている場合のエラー状態を示す。
ippStsMP4MVCodeErr	MV ストリームの処理中に無効なハフマン・コードが検出された場合のエラー状態を示す。

DecodeMV_BVOP_Interpolate_MPEG4

B-VOP 補間モードでマクロブロックの動きベクトルをデコードする。

```
IppStatus ippiDecodeMV_BVOP_Interpolate_MPEG4(
      Ipp8u** ppBitStream, int* pBitOffset, IppMotionVector
      pSrcDstMVF[4], IppMotionVector pSrcDstMVB[4], int fcodeForward,
      int fcodeBackward);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ。**ppBitStream* は動きベクトルのデコード後に更新される。
pBitOffset	*ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。**pBitOffset* は動きベクトルのデコード後に更新される。
pSrcDstMVF	入力の順方向動きベクトル予測子へのポインタ。出力の現在のマクロブロックの順方向動きベクトルへのポインタ
pSrcDstMVB	入力の逆方向動きベクトル予測子へのポインタ。出力の現在のマクロブロックの逆方向動きベクトルへのポインタ
fcodeForward	MPEG-4 ビットストリーム構文中の vop_fcode_forward と同じコード
fcodeBackward	MPEG-4 ビットストリーム構文中の vop_fcode_backward と同じコード

説明

関数 ippiDecodeMV_BVOP_Interpolate_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、B-VOP 補間モードでマクロブロックの動きベクトルをデコードする。動きベクトルのデコード・プロセスは、[ISO14496]、副項 7.6.8 に規定されている。

順方向および逆方向の動きベクトル予測子は、各マクロブロック行の始まりで 0 にリセットする必要がある。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *ppBitStream*、**ppBitStream*、*pBitOffset*、*pSrcDstMVF*、*pSrcDstMVB* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4BitOffsetErr	**pBitOffset* が [0, 7] の範囲から外れている場合のエラー状態を示す。
ippStsMP4FcodeErr	*fcodeForward* または *fcodeBackward* が [1, 7] の範囲から外れている場合のエラー状態を示す。
ippStsMP4MVCodeErr	MV ストリームの処理中に無効なハフマン・コードが検出された場合のエラー状態を示す。

DecodeMV_BVOP_Direct_MPEG4

直接モードを使用して、B-VOP 内のマクロブロックの動きベクトルをデコードする。

```
IppStatus ippiDecodeMV_BVOP_Direct_MPEG4(Ipp8u** ppBitStream,
      int* pBitOffset, const IppMotionVector pSrcMV[4],
      IppMotionVector pDstMVF[4], IppMotionVector pDstMVB[4],
      Ipp8u pTranspSrcMB[4], int TRB, int TRD);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ。**ppBitStream* は動きベクトルのデコード後に更新される。
pBitOffset	*ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。**pBitOffset* は動きベクトルのデコード後に更新される。
pSrcMV	最近デコードされた I-VOP または P-VOP 内の同じ位置にあるマクロブロックの動きベクトル・バッファへのポインタ
pDstMVF	現在のマクロブロックの順方向動きベクトル・バッファへのポインタ

pDstMVB	現在のマクロブロックの逆方向動きベクトル・バッファへのポインタ
pTranspSrcMB	同じ位置にあるマクロブロックの透過ステータス・バッファへのポインタ
TRB	B-VOP と直前のリファレンス VOP の時間的リファレンスの差
TRD	時間的に次のリファレンス VOP と時間的に直前のリファレンス VOP の時間的リファレンスの差

説明

関数 ippiDecodeMV_BVOP_Direct_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、直接モードを使用して、B-VOP 内のマクロブロックの動きベクトルをデコードする。ソース動きベクトルのデコード・プロセスは、[ISO14496]、副項 7.6.8 に規定されている。デスティネーション動きベクトルの計算プロセスは、[ISO14496]、副項に規定されている。

注： 透過ステータス・バッファへのポインタは、32 ビットにアライメントされていなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *ppBitStream*、**ppBitStream*、*pBitOffset*、*pSrcMV*、*pDstMVF*、*pDstMVB*、*pTranspSrcMB* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4BitOffsetErr	**pBitOffset* が [0, 7] の範囲から外れている場合のエラー状態を示す。
ippStsMP4TempDiffErr	*TRB* または *TRD* の値が 0 または負の場合のエラー状態を示す。
ippStsMP4MVCodeErr	MV ストリームの処理中に無効なハフマン・コードが検出された場合のエラー状態を示す。

DecodeMV_BVOP_DirectSkip_MPEG4

現在のマクロブロックがスキップされる場合、直接モードを使用して、B-VOP 内のマクロブロックの動きベクトルをデコードする。

```
IppStatus ippiDecodeMV_BVOP_DirectSkip_MPEG4(
      const IppMotionVector pSrcMV[4], IppMotionVector pDstMVF[4],
      IppMotionVector pDstMVB[4], Ipp8u pTranspSrcMB[4], int TRB,
      int TRD);
```

引数

pSrcMV	最近デコードされた I-VOP または P-VOP 内の同じ位置にあるマクロブロックの動きベクトル・バッファへのポインタ
pTranspSrcMB	同じ位置にあるマクロブロックの透過ステータス・バッファへのポインタ
pDstMVF	現在のマクロブロックの順方向動きベクトル・バッファへのポインタ
pDstMVB	現在のマクロブロックの逆方向動きベクトル・バッファへのポインタ
TRB	B-VOP と直前のリファレンス VOP の時間的リファレンスの差 .
TRD	時間的に次のリファレンス VOP と時間的に直前のリファレンス VOP の時間的リファレンスの差。中間に B-VOP またはスキップされた VOP があるものとする。

説明

関数 ippiDecodeMV_BVOP_DirectSkip_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、現在のマクロブロックがスキップされる場合 （つまり、modb == "1")、直接モードを使用して、B-VOP 内のマクロブロックの動きベクトルをデコードする。デスティネーション動きベクトルの計算プロセスは、[ISO14496]、副項 7.6.9.5.2 に規定されている。マクロブロックがスキップされるため、*MVDx* と *MVDy* は 0 に設定される。

注： 透過ステータス・バッファへのポインタは、32 ビットにアライメントされていなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *pSrcMV*、*pDstMVF*、*pDstMVB*、*pTranspSrcMB* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4TempDiffErr	*TRB* または *TRD* の値が 0 または負の場合のエラー状態を示す。

PadMV_MPEG4

透過でないマクロブロックのベクトル・パディングを実行する。

```
ppStatus ippiPadMV_MPEG4(IppMotionVector pSrcDstMV[4],
      Ipp8u pTransp[4]);
```

引数

pSrcDstMV	パディングされる現在のマクロブロックの動きベクトル・バッファへのポインタ
pTransp	現在のマクロロックの透過ステータス・バッファへのポインタ

説明

関数 ippiPadMV_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、透過でないマクロブロックのベクトル・パディングを実行する。動きベクトルのパディング・プロセスは、[ISO14496]、副項 7.6.1.6 に規定されている。

注： 透過ステータス・バッファへのポインタは、32 ビットにアライメントされていなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つが NULL の場合のエラー状態を示す。

ComputeChromaMV_MPEG4

マクロブロックが各輝度ブロックに対して同じ動きベクトルを持つ場合に、クロミナンス・ブロックの動きベクトルを計算する。

```
IppStatus ippiComputeChromaMV_MPEG4(const IppMotionVector* pLumaMV,
      IppMotionVector* pChromaMV);
```

引数

pLumaMV	輝度ブロックの動きベクトルへのポインタ
pChromaMV	クロミナンス・ブロックの動きベクトルへのポインタ

説明

関数 ippiComputeChromaMV_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、マクロブロックが各輝度（Y）ブロックに対して動きベクトルを 1 つだけ持つ場合に、両方のクロミナンス・ブロック（Cb、Cr）の動きベクトルを計算する。ただし、クロミナンス・ブロックの動きベクトルはデコードされず、（[ISO14496]、副項 7.6.5、表 7-9 の定義に従って）輝度ブロックの動きベクトルから計算される。この関数は、P-VOP および B-VOP フレームのデコードに使用される。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。

ComputeChroma4MV_MPEG4

マクロブロックが各輝度ブロックに対して個別の動きベクトルを持つ場合に、クロミナンス・ブロックの動きベクトルを計算する。

```
IppStatus ippiComputeChroma4MV_MPEG4 (const IppMotionVector
     pLumaMV[4], IppMotionVector* pChromaMV, Ipp8u pTranspMB[4]);
```

引数

pLumaMV	現在のマクロブロック内の輝度ブロックの動きベクトルへのポインタの配列
pChromaMV	クロミナンス・ブロックの動きベクトルへのポインタ
pTranspMB	現在のマクロブロック内の各輝度ブロックの透過ステータスへのポインタの配列

説明

関数 ippiComputeChroma4MV_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、マクロブロックが各輝度（Y）ブロックに対して個別の動きベクトルを持つ場合に、両方のクロミナンス・ブロック（Cb、Cr）の動きベクトルを計算する。ただし、クロミナンス・ブロックの動きベクトルはデコードされず、（[ISO14496]、副項7.6.5、表7-6～7-9の定義に従って）輝度ブロックの動きベクトルから計算される。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。

LimitMVToRect_MPEG4

現在のブロック / マクロブロックの動きベクトルを、拡張された境界矩形内に制限する。

```
IppStatus ippiLimitMVToRect_MPEG4(const IppMotionVector* pSrcMV,
      IppMotionVector* pDstMV, IppiRect* pRectVOPRef, int xcoord,
      int ycoord, int size);
```

引数

pSrcMV	現在のブロックまたはマクロブロックの動きベクトルへのポインタ
pRectVOPRef	境界矩形へのポインタ
pDstMV	制限される動きベクトルへのポインタ
xcord, ycoord	現在のブロックまたはマクロブロックの左上座標
size	ブロック（8）またはマクロブロック（16）のサイズ

説明

関数 ippiLimitMVToRect_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、[ISO14496]、副項 7.6.4 の規定に従って、現在のブロック / マクロブロックの動きベクトルを、拡張された境界矩形内に制限する。

size の値は、ブロックの場合は 8、マクロブロックの場合は 16 である。境界矩形の幅と高さは、*size* の値の 2 倍より大きくなければならない。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つが NULL の場合のエラー状態を示す。
ippStsMP4BlockSizeErr	*size* の値が 8 でない場合（ブロック）または 16 でない場合 （マクロブロック）のエラー状態を示す。
ippStsSizeErr	*pRectVOPRef* のサイズが不適当な場合 （すなわち、矩形の幅または高さの値が 0 または負であるか、2**size* より小さい場合）のエラー状態を示す。

PadCurrent_16x16_MPEG4, PadCurrent_8x8_MPEG4

輝度 / アルファ・マクロブロックまたはクロミナンス・ブロック上で、水平方向および垂直方向の反復パディング・プロセスを実行する。

```
IppStatus ippiPadCurrent_16x16_MPEG4_8u_I(Ipp8u* pSrcDst,int step,
      const Ipp8u* pBAB);
IppStatus ippiPadCurrent_8x8_MPEG4_8u_I(Ipp8u* pSrcDst,int step,
      const Ipp8u* pBAB);
```

引数

pSrcDst	パディングされるマクロブロック / ブロックへのポインタ
step	ソース・プレーンの幅（バイト単位）
pBAB	バイナリ・アルファ・ブロック・バッファ（BAB）へのポインタ。このバッファは 256 バイト（1 つのピクセルのプロパティに 1 バイト）を格納する。

説明

関数 ippiPadCurrent_16x16_MPEG4 と ippiPadCurrent_8x8_MPEG4 は、ippmp.h ファイルの中で宣言される。これらの関数は、それぞれ、16 × 16 輝度 / アルファ・マクロブロックまたは 8 × 8 クロミナンス・ブロック上で、水平方向および垂直方向の反復パディング・プロセスを実行する。水平方向および垂直方向の反復パディング・プロセスは、[ISO14496]、副項 7.6.1.1 および 7.6.1.2 に規定されている。クロミナンス成分については、対応する輝度成分のシェープ・ブロックをサブサンプリングして、BAB を生成する必要がある。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つが NULL の場合のエラー状態を示す。
ippStsStepErr	*step* が 16（16 × 16 バージョン）または 8（8 × 8 バージョン）より小さいか、4 で割り切れない場合のエラー状態を示す。

ippStsMP4ZeroBABErr	すべての BAB 要素の値が 0 の場合のエラー状態を示す。

PadMBHorizontal_MPEG4

境界マクロブロックに隣接する外側のマクロブロック上で、水平方向の拡張パディング・プロセスを実行する。

```
IppStatus ippiPadMBHorizontal_MPEG4_8u(const Ipp8u* pSrcY,
      const Ipp8u* pSrcCb, const Ipp8u* pSrcCr, const Ipp8u* pSrcA,
      Ipp8u* pDstY, Ipp8u* pDstCb, Ipp8u* pDstCr, Ipp8u* pDstA,
      int stepYA, int stepCbCr);
```

引数

pSrcY	外側のマクロブロックのパディングのために選ばれた、境界輝度ブロックの垂直ボーダのうち 1 つへのポインタ
pSrcCb	外側のマクロブロックのパディングのために選ばれた、境界 Cb ブロックの垂直ボーダのうち 1 つへのポインタ
pSrcCr	外側のマクロブロックのパディングのために選ばれた、境界 Cr ブロックの垂直ボーダのうち 1 つへのポインタ
pSrcA	外側のマクロブロックのパディングのために選ばれた、境界アルファ・ブロックの垂直ボーダのうち 1 つへのポインタ
pDstY	パディングされた外側の輝度ブロックへのポインタ
pDstCb	パディングされた外側の Cb ブロックへのポインタ
pDstCr	パディングされた外側の Cr ブロックへのポインタ
pDstA	パディングされた外側のアルファ・ブロックへのポインタ
stepYA	輝度プレーンまたはアルファ・プレーン、あるいはその両方の幅（バイト単位）
stepCbCr	クロミナンス・プレーンの幅（バイト単位）

説明

関数 ippiPadMBHorizontal_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、[ISO14496]、副項 7.6.1.3 の規定に従って、境界マクロブロックに隣接する外側のマクロブロック上で、水平方向の拡張パディング・プロセスを実行する。

pSrcA の値が 0 の場合は、アルファ・プレーンは使用できない。それ以外の場合は、アルファ・プレーンをパディングする必要がある。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *pSrcY*、*pSrcCb*、*pSrcCr*、*pDstY*、*pDstCb*、*pDstCr* のうち 1 つが NULL の場合、または *pDstA* が NULL で *pSrcA* が NULL でない場合のエラー状態を示す。
ippStsStepErr	*stepYA* が 16 より小さいか、*stepCbCr* が 8 より小さいか、これらの引数のうち少なくとも 1 つが 4 で割り切れない場合のエラー状態を示す。

PadMBVertical_MPEG4

境界マクロブロックに隣接する外側のマクロブロック上で、垂直方向の拡張パディング・プロセスを実行する。

```
IppStatus ippiPadMBVertical_MPEG4_8u(const Ipp8u* pSrcY,
      const Ipp8u* pSrcCb, const Ipp8u* pSrcCr, const Ipp8u* pSrcA,
      Ipp8u* pDstY, Ipp8u* pDstCb, Ipp8u* pDstCr, Ipp8u* pDstA,
      int stepYA, int stepCbCr);
```

引数

pSrcY	外側のマクロブロックのパディングのために選ばれた、境界輝度ブロックの水平ボーダのうち 1 つへのポインタ
pSrcCb	外側のマクロブロックのパディングのために選ばれた、境界 Cb ブロックの水平ボーダのうち 1 つへのポインタ
pSrcCr	外側のマクロブロックのパディングのために選ばれた、境界 Cr ブロックの水平ボーダのうち 1 つへのポインタ

pSrcA	外側のマクロブロックのパディングのために選ばれた、境界アルファ・ブロックの水平ボーダのうち 1 つへのポインタ
pDstY	パディングされた外側の輝度ブロックへのポインタ
pDstCb	パディングされた外側の Cb ブロックへのポインタ
pDstCr	パディングされた外側の Cr ブロックへのポインタ
pDstA	パディングされた外側のアルファ・ブロックへのポインタ
stepYA	輝度プレーンまたはアルファ・プレーン、あるいはその両方の幅（バイト単位）
stepCbCr	クロミナンス・プレーンの幅（バイト単位）

説明

関数 ippiPadMBVertical_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、[ISO14496]、副項 7.6.1.3 の規定に従って、境界マクロブロックに隣接する外側のマクロブロック上で、垂直方向の拡張パディング・プロセスを実行する。

pSrcA の値が 0 の場合は、アルファ・プレーンは使用できない。それ以外の場合は、アルファ・プレーンをパディングする必要がある。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *pSrcY*、*pSrcCb*、*pSrcCr*、*pDstY*、*pDstCb*、*pDstCr* のうち 1 つが NULL の場合、または *pDstA* が NULL で *pSrcA* が NULL でない場合のエラー状態を示す。
ippStsStepErr	*stepYA* が 16 より小さいか、*stepCbCr* が 8 より小さいか、これらの引数のうち少なくとも 1 つが 4 で割り切れない場合のエラー状態を示す。

PadMBGray_MPEG4

境界マクロブロックに隣接しない外側のマクロブロック内のグレー値を充填する。

```
IppStatus ippiPadMBGray_MPEG4_8u(Ipp8u grayVal, Ipp8u* pDstY,
      Ipp8u* pDstCb, Ipp8u* pDstCr, Ipp8u* pDstA, int stepYA,
      int stepCbCr);
```

引数

grayVal	外側のマクロブロック / ブロックを充填するグレー値
pDstY	パディングされた外側の輝度ブロックへのポインタ
pDstCb	パディングされた外側の Cb ブロックへのポインタ
pDstCr	パディングされた外側の Cr ブロックへのポインタ
pDstA	パディングされた外側のアルファ・ブロックへのポインタ
stepYA	輝度プレーンまたはアルファ・プレーン、あるいはその両方の幅（バイト単位）
stepCbCr	クロミナンス・プレーンの幅（バイト単位）

説明

関数 ippiPadMBGray_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、境界マクロブロックに隣接しない外側のマクロブロック内のグレー値を充填する。

pDstA の値が 0 の場合は、アルファ・プレーンは使用できない。それ以外の場合は、アルファ・プレーンをパディングする必要がある。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *pDstY*、*pDstCb*、*pDstCr* のうち 1 つが NULL の場合のエラー状態を示す。
ippStsStepErr	*stepYA* が 16 より小さいか、*stepCbCr* が 8 より小さいか、これらの引数のうち少なくとも 1 つが 4 で割り切れない場合のエラー状態を示す。

DecodeVLCZigzag_IntraDCVLC_MPEG4, DecodeVLCZigzag_IntraACVLC_MPEG4

1 つのイントラ・コード化ブロックの VLC デコードと逆ジグザグ・スキャンを実行する。

```
IppStatus ippiDecodeVLCZigzag_IntraDCVLC_MPEG4_1u16s(Ipp8u**
     ppBitStream, int* pBitOffset, Ipp16s* pDst, int predDir,
     IppVideoComponent videoComp);
IppStatus ippiDecodeVLCZigzag_IntraACVLC_MPEG4_1u16s(Ipp8u**
     ppBitStream, int* pBitOffset, Ipp16s* pDst, int predDir);
```

引数

ppBitStream ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ。**ppBitStream* はブロックのデコード後に更新される。

pBitOffset *ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。*pBitOffset* はブロックのデコード後に更新される。

pDst 現在のブロックの量子化された係数の残差（PQF）へのポインタ

predDir ジグザグ・スキャン・パターンの決定に使用される AC 予測の方向。次のいずれかの値になる。

`IPP_VIDEO_NONE` AC 予測を使用せず、標準形式のジグザグ・スキャンを実行する。

`IPP_VIDEO_HORIZONTAL` 水平方向の予測。垂直方向の交互ジグザグ・スキャンを実行する。

`IPP_VIDEO_VERTICAL` 垂直方向の予測。水平方向の交互ジグザグ・スキャンを実行する。

videoComp 現在のブロックのビデオ・コンポーネントのタイプ（輝度、クロミナンス、またはアルファ）。この値は DC 係数のデコードに使用される。

説明

関数 ippiDecodeVLCZigzag_IntraDCVLC_MPEG4 と ippiDecodeVLCZigzag_IntraACVLC_MPEG4 は、ippmp.h ファイルの中で宣言される。これらの関数は、1 つのイントラ・コード化ブロックの VLC デコードと逆ジグザグ・スキャンを実行する。Intra DC の VLC デコード・プロセスは、[ISO14496]、副項 7.4.1.1 に規定されている。Intra AC の VLC デコード・プロセスは、[ISO14496]、副項 7.4.1.2、7.4.1.3 に規定されている。逆ジグザグ・スキャンは、[ISO14496]、副項 7.4.2 に規定されている。

関数の IntraDCVLC バージョンは、Intra DC の VLC を使用して Intra DC 係数をデコードする。IntraACVLC バージョンは、Intra AC の VLC を使用して Intra DC 係数をデコードする。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *ppBitStream*、**ppBitStream*、*pBitOffset*、*pDst* のうち 1 つが NULL の場合のエラー状態を示す。
ippStsMP4BitOffsetErr	**pBitOffset* が [0, 7] の範囲から外れている場合のエラー状態を示す。
ippStsMP4DCCodeErr	（DC バージョンのみ）DC ストリームの処理中に無効なコードが検出された場合のエラー状態を示す。
ippStsMP4VLCCodeErr	VLC ストリームの処理中に無効な Huffman コードが検出された場合のエラー状態を示す。
ippStsMPEG4PredDirErr	*predDir* が有効でない場合のエラー状態を示す。

DecodeVLC_IntraDCVLC_MPEG4

1 つのイントラ・コード化ブロックの DC 係数の VLC デコードだけを実行する。

```
IppStatus ippiDecodeVLC_IntraDCVLC_MPEG4_1u16s(Ipp8u ** ppBitStream,
      int* pBitOffset, Ipp16s* pDst, IppVideoComponent videoComp);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ。**ppBitStream* はブロックのデコード後に更新される。
pBitOffset	*ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。*pBitOffset* はブロックのデコード後に更新される。
pDst	デコードされた DC 係数へのポインタ
videoComp	現在のブロックのビデオ・コンポーネントのタイプ（輝度、クロミナンス、またはアルファ）。この値は DC 係数のデコードに使用される。

説明

関数 ippiDecodeVLC_IntraDCVLC_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、1 つのイントラ・コード化ブロックの DC 係数の VLC デコードだけを実行する。Intra DC の VLC デコード・プロセスは、[ISO14496]、副項 7.4.1.1 に規定されている。

関数 ippiDecodeVLC_IntraDCVLC_MPEG4 は、データ・パーティション・モード（フラグ data_partitioned がセットされている場合）に必要である。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4BitOffsetErr	**pBitOffset* が [0, 7] の範囲から外れている場合のエラー状態を示す。
ippStsMP4DCCodeErr	DC ストリームの処理中に無効なコードが検出された場合のエラー状態を示す。

DecodeVLCZigzag_Inter_MPEG4

1 つのインター・コード化ブロックの VLC デコードと逆ジグザグ・スキャンを実行する。

```
IppStatus ippiDecodeVLCZigzag_Inter_MPEG4_1u16s(Ipp8u** ppBitStream,
     int* pBitOffset, Ipp16s* pDst);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ。**ppBitStream* はブロックのデコード後に更新される。
pBitOffset	*ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。*pBitOffset* はブロックのデコード後に更新される。
pDst	現在のブロックの量子化された係数の残差（PQF）へのポインタ

説明

関数 ippiDecodeVLCZigzag_Inter_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、1 つのインター・コード化ブロックの VLC デコードを実行する。Inter VLC デコード・プロセスは、[ISO14496]、副項 7.4.1.2、7.4.1.3 に規定されている。逆ジグザグ・スキャンは、[ISO14496]、副項 7.4.2 に規定されている。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *ppBitStream*、**ppBitStream*、*pBitOffset*、*pDst* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4BitOffsetErr	**pBitOffset* が [0, 7] の範囲から外れている場合のエラー状態を示す。
ippStsMP4VLCCodeErr	VLC ストリームの処理中に無効な Huffman コードが検出された場合のエラー状態を示す。

PredictReconCoefIntra_MPEG4

イントラ・ブロックの適応型 DC/AC 係数予測を実行する。

```
IppStatus ippiPredictReconCoefIntra_MPEG4_16s(Ipp16s* pSrcDst,
      Ipp16s* pPredBufRow, Ipp16s* pPredBufCol, int curQP, int predQP,
      int predDir, int ACPredFlag, IppVideoComponent videoComp);
```

引数

pSrcDst	入力の現在のブロックの量子化された係数の残差（PQF）と出力の現在のブロックの量子化された係数（QF）を格納する係数バッファへのポインタ
pPredBufRow	対応する係数行バッファ内の空間的に上側のブロックの最初の要素へのポインタ（この行バッファは、予測の実行後に更新される）。詳細は、本章のバッファの項を参照のこと。
pPredBufCol	対応する係数列バッファ内の空間的に左側のブロックの最初の要素へのポインタ（この列バッファは、予測の実行後に更新される）。詳細は、本章のバッファの項を参照のこと。
curQP	現在のブロックの量子化パラメータ
predQP	予測子ブロックの量子化パラメータ
predDir	予測の方向を示す。次のいずれかの値になる。
	IPP_VIDEO_HORIZONTAL　水平方向に予測する。
	IPP_VIDEO_VERTICAL　垂直方向に予測する。
ACPredFlag	AC 予測を実行するかどうかを示すフラグ。MPEG-4 のビットストリーム構文中の ac_pred_flag と同じになる。
videoComp	現在のブロックのビデオ・コンポーネントのタイプ（輝度、クロミナンス、またはアルファ）。

説明

関数 ippiPredictReconCoefIntra_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、イントラ・ブロックの適応型 DC/AC 係数予測を実行する。DC 係数予測は、[ISO14496]、副項 7.4.3.2 に規定されている。AC 係数予測は、[ISO14496]、副項 7.4.3.3 に規定されている。

この関数の呼び出しの前に、[ISO14496]、副項 7.4.3.1 の規定に従って、予測の方向 *predDir* を選択する必要がある。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *pSrcDst*、*pPredBufRow*、*pPredBufCol* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4QPErr	*curQP* または *predQO* が [1, 31] の範囲から外れている場合のエラー状態を示す。
ippStsMP4PredDirErr	*PredDir* が有効でない場合のエラー状態を示す。

QuantInvIntraFirst_MPEG4, QuantInvInterFirst_MPEG4

第 1 の手法によって、イントラ / インター・コード化ブロックの逆量子化を実行する。

```
IppStatus ippiQuantInvIntraFirst_MPEG4_16s_I(ipp16s* pSrcDst, int QP,
      const Ipp8u* pQMatrix, int linearMode, int videoComp,
      int bitsPerPixel);
IppStatus ippiQuantInvInterFirst_MPEG4_16s_I(Ipp16s* pSrcDst, int QP,
      const Ipp8u* pQMatrix, int bitsPerPixel);
```

引数

pSrcDst	イントラ / インター・ブロックの係数バッファへのポインタ。入力の量子化された係数（QF）と出力の脱量子化された係数（F）を格納する。
QP	量子化パラメータ（quantiser_scale）
pQMatrix	量子化重み付け行列へのポインタ

linearMode	（イントラ・バージョンのみ）DC リニア・モードを示すフラグ。次のいずれかのフラグになる。 IPP_DCScalerLinear、 IPP_DCScalerNonLinear
videoComp	（イントラ・バージョンのみ）現在のブロックのビデオ・コンポーネントのタイプ。DC 係数の脱量子化に使用される。次のいずれかのフラグになる。 IPP_VIDEO_LUMINANCE、 IPP_VIDEO_CHROMINANCE、 IPP_VIDEO_ALPHA
bitsPerPixel	飽和処理に使用されるビデオ・データ精度（ビット / ピクセル単位）。[4, 12] の範囲内で有効。

説明

関数 ippiQuantInvIntraFirst_MPEG4 と ippiQuantInvInterFirst_MPEG4 は、ippmp.h ファイルの中で宣言される。これらの関数は、イントラ / インター・コード化ブロックの算術的な逆量子化、[-2 *bitsPerPixel* + 3, 2 *bitsPerPixel* + 3 – 1] の範囲内の飽和処理、不一致の制御を実行する。イントラ・ブロックの DC 係数の脱量子化は、[ISO14496]、副項 7.4.4.1.1 に規定されている。その他の係数の脱量子化は、[ISO14496]、副項 7.4.4.1.2 に規定されている。飽和処理は、[ISO14496]、副項 7.4.4.1.4 に規定されている。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *pSrcDst*、*pQMatrix* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4QPErr	*QP* が [1, 31] の範囲から外れている場合のエラー状態を示す。
ippStsMP4BitsPerPixelErr	*bitsPerPixel* が [4, 12] の範囲から外れている場合のエラー状態を示す。
ippStsMP4LinearModeErr	（イントラ・バージョンのみ）*linearMode* が有効でない場合のエラー状態を示す。

QuantInvIntraSecond_MPEG4, QuantInvInterSecond_MPEG4

第 2 の手法によって、イントラ / インター・コード化ブロックの逆量子化を実行する。

```
IppStatus ippiQuantInvIntraFirst_MPEG4_16s_I(ipp16s* pSrcDst, int QP,
     int linearMode, int videoComp, int bitsPerPixel);
IppStatus ippiQuantInvInterFirst_MPEG4_16s_I(Ipp16s* pSrcDst, int QP,
     int bitsPerPixel);
```

引数

pSrcDst	イントラ / インター・ブロックの係数バッファへのポインタ。入力の量子化された係数（QF）と出力の脱量子化された係数（F）を格納する。
QP	量子化パラメータ（quantiser_scale）
linearMode	（イントラ・バージョンのみ）DC リニア・モードを示すフラグ。次のいずれかのフラグになる。 IPP_DCScalerLinear、 IPP_DCScalerNonLinear
videoComp	（イントラ・バージョンのみ）現在のブロックのビデオ・コンポーネントのタイプ。DC 係数の脱量子化に使用される。次のいずれかのフラグになる。 IPP_VIDEO_LUMINANCE、 IPP_VIDEO_CHROMINANCE、 IPP_VIDEO_ALPHA
bitsPerPixel	飽和処理に使用されるビデオ・データ精度（ビット / ピクセル単位）。[4, 12] の範囲内で有効。

説明

関数 ippiQuantInvIntraSecond_MPEG4 と ippiQuantInvInterSecond_MPEG4 は、ippmp.h ファイルの中で宣言される。これらの関数は、イントラ / インター・コード化ブロックの算術的な逆量子化と、$[-2^{bitsPerPixel+3}, 2^{bitsPerPixel+3} - 1]$ の範囲内の飽和処理を実行する。イントラ・ブロックの DC 係数の脱量子化は、[ISO14496]、副項 7.4.4.1.1 に規定されている。その他の係数の脱量子化は、[ISO14496]、副項 7.4.4.1.2 に規定されている。飽和処理は、[ISO14496]、副項 7.4.4.1.4 に規定されている。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *pSrcDst* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4QPErr	*QP* が [1, 31] の範囲から外れている場合のエラー状態を示す。
ippStsMP4BitsPerPixelErr	*bitsPerPixel* が [4, 12] の範囲から外れている場合のエラー状態を示す。
ippStsMP4LinearModeErr	（イントラ・バージョンのみ）*linearMode* が有効でない場合のエラー状態を示す。

QuantInvIntra_MPEG4, QuantInvInter_MPEG4

イントラ / インター・コード化ブロックの逆量子化を実行する。

```
IppStatus ippiQuantInvIntra_MPEG4_16s_I(ipp16s* pSrcDst, int QP,
      const Ipp8u* pQMatrix, IppVideoComponent videoComp);
IppStatus ippiQuantInvInter_MPEG4_16s_I(Ipp16s* pSrcDst, int QP,
      const Ipp8u* pQMatrix);
```

引数

pSrcDst	イントラ / インター・ブロックの係数バッファへのポインタ。入力の量子化された係数（QF）と出力の脱量子化された係数（F）を格納する。
QP	量子化パラメータ（quantiser_scale）
pQMatrix	量子化重み付け行列へのポインタ。*pQMatrix* が NULL の場合、この関数は第 2 の逆量子化手法を使用する。NULL でない場合、この関数は第 1 の手法を使用する。
videoComp	（イントラ・バージョンのみ）現在のブロックのビデオ・コンポーネントのタイプ。DC 係数の脱量子化に使用される。次のいずれかのフラグになる。

IPP_VIDEO_LUMINANCE、
IPP_VIDEO_CHROMINANCE、
IPP_VIDEO_ALPHA

説明

関数 ippiQuantInvIntra_MPEG4 と ippiQuantInvInter_MPEG4 は、ippmp.h ファイルの中で宣言される。これらの関数は、イントラ / インター・コード化ブロックの算術的な逆量子化、飽和処理、および（第 1 の逆量子化手法の場合のみ）不一致の制御を実行する。これらの関数は、bits_per_pixel =8 をサポートしている。つまり、出力係数は、[-2048,2047] の範囲で飽和処理される。イントラ・ブロックの DC 係数の脱量子化は、[ISO14496]、副項 7.4.4.1.1、7.4.4.3 に規定されている。第 1 の手法によるその他の係数の脱量子化は、[ISO14496]、副項 7.4.4.1.2 に規定されている。第 2 の手法によるその他の係数の脱量子化は、[ISO14496]、副項 7.4.4.1.3 に規定されている。飽和処理は、[ISO14496]、副項 7.4.4.1.4 に規定されている。不一致の制御は、[ISO14496]、副項 7.4.4.1.5 に規定されている。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *pSrcDst* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4QPErr	*QP* が [1, 31] の範囲から外れている場合のエラー状態を示す。

DecodeBlockCoef_Intra_MPEG4

INTRA ブロックの係数をデコードする。

```
IppStatus ippiDecodeBlockCoef_Intra_MPEG4_1u8u(Ipp8u** ppBitStream,
     int* pBitOffset, Ipp8u* pDst, int step, Ipp16s* pCoefBufRow,
     Ipp16s* pCoefBufCol, Ipp8u curQP, Ipp8u* pQPBuf, Ipp8u* pQMatrix,
     int blockIndex, int bitsPerPixel, int intraDCVLC,
     int ACPredFlag);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ。**ppBitStream* はブロックのデコード後に更新される。

pBitOffset	*ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。*pBitOffset* はブロックのデコード後に更新される。
pDst	デスティネーション・プレーン内のブロックへのポインタ
step	デスティネーション・プレーンの幅（バイト単位）
pCoefBufRow	係数行バッファへのポインタ。ブロックのデコード後に更新される。
pCoefBufCol	係数列バッファへのポインタ。ブロックのデコード後に更新される。
curQP	現在のブロックが所属するマクロブロックの量子化パラメータ
pQPBuf	量子化パラメータ・バッファへのポインタ
pQMatrix	イントラ・マクロブロックの量子化重み付け行列へのポインタ。このポインタが NULL の場合は、第 2 の逆量子化手法が使用される。NULL でない場合は、第 1 の逆量子化手法が使用される。
blockIndex	[ISO14496]、副項 6.1.3.8、図 6-5 に定義されたコンポーネントのタイプと位置を示すブロック・インデックス。さらに、インデックス 6 ～ 9 は、同じマクロブロック内の輝度ブロック 0 ～ 3 に空間的に対応するアルファ・ブロックを示す。
intraDCVLC	Intra AC の VLC の代わりに Intra DC の VLC を使用して DC 係数をデコードするかどうかを示すフラグ。このフラグは、[ISO14496]、表 6-21 に従って、intra_dc_vlc_thr と QP によって決定される。
ACPredFlag	ac_pred_flag （輝度）または ac_pred_flag_alpha （アルファ・ブロック）と同じ値のフラグ。最初の行の AC 係数を予測するか、それとも最初の列の AC 係数を予測するかを示す。

説明

関数 ippiDecodeBlockCoef_Intra_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、INTRA ブロック係数をデコードする。この関数は、可変長デコード（[ISO14496]、副項 7.4.1.1, 7.4.1.2、7.4.1.3）、逆ジグザグ・スキャン（[ISO14496]、副項 7.4.2）、逆方向の DC および AC 予測 （[ISO14496]、副項 7.4.3）、逆方向の脱量

子化 （[ISO14496]、副項 7.4.4）、逆 DCT を実行する。出力値は、[0, 255] の範囲でクリッピングされ、デスティネーション・プレーン内の現在のフレームに書き込める状態になる。

係数バッファは、あらかじめ定義された構造に従って更新される（詳細は、本章のバッファの項を参照）。

この関数は、現在のブロックの 0 でない AC 係数が、ビットストリーム内に少なくとも 1 つ存在する場合にのみ使用される。DC 係数のみの場合は、関数 ippiDecodeBlockCoef_IntraDCOnly_MPEG4 が使用される。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *ppBitStream*、**ppBitStream*、*pBitOffset*、*pDst*、*pQPBuf*、*pCoefBufRow*、*pCoefBufCol* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4BitOffsetErr	**pBitOffset* が [0, 7] の範囲から外れている場合のエラー状態を示す。
ippStsStepErr	*step* が 8 より小さいか、4 で割り切れない場合のエラー状態を示す。
ippStsMP4QPErr	curQP または pQPBuf の必要な要素が [1, 31] の範囲から外れている場合のエラー状態を示す。
ippStsMP4BlockIdxErr	*blockIndex* が [0, 9] の範囲から外れている場合のエラー状態を示す。
ippStsMP4DCCodeErr	DC ストリームの処理中に無効なコードが検出された場合のエラー状態を示す。
ippStsMP4VLCCodeErr	VLC ストリームの処理中に無効な Huffman コードが検出された場合のエラー状態を示す。

DecodeBlockCoef_IntraDCOnly_MPEG4

DC 係数だけを含む INTRA ブロックをデコードする。

```
IppStatus ippiDecodeBlockCoef_IntraDCOnly_MPEG4_1u8u (Ipp8u**
      ppBitStream, int* pBitOffset, Ipp8u* pDst, int dstStep,
      Ipp16s* pCoefBufRow, Ipp16s* pCoefBufCol, Ipp8u curQp,
      Ipp8u* pQpBuf, const Ipp8u* pQMatrix, int blockIndex,
      int IntraDCVLC, int ACPredFlag);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ。**ppBitStream* はブロックのデコード後に更新される。
pBitOffset	*ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。*pBitOffset* はブロックのデコード後に更新される。
pDst	デスティネーション・プレーン内のブロックへのポインタ
step	デスティネーション・プレーンの幅（バイト単位）
pCoefBufRow	係数行バッファへのポインタ。ブロックのデコード後に更新される。
pCoefBufCol	係数列バッファへのポインタ。ブロックのデコード後に更新される。
curQP	現在のブロックが所属するマクロブロックの量子化パラメータ
pQPBuf	量子化パラメータ・バッファへのポインタ
pQMatrix	イントラ・マクロブロックの量子化重み付け行列へのポインタ。このポインタが NULL の場合は、第 2 の逆量子化手法が使用される。NULL でない場合は、第 1 の逆量子化手法が使用される。
blockIndex	[ISO14496]、副項 6.1.3.8、図 6-5 に定義されたコンポーネントのタイプと位置を示すブロック・インデックス。さらに、インデックス 6 ～ 9 は、同じマクロブロック内の輝度ブロック 0 ～ 3 に空間的に対応するアルファ・ブロックを示す。

intraDCVLC	Intra AC の VLC の代わりに Intra DC の VLC を使用して DC 係数をデコードするかどうかを示すフラグ。このフラグは、[ISO14496]、表 6-21 に従って、intra_dc_vlc_thr と QP によって決定される。
ACPredFlag	ac_pred_flag （輝度）または ac_pred_flag_alpha （アルファ・ブロック）と同じ値になるフラグ。イントラ・コード化された MB について、最初の行の AC 係数を差分的にコード化するか、それとも最初の列の AC 係数を差分的にコード化するかを示す。

説明

関数 ippiDecodeBlockCoef_IntraDCOnly_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、DC 係数だけがエンコードされている INTRA ブロックをデコードする。この関数は、可変長デコード（[ISO14496]、副項 7.4.1.1）、逆ジグザグ・スキャン（[ISO14496]、副項 7.4.2）、逆方向の DC および AC 予測（[ISO14496]、副項 7.4.3）、逆方向の脱量子化 （[ISO14496]、副項 7.4.4）、逆 DCT を実行する。出力値は、[0, 255] の範囲でクリッピングされ、デスティネーション・プレーン内の現在のフレームに書き込める状態になる。

係数バッファは、あらかじめ定義された構造に従って更新される（詳細は、本章のバッファの項を参照）。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタのうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4BitOffsetErr	*pBitOffset が [0, 7] の範囲から外れている場合のエラー状態を示す。
ippStsStepErr	*step* が 8 より小さいか、4 で割り切れない場合のエラー状態を示す。
ippStsMP4QPErr	*curQP* または *pQPBuf* の必要な要素が [1, 31] の範囲から外れている場合のエラー状態を示す。
ippStsMP4BlockIdxErr	*blockIndex* が [0, 9] の範囲から外れている場合のエラー状態を示す。
ippStsMP4VDCCodeErr	DC ストリームの処理中に無効なコードが検出された場合のエラー状態を示す。

DecodeBlockCoef_Inter_MPEG4

INTER ブロック係数をデコードする。

```
IppStatus ippiDecodeBlockCoef_Inter_MPEG4_1u16s(Ipp8u** ppBitStream,
     int* pBitOffset, Ipp16s* pDst, int QP, const Ipp8u* pQMatrix);
```

引数

ppBitStream	ビットストリーム・バッファ内の現在のバイトへのポインタへのポインタ。**ppBitStream* はブロックのデコード後に更新される。
pBitOffset	*ppBitStream* によって指定されるバイト内のビット位置へのポインタ。0 ～ 7 の範囲内で有効。*pBitOffset* はブロックのデコード後に更新される。
pDst	デコードされた残差バッファへのポインタ
QP	量子化パラメータ
pQMatrix	イントラ・マクロブロックの量子化重み付け行列へのポインタ。このポインタが NULL の場合は、第 2 の逆量子化手法が使用される。NULL でない場合は、第 1 の逆量子化手法が使用される。

説明

関数 ippiDecodeBlockCoef_Inter_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、INTER ブロック係数をデコードする。この関数は、可変長デコード（[ISO14496]、副項 7.4.1.2、7.4.1.3）、標準形式の逆ジグザグ・スキャン（[ISO14496]、副項 7.4.2）、逆方向の脱量子化 （[ISO14496]、副項 7.4.4）、逆 DCT を実行する。出力バッファには、その後の再構成のための残差が格納される。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *ppBitStream*、**ppBitStream*、*pBitOffset*、*pDst* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。
ippStsMP4BitOffsetErr	**pBitOffset* が [0, 7] の範囲から外れている場合のエラー状態を示す。

ippStsMP4QPErr	*QP* が [1, 31] の範囲から外れている場合のエラー状態を示す。
ippStsMP4VLCCodeErr	VLC ストリームの処理中に無効な Huffman コードが検出された場合のエラー状態を示す。

FilterDeblocking_HorEdge_MPEG4, FilterDeblocking_VerEdge_MPEG4

2 つの隣接するブロックのデブロック・フィルタリングを水平エッジまたは垂直エッジ上で実行する。

```
IppStatus ippiFilterDeblocking_HorEdge_MPEG4_8u_I(Ipp8u* pSrcDst, int
     step, int QP, int THR1, int THR2);
IppStatus ippiFilterDeblocking_VerEdge_MPEG4_8u_I(Ipp8u* pSrcDst, int
     step, int QP, int THR1, int THR2);
```

引数

pSrcDst	下側（HorEdge）または右側（VerEdge）のブロックの最初のピクセルへのポインタ
step	ソース・プレーンの幅（バイト単位）
QP	量子化パラメータ
THR1, THR2	フィルタ・モード（[ISO14496]、付録 F.3.1）を指定するしきい値

説明

関数 ippiFilterDeblocking_HorEdge_MPEG4 と ippiFilterDeblocking_VerEdge_MPEG4 は、ippmp.h ファイルの中で宣言される。これらの関数は、2 つの隣接するブロックのデブロック・フィルタリングを、それぞれ水平エッジまたは垂直エッジ上で実行する（[ISO14496]、付録 F.3.1）。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	ポインタ *pSrcDst* のうち少なくとも 1 つが NULL の場合のエラー状態を示す。

ippStsMP4QPErr	*QP* が [1, 31] の範囲から外れている場合のエラー状態を示す。

FilterDeringingThresholdMB_MPEG4

マクロブロックのデリンギング・フィルタリング用のしきい値を計算する。

```
IppStatus ippiFilterDeringingThresholdMB_MPEG4_8u(Ipp8u* pSrcY,
     int stepY, Ipp8u* pSrcCb, int stepCb, Ipp8u* pSrcCr, int stepCr,
     int threshold[6]);
```

引数

pSrcY	現在のマクロブロック内の左上の Y ブロックへのポインタ
stepY	輝度プレーンの幅（バイト単位）
pSrcCb	現在のマクロブロック内の Cb ブロックへのポインタ
stepCb	クロミナンス （Cb）プレーンの幅（バイト単位）
pSrcCr	現在のマクロブロック内の Cr ブロックへのポインタ
stepCr	クロミナンス （Cr）プレーンの幅（バイト単位）
threshold	すべてのブロックのしきい値の配列

説明

関数 ippiFilterDeringingThresholdMB_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、しきい値の計算を実行する（[ISO14496]、付録 F.3.2.1）。しきい値の計算は、デリンギング・フィルタリングの最初のサブプロセスである。得られたしきい値は、関数 ippiFilterDeringingSmoothBlock_MPEG4 に必要である。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つが NULL の場合のエラー状態を示す。

FilterDeringingSmoothBlock_MPEG4

ブロックのデリンギング・フィルタリング
を実行する。

```
IppStatus ippiFilterDeringingSmoothBlock_MPEG4_8u(Ipp8u* pSrc, int
     stepSrc, Ipp8u* pDst, int stepDst, int QP, int threshold);
```

引数

pSrc	ソース・ブロックへのポインタ
stepSrc	ソース・プレーンの幅（バイト単位）
QP	量子化パラメータ（quantiser_scale）
threshold	ブロックのしきい値
pDst	デスティネーション・ブロックへのポインタ
stepDst	デスティネーション・プレーンの幅（バイト単位）

説明

関数 ippiFilterDeringingSmoothBlock_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、デリンギング・フィルタリング （厳密には、ソース・ブロック *pSrc* のインデックスの取得とアダプティブ・スムージング （[ISO14496]、付録 F.3.2.1、3.2.2）） を実行し、フィルタリングの結果をデスティネーション・ブロック *pDst* に格納する。しきい値 *threshold* は、この関数の前に呼び出される補助関数 ippiFilterDeringingThresholdMB_MPEG4 によって返される。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つが NULL の場合のエラー状態を示す。
ippStsMP4QPErr	*QP* が [1, 31] の範囲から外れている場合のエラー状態を示す。

AverageBlock_MPEG4, AverageMB_MPEG4

2 つのブロック / マクロブロックの平均を求める。

```
IppStatus ippiAverageBlock_MPEG4_8u(Ipp8u* pSrc1, int src1Step, Ipp8u*
     pSrc2, int src2Step, Ipp8u* pDst, int dstStep);
IppStatus ippiAverageBlock_MPEG4_8u_I(Ipp8u* pSrcDst, int srcDstStep,
     Ipp8u* pSrc, int srcStep);
IppStatus ippiAverageMB_MPEG4_8u(Ipp8u* pSrc1, int src1Step,
     Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep);
IppStatus ippiAverageMB_MPEG4_8u_I(Ipp8u* pSrcDst, int srcDstStep,
     Ipp8u* pSrc, int srcStep);
```

引数

pSrc, pSrc1, pSrc2	ソース・ブロック / マクロブロックへのポインタ
srcStep, src1Step, src2Step	ソース・プレーンの幅（バイト単位）
pDst	デスティネーション・ブロック / マクロブロックへのポインタ
dstStep	デスティネーション・プレーンの幅（バイト単位）
pSrcDst	（インプレース操作の場合）ソースおよびデスティネーション・ブロック / マクロブロックへのポインタ
srcDstStep	（インプレース操作の場合）ソースおよびデスティネーション・プレーンの幅（バイト単位）

説明

関数 ippiAverageBlock_MPEG4 と ippiAverageMB_MPEG4 は、ippmp.h ファイルの中で宣言される。これらの関数は、[ISO14496]、副項 7.6.9.4 の規定に従って、ピクセルごとに 2 つのブロック / マクロブロックの平均を求める。
この関数は、B フレームの再構成に使用される。

戻り値

ippStsNoErr	エラーがないことを示す。

ippStsNullPtrErr	指定されたポインタの 1 つが NULL の場合のエラー状態を示す。

CopyBlockHalfpel_MPEG4, CopyMBHalfpel_MPEG4

ハーフ・ピクセル精度でブロック / マクロブロックをコピーする。

```
IppStatus ippiCopyBlockHalfpel_MPEG4_8u(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, const IppMotionVector* pMV, int roundControl);
IppStatus ippiCopyMBHalfpel_MPEG4_8u(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, const IppMotionVector* pMV, int roundControl);
```

引数

pSrc	レファレンス・ブロック / マクロブロックへのポインタ
srcStep	ソース・プレーンの幅（バイト単位）
pDst	デスティネーション・ブロック / マクロブロックへのポインタ
dstStep	デスティネーション・プレーンの幅（バイト単位）
pMV	現在のブロック / マクロブロックの動きベクトルへのポインタ
roundControl	ハーフ・ピクセルの近似に使用される丸めのタイプを指定するパラメータ。0 または 1。

説明

関数 ippiCopyBlockHalfpel_MPEG4 と ippiCopyMBHalfpel_MPEG4 は、ippmp.h ヘッダ・ファイルの中で宣言される。これらの関数は、ポインタ *pSrc* と動きベクトル *pMV* によって指定される領域のピクセル値を、ハーフ・ピクセル精度（[ISO14496]、副項 7.6.2.1）でデスティネーション・ブロック / マクロブロック *pDst* にコピーする。この関数は、P-VOP と B-VOP の再構成に使用される。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つが NULL の場合のエラー状態を示す。

ReconBlockHalfpel_MPEG4

ハーフ・ピクセル精度でブロックを再構成する。

```
IppStatus ippiReconBlockHalfpel_MPEG4_8u(const Ipp8u* pSrc,
     int srcStep, Ipp16s pResidue[64], Ipp8u* pDst, int dstStep,
     const IppMotionVector* pMV, int roundControl);
```

引数

pSrc	リファレンス・ブロックへのポインタ
srcStep	ソース・プレーンの幅（バイト単位）
pDst	デスティネーション・ブロックへのポインタ
dstStep	デスティネーション・プレーンの幅（バイト単位）
pResidue	残差の値の配列へのポインタ
pMV	現在のブロック / マクロブロックの動きベクトル
roundControl	ハーフ・ピクセルの近似に使用される丸めのタイプを指定するパラメータ。0 または 1。

説明

関数 ippiReconBlockHalfpel_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、逆変換の結果（残差）*pResidue* を加算することにより、ポインタ *pSrc* と動きベクトル *pMV* によって指定される領域を再構成する。再構成はハーフ・ピクセル精度（[ISO14496]、副項 7.6.2.1）で実行される。再構成の結果はデスティネーション・ブロック *pDst* に格納される。この関数は、P-VOP と B-VOP の再構成に使用される。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つが NULL の場合のエラー状態を示す。

OBMCHalfpel_MPEG4

重複ブロック動き補償（OBMC）を実行する。

```
IppStatus ippiOBMCHalfpel_MPEG4_8u(const Ipp8u* pSrc, int srcStep,
     Ipp8u* pDst, int dstStep, const IppMotionVector* pMVCur,
     const IppMotionVector* pMVLeft, const IppMotionVector* pMVRight,
     const IppMotionVector* pMVAbove, const IppMotionVector* pMVBelow,
     int roundControl);
```

引数

pSrc	ソース・ブロックへのポインタ
srcStep	ソース・プレーンの幅（バイト単位）
pDst	デスティネーション・ブロックへのポインタ
dstStep	デスティネーション・プレーンの幅（バイト単位）
pMVCur	現在のブロックの動きベクトルへのポインタ
pMVLeft	左側のブロックの動きベクトルへのポインタ
pMVRight	右側のブロックの動きベクトルへのポインタ
pMVAbove	上側のブロックの動きベクトルへのポインタ
pMVBelow	下側のブロックの動きベクトルへのポインタ
roundControl	ハーフ・ピクセルの近似に使用される丸めのタイプを指定するパラメータ。0　または　1。

説明

関数 ippiOBMCHalfpel_MPEG4 は、ippmp.h ファイルの中で宣言される。この関数は、*pSrc* の重複ブロック動き補償をハーフ・ピクセル精度（[ISO14496]、副項 7.6.2.1）で実行し、処理の結果をデスティネーション・ブロック *pDst* に格納する。重複動き補償は、[ISO14496]、副項 7.6.6 に規定されている。

戻り値

ippStsNoErr	エラーがないことを示す。
ippStsNullPtrErr	指定されたポインタの 1 つが NULL の場合のエラー状態を示す。

18 ビデオ符号化

ビデオ符号化用インテグレーテッド・パフォーマンス・プリミティブは、MPEG-1（[ISO11172]）、MPEG-2（[ISO13818]）、MPEG-4（[ISO14496A]）、H.263（[ITUH263]）規格に準拠するビデオ・データのエンコードおよびデコード用関数のライブラリである。これらの関数は、エンコードおよびデコード・パイプラインに最適なソリューションを提供する。ビデオ符号化関数は、インテル® IPP の他の関数と同じように、高性能クロスプラットフォーム・コードの開発に使用される。

画像処理用インテル IPP に使用される共通のデータ・タイプ（第 2 章の「データ・タイプ」を参照）以外に、ビデオ符号化関数には、独自のデータ・タイプが定義されている

表 18-1　標準データ・タイプとインテル® IPP のデータ・タイプ

色分解能	標準型	インテル IPP 型
32	符号付き整数	IppVCHuffmanSpec_32s

ビデオ符号化関数の関数名中の記述子（第 2 章の「関数の命名」を参照）は、関数の動作を規定する規格（MPEG1, MPEG2, MPEG4, H263）か、まとめて処理される要素のブロックのサイズ（16 × 16、16 × 8、8 × 16、8 × 8、8 × 4）を示す。最初の数値はブロックの幅、2 番目の数値は高さである。1 つのチャネルを処理する関数には、記述子 C1 が使用される。

列挙子 IPPVC_MC_APX は、実行される動き補償のタイプを示す。

```
typedef enum _IPPVC_MC_APX{
        IPPVC_MC_APX_FF =       0x0,
        IPPVC_MC_APX_FH =       0x4,
        IPPVC_MC_APX_HF =       0x8,
        IPPVC_MC_APX_HH =       0x0c
    }IPPVC_MC_APX;
```

最初の記述子 FF、FH、HF、HH は、動きベクトル処理の精度がフル・ピクセル（F）かハーフ・ピクセル（H）かを示す。最初の文字は水平方向の精度を示し、2 番目の文字は垂直方向の精度を示す。

次の表に、インテル IPP ビデオ符号化関数の一覧を示す。

表 18-2　インテル® IPP ビデオ・デコード関数

関数の基本名	説明
可変長デコード関数	
VCHuffmanDecodeInitAlloc	メモリを割り当てて、マクロブロック・アドレスのインクリメント、マクロブロックのタイプ、マクロブロックのパターン、または動きベクトルのコードを格納するテーブルを初期化する。
VCHuffmanDecodeInitAllocRL	メモリを割り当てて、実行レベル・コードのテーブルを初期化する。
VCHuffmanDecodeOne	指定されたテーブルを使用して、ビット・ストリームからコードをデコードする。
ReconstructDCTBlock_MPEG1	標準 MPEG1 用の実行レベル・コードのテーブルを使用して 8 × 8 非イントラ・ブロックをデコードし、再編成と逆量子化を実行する。
ReconstructDCTBlockIntra_MPEG1	標準 MPEG1 用の実行レベル・コードのテーブルを使用して 8 × 8 イントラ・ブロックをデコードし、再編成と逆量子化を実行する。
ReconstructDCTBlock_MPEG2	標準 MPEG2 用の実行レベル・コードのテーブルを使用して 8 × 8 非イントラ・ブロックをデコードし、再編成と逆量子化を実行する。
ReconstructDCTBlockIntra_MPEG2	標準 MPEG2 用の実行レベル・コードのテーブルを使用して 8 × 8 イントラ・ブロックをデコードし、再編成と逆量子化を実行する。
VCHuffmanDecodeFree	VLC テーブルに割り当てられたメモリを解放する。
逆量子化関数	
QuantInvIntra_MPEG2	MPEG-2 規格に従って、イントラ・フレームの逆量子化を実行する。
QuantInv_MPEG2	MPEG-2 規格に従って、非イントラ・フレームの逆量子化を実行する。
QuantInvIntra_MPEG4	MPEG-1 および MPEG-4 規格に従って、イントラ・フレームの逆量子化を実行する。
QuantInv_MPEG4	MPEG-1 および MPEG-4 規格に従って、非イントラ・フレームの逆量子化を実行する。
QuantInvIntra_H263	H263 規格に従って、イントラ・フレームの逆量子化を実行する。
QuantInv_H263	H263 規格に従って、非イントラ・フレームの逆量子化を実行する。
動き補償関数	
MC16x16	予測される 16 × 16 ブロックの動き補償を実行する。

表 18-2　インテル® IPP ビデオ・デコード関数

関数の基本名	説明
MC16x8	予測される 16 × 8 ブロックの動き補償を実行する。
MC8x16	予測される 8 × 16 ブロックの動き補償を実行する。
MC8x8	予測される 8 × 8 ブロックの動き補償を実行する。
MC8x4	予測される 8 × 4 ブロックの動き補償を実行する。
MC16x16B	双方向予測される 16 × 16 ブロックの動き補償を実行する。
MC16x8B	双方向予測される 16 × 8 ブロックの動き補償を実行する。
MC8x16B	双方向予測される 8 × 16 ブロックの動き補償を実行する。
MC8x8B	双方向予測される 8 × 8 ブロックの動き補償を実行する。
MC8x4B	双方向予測される 8 × 4 ブロックの動き補償を実行する。

表 18-3　インテル® IPP ビデオ・エンコード関数

関数の基本名	説明
動き推定関数と動き補償関数	
GetDiff16x16	16 × 16 要素の現在の予測されるブロックとリファレンス・ブロックの差を求める。
GetDiff16x8	16 × 8 要素の現在の予測されるブロックとリファレンス・ブロックの差を求める。
GetDiff8x8B	8 × 8 要素の現在の予測されるブロックとリファレンス・ブロックの差を求める。
GetDiff8x16	8 × 16 要素の現在の予測されるブロックとリファレンス・ブロックの差を求める。
GetDiff8x4	8 × 4 要素の現在の予測されるブロックとリファレンス・ブロックの差を求める。
GetDiff16x16B	16 × 16 要素の現在の双方向予測されるブロックと 2 つのリファレンス・ブロックの平均の差を求める。
GetDiff16x8B	16 × 8 要素の現在の双方向予測されるブロックと 2 つのリファレンス・ブロックの平均の差を求める。
GetDiff8x8B	8 × 8 要素の現在の双方向予測されるブロックと 2 つのリファレンス・ブロックの平均の差を求める。

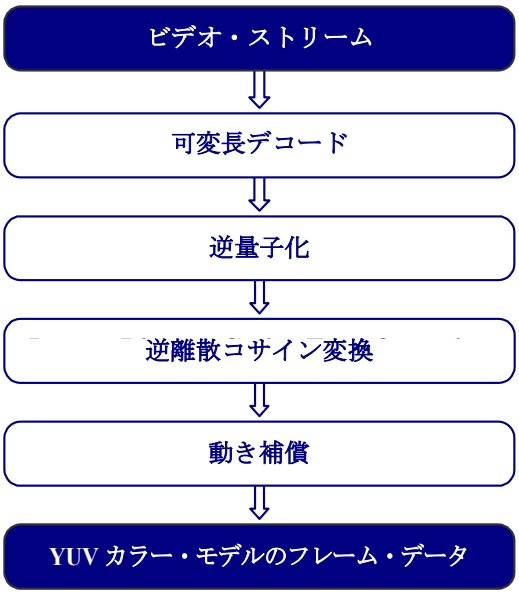
表 18-3　インテル® IPP ビデオ・エンコード関数 （続き）

関数の基本名	説明
GetDiff8x16B	8 × 16 要素の現在の双方向予測されるブロックと 2 つのリファレンス・ブロックの平均の差を求める。
GetDiff8x4B	8 × 4 要素の現在の双方向予測されるブロックと 2 つのリファレンス・ブロックの平均の差を求める。
SAD16x16	現在のブロックとリファレンス・ブロックの差の絶対値の合計を求める。
Variance16x16	現在のブロックの分散を求める。
SqrDiff16x16	現在のブロックとリファレンス・ブロックの差の 2 乗の合計を求める。
SqrDiff16x16B	現在の双方向予測されるブロックと 2 つのリファレンス・ブロックの平均の差の 2 乗の合計を求める。
量子化関数	
QuantIntra_MPEG2	MPEG-2 規格に従って、指定された量子化行列を使用してイントラ・ブロックの DCT 係数の量子化をインプレースで実行する。
QuantIntra_MPEG4	MPEG-4 規格に従って、指定された量子化行列を使用してイントラ・ブロックの DCT 係数の量子化をインプレースで実行する。
QuantIntra_H263	H263 規格に従って、指定された量子化行列を使用してイントラ・ブロックの DCT 係数の量子化をインプレースで実行する。
Quant_MPEG2	MPEG-2 規格に従って、指定された量子化行列を使用して非イントラ・ブロックの DCT 係数の量子化をインプレースで実行する。
Quant_MPEG4	MPEG-4 規格に従って、指定された量子化行列を使用して非イントラ・ブロックの DCT 係数の量子化をインプレースで実行する。
Quant_H263	H263 規格に従って、指定された量子化行列を使用して非イントラ・ブロックの DCT 係数の量子化をインプレースで実行する。
ハフマン・エンコード関数	
EncodeTableInitAlloc	実行レベルのエンコード・テーブルを作成する。
PutIntraBlock	イントラ・ブロックのエンコード、再編成、ビット・ストリーム内への配置を実行する。
PutNonIntraBlock	非イントラ・ブロックのエンコード、再編成、ビット・ストリーム内への配置を実行する。

ビデオ・データ・デコード関数

本節では、図 18-1 に示す標準的なデコード・パイプラインに従って、MPEG ビデオ・デコードの主な手順について説明する。.

図 18-1　標準的なデコード・パイプライン



注： 逆 DCT は、最も一般的な変換の 1 つである。逆 DCT は、非イントラ型の 8 × 8 ブロックの場合は IPP 画像処理関数 `ippiDCT8x8Inv_16s_C1R`、イントラ型の 8 × 8 ブロックの場合は `ippiDCT8x8Inv_16s8u_C1R` を使用して実行できる。

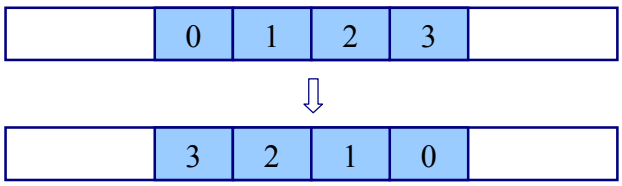
可変長デコード

ビット・ストリーム内のビデオ・データは、可変長コード（VLC）テーブルを使用して、最も短いコードが最も頻度の高い値に対応し、長いコードほど頻度の低い値に対応するようにエンコードされる。MPEG 標準テーブルは、可能なコードと対応する値を格納する。

ビデオ・データのデコードでは、最初にメモリを割り当てて、テーブルを初期化する。次に、デコードそれ自体を実行し、割り当てられたメモリを解放する。

デコードの実行とインテル IPP 関数の適用の前に、デコード関数のパフォーマンスを上げるために、ビット・ストリーム内の各 32 ビット・ダブルワード内のすべてのバイトをフリップする必要がある（図 18-2）。

図 18-2　バイトのフリップ



メモリの割り当てと初期化

ビデオ・デコード関数は、メモリを割り当てて、次の構造を持つテーブルを初期化する。

例 18-1　ソース・テーブルの構造

```
static Ipp32s Table[]=
{
max_bits;                              The maximum length of code
total_subt;                            The total number of all subtables
sub_sz1;                               The sizes of all subtables. Their sum
                                       must be equal to the maximum length of
                                       code.
sub_sz2;
…;
sub_szTotal;
N1;                                    The number of 1-bit codes
(code1, value1) or (code1, run1,       The 1-bit codes and values or codes,
level1) for RL table;                  run values and level values for
                                       RL-tables. The number of groups must be
                                       equal to sub_sz1.
(code2, value2) or (code2, run2,
level2) for RL table;
…;
(codeN1,valueN1) or (codeN1, runN1,
levelN1) for RL table;
N2;                                    The number of 2-bit codes
```

例 18-1　ソース・テーブルの構造　（続き）

(code1, value1) or(code1, run1, level1) for RL table;	The 2-bit codes and values or codes, run values and level values for RL-tables. The number of groups must be equal to sub_sz2.
(code2, value2) or(code2, run2, level2) for RL table;	
…;	
(codeN2,valueN2) or(codeN2, runN2, levelN2) for RL table;	
…;	…
Nm;	The number of maximum length codes
(code1, value1) or(code1, run1, level1) for RL table;	The maximum length codes and values or codes, run values and level values for RL-tables. The number of groups must be equal to sub_szTotal.
(code2, value2) or(code2, run2, level2) for RL table;	
…;	
(codeNm,valueNm) or(codeNm, runNm, levelNm) for RL table;	
-1;	The significant value to indicate the end of table
};	

VCHuffmanDecodeInitAlloc

メモリを割り当てて、テーブルを初期化する。

```
IppStatus ippiVCHuffmanDecodeInitAlloc_32s (const Ipp32s* pSrcTable,
    IppVCHuffmanSpec_32s** pDstSpec);
```

引数

pSrcTable	ソース・テーブルへのポインタ
pDstSpec	デスティネーション・デコード・テーブルへの二重ポインタ

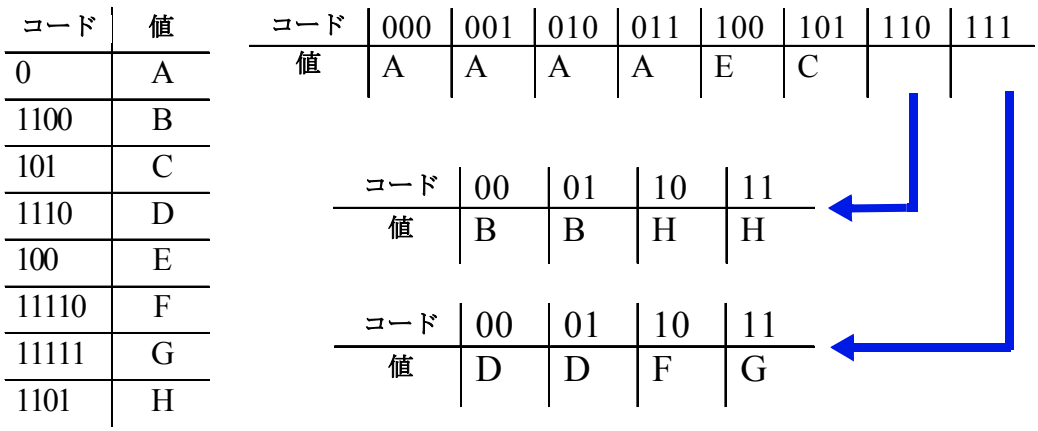
説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiVCHuffmanDecodeInitAlloc_32s は、メモリを割り当てて、マクロブロック・アドレスのインクリメント、マクロブロックのタイプ、マクロブロックのパターン、または動きベクトルのコードを格納するテーブルを初期化する。この関数は、*pDstSpec が指定するアドレスにメモリを割り当て、pSrcTable が指定するソース・テーブルから得られるデータを格納する。ソース・テーブルの構造は、例 18-1 を参照のこと。

デコード中に、最も頻度の高い（最も短い）コードを格納する最初のサブテーブる（図 18-3）内で、コードの検索が実行される。コード値が見つからない場合は、より長いコードの追加ビットを格納する後続のサブテーブル内で検索が続けられる。それでもコード値が見つからない場合は、さらに次のサブテーブルに検索が進められる。

サブテーブルの数が多いほど、テーブル構造のサイズは小さくなるが、デコード操作の回数が増える。このようなテーブル構造により、テーブルの冗長性とデコード操作の回数の間の最適な比が得られる。

図 18-3　VLC テーブルとサブテーブル



戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

VCHuffmanDecodeInitAllocRL

メモリを割り当てて、実行レベル・コードのテーブルを初期化する。

```
IppStatus ippiVCHuffmanDecodeInitAllocRL_32s (const Ipp32s* pSrcTable,
    IppVCHuffmanSpec_32s** pDstSpec);
```

引数

pSrcTable	ソース・テーブルへのポインタ
pDstSpec	デスティネーション・デコード・テーブルへの二重ポインタ

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiVCHuffmanDecodeInitAllocRL_32s は、メモリを割り当てて、DCT 係数のコード（すなわち、実行レベル・コード）を格納するテーブルを初期化する。この関数は、*pDstSpec が指定するアドレスにメモリを割り当て、*pSrcTable* が指定するソース・テーブルから得られるデータを格納する。

ソース・テーブルの構造は、例 18-1 を参照のこと。VLC テーブルとサブテーブルの例は、図 18-3 を参照のこと。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

可変長コードのデコード

この関数は、DCT 係数以外のすべてのコードをデコードする。

VCHuffmanDecodeOne

指定されたテーブルを使用して、1 つの
コードをデコードする。

```
IppStatus ippiVCHuffmanDecodeOne_1u32s (Ipp32u** pBitStream, int* pOffset,
    Ipp32s* pDst, const IppVCHuffmanSpec_32s *pDecodeTable);
```

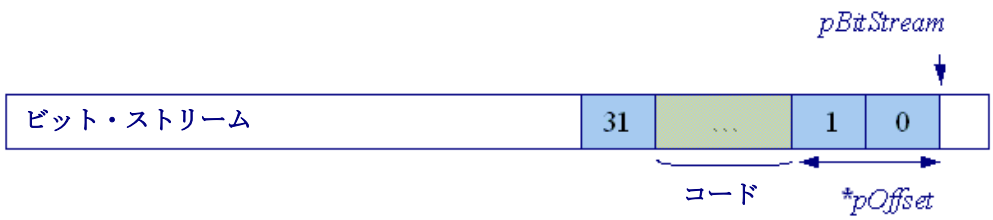
引数

pBitStream	ビット・ストリーム内の現在の位置への二重ポインタ
pOffset	*pBitStream* が指すビットとコードの始点の間のオフセットへのポインタ
pDst	デスティネーション結果へのポインタ
pDecodeTable	デコード・テーブルへのポインタ

説明

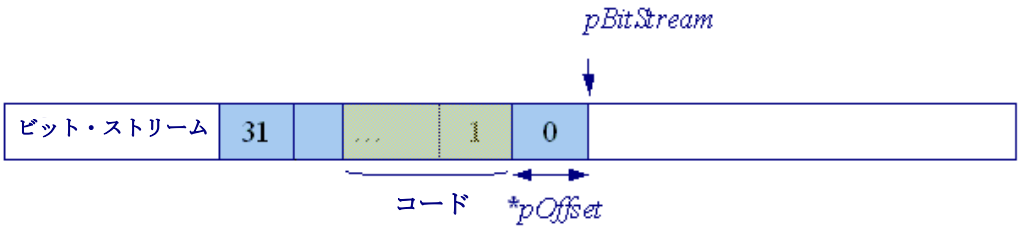
この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiVCHuffmanDecodeOne_1u32s は、指定されたテーブルを使用してビット・ストリームから 1 つのコードをデコードし、結果をデスティネーション・データに送り、ポインタを新しい位置にリセットする（図 18-4 と図 18-5 を参照）。

図 18-4　VCHuffmanDecodeOne_1u32s の入力データ



ポインタ *pBitStream* は、32 ビット値を指す。ビット・オフセットは、0 ～ 31 の範囲で変化する。処理の終了後、図 18-5 に示すように、ポインタは変更され、ポインタの新しい値が返される。

図 18-5　VCHuffmanDecodeOne_1u32s の出力データ



戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。
IppStsH263VLCCodeErr	H263 規格に基づくデコード中のエラーを示す。

ブロック処理関数

ブロック処理関数は、実行レベル・テーブルを使用して、DCT 係数の非イントラおよびイントラ 8 × 8 ブロックのデコード、再編成、逆量子化を実行する。

ReconstructDCTBlock_MPEG1

標準 MPEG1 用の実行レベル・コードのテーブルを使用して 8 × 8 非イントラ・ブロックをデコードし、再編成と逆量子化を実行する。

```
IppStatus ippiReconstructDCTBlock_MPEG1_32s (Ipp32u **pBitStream, int
    *pOffset, const IppVCHuffmanSpec_32s *pDCTable, const IppVCHuffmanSpec_32s
    *pACTable, int *scanMatrix, short quant, Ipp16s *pQuantMatrix, Ipp16s
    *pDstBlock, int *pDstSize);
```

引数

pBitStream	ビット・ストリーム内の現在の位置への二重ポインタ
pOffset	*pBitStream* が指すビットとコードの始点の間のオフセットへのポインタ
pDCTable	DC 係数のコードを格納するテーブルへのポインタ
pACTable	最初の DCT 係数を除くすべての DCT 係数の実行レベル・コードを格納するテーブルへのポインタ
scanMatrix	スキャン・シーケンス内の要素のインデックスを格納する行列へのポインタ
quant	ビット・ストリームから読み出される量子化器のスケール係数
pQuantMatrix	MPEG 規格またはユーザによって定義される重み付け行列へのポインタ
pDstBlock	デコードされた要素へのポインタ
pDstSize	スキャン・シーケンス内の最後の 0 でないブロック係数の位置

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiReconstructDCTBlock_MPEG1_32s は、予測されるブロックと双方向予測されるブロックに適用され、実行レベル・コードのテーブルを使用して、ブロック内のすべての DCT 係数を処理する（DC 係数には *pDCTable*、AC 係数には *pACTable* を使用する）。

この関数は、VCHuffmanDecodeInitAllocRL 関数によって得られるテーブルを使用する。

ポインタ *scanMatrix* は、MPEG 規格に規定されたスキャン行列を指す。図 18-6 に、簡単な行列とシーケンスを示す。デコードの終了後、デコードされたデータはスキャン・シーケンスからリニア・シーケンスに再編成される。次に、逆量子化が実行される。各 DCT 係数は、重み付け行列 *pQuantMatrix* から得られる対応する値で乗算され、ビット・ストリームから読み出される量子化器のスケール係数で乗算される。

この関数は、*pDCTable* の位置にメモリを割り当て、*pRLTable* から得られるデータを使用して実行レベル・コードを格納するデコード・テーブルを初期化し、次に 8 × 8 DCT 係数のブロックをデコードする。DCT 係数を同時に処理することで、パフォーマンスが向上する。処理の結果は *pDstBlock* に送られる。**pDstSize* は、最後の 0 でない係数の位置を指す。デコードの終了後、ビット・ストリーム内のコードへのポインタは、図 18-4 と図 18-5 に示すようにリセットされる。

戻り値

IppStsOk	エラーがないことを示す。
IppStsH263VLCCodeErr	H263 規格に基づくデコード中のエラーを示す。

ReconstructDCTBlockIntra_MPEG1

標準 MPEG1 用の実行レベル・コードのテーブルを使用して 8 × 8 イントラ・ブロックをデコードし、再編成と逆量子化を実行する。

```
IppStatus ippiReconstructDCTBlockIntra_MPEG1_32s (Ipp32u **pBitStream, int
    *pOffset, const IppVCHuffmanSpec_32s *pDCTable, const IppVCHuffmanSpec_32s
    *pACTable, int *scanMatrix, short quant, Ipp16s *pQuantMatrix, short
    *dct_dc_past, Ipp16s *pDstBlock, int *pDstSize);
```

引数

pBitStream	ビット・ストリーム内の現在の位置への二重ポインタ
pOffset	*pBitStream* が指すビットとコードの始点の間のオフセットへのポインタ
pDCTable	DC 係数のコードを格納するテーブルへのポインタ
pACTable	最初の DCT 係数を除くすべての DCT 係数の実行レベル・コードを格納するテーブルへのポインタ
scanMatrix	MPEG 規格またはユーザによって定義されるスキャン行列へのポインタ
quant	ビット・ストリームから読み出される量子化器のスケール係数
pQuantMatrix	MPEG 規格またはユーザによって定義される重み付け行列へのポインタ
dct_dc_past	DC 係数に加算される値へのポインタ。この値は、MPEG 規格によって定義される特殊なテーブルから読み出される。
pDstBlock	デコードされた要素へのポインタ
pDstSize	スキャン・シーケンス内の最後の 0 でないブロック係数の位置

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiReconstructDCTBlockIntra_MPEG1_32s は、イントラ・ブロックに適用される。この関数は、AC 係数用の実行レベル・コードを格納する *pACTable* を使用して、ブロック内のすべての DCT 係数（*pDCTable* を使用して処理される DC 係数を除く）を処理する。

この関数は、VCHuffmanDecodeInitAllocRL 関数によって得られるテーブルを使用する。

ポインタ *scanMatrix* は、MPEG 規格に規定されたスキャン行列を指す。図 18-6 に、簡単な行列とシーケンスを示す。デコードの終了後、デコードされたデータはスキャン・シーケンスからリニア・シーケンスに再編成される。次に、逆量子化が実行される。各 DCT 係数は、重み付け行列 *pQuantMatrix* から得られる対応する値で乗算され、ビット・ストリームから読み出される量子化器のスケール係数で乗算される。

DC 係数には、*dct_dc_past* の値が加算され、$2^{intra\_dc\_shift}$ で乗算される。関数の実行後、*dct_dc_past* には、この引数の初期値と DC 係数の和が格納される。

この関数は、*pDCTable* の位置にメモリを割り当て、*pRLTable* から得られるデータを使用して実行レベル・コードを格納するデコード・テーブルを初期化し、次に 8 × 8 DCT 係数のブロックをデコードする。DCT 係数を同時に処理することで、パフォーマンスが向上する。処理の結果は *pDstBlock* に送られる。**pDstSize* は、最後の 0 でない係数の位置を指す。デコードの終了後、ビット・ストリーム内のコードへのポインタは、図 18-4 と図 18-5 に示すようにリセットされる。

戻り値

IppStsOk	エラーがないことを示す。
IppStsH263VLCCodeErr	H263 規格に基づくデコード中のエラーを示す。

ReconstructDCTBlock_MPEG2

標準 MPEG2 用の実行レベル・コードのテーブルを使用して 8 × 8 非イントラ・ブロックをデコードし、再編成と逆量子化を実行する。

```
IppStatus ippiReconstructDCTBlock_MPEG2_32s (Ipp32u **pBitStream, int
    *pOffset, const IppVCHuffmanSpec_32s *pDCTable, const IppVCHuffmanSpec_32s
    *pACTable, int *scanMatrix, short quant, short *pQuantMatrix, short
    *pDstBlock, int *pDstSize);
```

引数

pBitStream	ビット・ストリーム内の現在の位置への二重ポインタ
pOffset	*pBitStream* が指すビットとコードの始点の間のオフセットへのポインタ
pDCTable	DC 係数（つまり、DCT 係数の中で最初の係数）のコードを格納するテーブルへのポインタ
pACTable	最初の DCT 係数を除くすべての DCT 係数の実行レベル・コードを格納するテーブルへのポインタ
scanMatrix	スキャン・シーケンス内の要素のインデックスを格納する行列へのポインタ
quant	ビット・ストリームから読み出される量子化器のスケール係数
pQuantMatrix	MPEG 規格またはユーザによって定義される重み付け行列へのポインタ
pDstBlock	デコードされた要素へのポインタ
pDstSize	スキャン・シーケンス内の最後の 0 でないブロック係数の位置

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiReconstructDCTBlock_MPEG2_32s は、実行レベル・コードのテーブルを使用して 8 × 8 ブロックをデコードし、ブロックの再編成と逆量子化を実行する。この関数は、予測されるブロックと双方向予測されるブロックに適用され、ブロック内のすべての DCT 係数を処理する。

この関数は、VCHuffmanDecodeInitAllocRL 関数によって得られるテーブルを使用する。

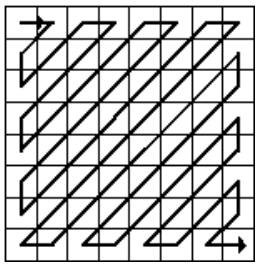
この関数は、8 × 8 DCT 係数のブロックをデコードする。DCT 係数を同時に処理することで、パフォーマンスが向上する。処理の結果は *pDstBlock* に送られる。**pDstSize* は、最後の 0 でない係数の位置を指す。デコードの終了後、ビット・ストリーム内のコードへのポインタは、図 18-4 と図 18-5 に示すようにリセットされる。

ポインタ *scanMatrix* は、ビット・ストリームから読み出されるスキャン行列を指す。図 18-6 に、簡単な行列とシーケンスを示す。デコードの終了後、デコードされたデータはスキャン・シーケンスからリニア・シーケンスに再編成される。

次に、逆量子化が実行される。各 DCT 係数は、重み付け行列 *pQuantMatrix* から得られる対応する値で乗算され、ビット・ストリームから読み出される量子化器のスケール係数で乗算される。

図 18-6　スキャン行列とシーケンス

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63



戻り値

`IppStsOk`	エラーがないことを示す。
`IppStsH263VLCCodeErr`	H263 規格に基づくデコード中のエラーを示す。

ReconstructDCTBlockIntra_MPEG2

標準 MPEG2 用の実行レベル・コードのテーブルを使用して 8 × 8 イントラ・ブロックをデコードし、再編成と逆量子化を実行する。

```
IppStatus ippiReconstructDCTBlockIntra_MPEG2_32s (Ipp32u **pBitStream, int
    *pOffset, const IppVCHuffmanSpec_32s *pDCSizeTable, const
    IppVCHuffmanSpec_32s *pACTable, int *scanMatrix, short quant, short
    *pQuantMatrix, short *dct_dc_past, int intra_dc_shift, short *pDstBlock,
    int *pDstSize);
```

引数

pBitStream	ビット・ストリーム内の現在の位置への二重ポインタ
pOffset	*pBitStream* が指すビットとコードの始点の間のオフセットへのポインタ
pDCSizeTable	DC 係数 （つまり、DCT 係数の中で最初の係数）のコードを格納するテーブルへのポインタ
pACTable	最初の DCT 係数を除くすべての DCT 係数の実行レベル・コードを格納するテーブルへのポインタ
scanMatrix	MPEG 規格またはユーザによって定義されるスキャン行列へのポインタ
quant	ビット・ストリームから読み出される量子化器のスケール係数
pQuantMatrix	MPEG 規格またはユーザによって定義される重み付け行列へのポインタ
dct_dc_past	DC 係数に加算される値へのポインタ。この値は、MPEG 規格によって定義される特殊なテーブルから読み出される。
intra_dc_shift	整数値。DC 係数は $2^{intra\_dc\_shift}$ で乗算される。
pDstBlock	デコードされた要素へのポインタ
pDstSize	スキャン・シーケンス内の最後の 0 でないブロック係数の位置

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiReconstructDCTBlockIntra_MPEG2_32s は、実行レベル・コードのテーブルを使用して 8 × 8 イントラ・ブロックをデコードし、ブロックの再編成と逆量子化を実行する。
この関数はイントラ・ブロックに適用され、ブロック内のすべての DCT 係数（テーブル *pDCSizeTable* を使用して処理される DC 係数を除く）を処理する。

この関数は、VCHuffmanDecodeInitAlloc と VCHuffmanDecodeInitAllocRL 関数によって得られるテーブルを使用する。

この関数は、8 × 8 DCT 係数のブロックをデコードする。DCT 係数を同時に処理することで、パフォーマンスが向上する。処理の結果は *pDstBlock* に送られる。**pDstSize* は、最後の 0 でない係数の位置を指す。デコードの終了後、ビット・ストリーム内のコードへのポインタは、図 18-4 と図 18-5 に示すようにリセットされる。

ポインタ *scanMatrix* は、ビット・ストリームから読み出されるスキャン行列を指す。図 18-6 に、簡単な行列とシーケンスを示す。デコードの終了後、デコードされたデータはスキャン・シーケンスからリニア・シーケンスに再編成される。

次に、逆量子化が実行される。各 DCT 係数は、重み付け行列 *pQuantMatrix* から得られる対応する値で乗算され、ビット・ストリームから読み出される量子化器のスケール係数で乗算される。

戻り値

IppStsOk	エラーがないことを示す。
IppStsH263VLCCodeErr	H263 規格に基づくデコード中のエラーを示す。

メモリ解放

VCHuffmanDecodeFree

VLC テーブルに割り当てられたメモリを解放する。

```
IppStatus ippiVCHuffmanDecodeFree_32s (IppVCHuffmanSpec **pDecodeTable);
```

引数

pDecodeTable　デコード・テーブルへの二重ポインタ

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiVCHuffmanDecodeFree_32s は、VLC テーブルに割り当てられた *pDecodeTable* の位置のメモリを解放する。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

逆量子化

ブロック内のデコードされた各 DCT 係数は、重み付け行列から得られる対応する値とビット・ストリームから読み出される量子化器のスケール係数で乗算すると逆量子化される。この操作を実行する前に、DCT 係数をジグザグ・スキャン・シーケンスからリニア・シーケンスに再編成する必要がある。

イントラ・フレームの場合、MPEG2 規格の規定に従って、ブロック内のすべての DCT 係数（デコーダ内で別に処理される DC 係数を除く）が処理される。非イントラ・フレーム用の関数は、ブロック内のすべての DCT 係数を処理する。

QuantInvIntra_MPEG2

MPEG-2 規格に従って、イントラ・フレームの逆量子化を実行する。

```
IppStatus ippiQuantInvIntra_MPEG2_16s_C1I (Ipp16s *pDctCoeff, int quant,
    Ipp16s *pQuantMatrix);
```

引数

pDctCoeff	ブロックの始点へのポインタ
quant	ビット・ストリームから読み出される量子化器のスケール係数

pQuantMatrix	MPEG 規格またはユーザによって定義される重み付け行列へのポインタ

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiQuantInvIntra_MPEG2_16s_C1I は、*pDctCoeff* から得られる DCT 係数を *quant* と *pQuantMatrix* の対応する値で乗算し、結果を *pDctCoeff* に返す。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNoError	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

QuantInv_MPEG2

MPEG-2 規格に従って、非イントラ・フレームの逆量子化を実行する。

```
IppStatus ippiQuantInv_MPEG2_16s_C1I (Ipp16s *pDctCoeff, int quant, Ipp16s
    *pQuantMatrix);
```

引数

pDctCoeff	ブロックの始点へのポインタ
quant	ビット・ストリームから読み出される量子化器
pQuantMatrix	MPEG 規格またはユーザによって定義される量子化行列へのポインタ

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiQuantIntra_MPEG2_16s_C1I は、*pDctCoeff* から得られる DCT 係数を *quant* と *pQuantMatrix* の対応する値で乗算し、結果を *pDctCoeff* に返す。

Return Values

IppStsOk	エラーがないことを示す。
IppStsNoError	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

QuantInvIntra_MPEG4

MPEG-1 および MPEG-4 規格に従って、イントラ・フレームの逆量子化を実行する。

```
IppStatus ippiQuantInvIntra_MPEG4_16s_C1I (Ipp16s *pDctCoeff, int quant,
    Ipp16s *pQuantMatrix);
```

引数

pDctCoeff	ブロックの始点へのポインタ
quant	ビット・ストリームから読み出される量子化器
pQuantMatrix	MPEG 規格またはユーザによって定義される量子化行列へのポインタ

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiQuantInvIntra_MPEG4_16s_C1I は、*pDctCoeff* から得られる DCT 係数を *quant* と *pQuantMatrix* の対応する値で乗算し、結果を *pDctCoeff* に返す。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNoError	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

QuantInv_MPEG4

MPEG-1 および MPEG-4 規格に従って、非イントラ・フレームの逆量子化を実行する。

```
IppStatus ippiQuantInv_MPEG4_16s_C1I (Ipp16s *pDctCoeff, int quant, Ipp16s
    *pQuantMatrix);
```

引数

pDctCoeff	ブロックの始点へのポインタ
quant	ビット・ストリームから読み出される量子化器
pQuantMatrix	MPEG 規格またはユーザによって定義される量子化行列へのポインタ

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiQuantInv_MPEG4_16s_C1I は、*pDctCoeff* から得られる DCT 係数を *quant* と *pQuantMatrix* の対応する値で乗算し、結果を *pDctCoeff* に返す。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNoError	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

QuantInvIntra_H263

H263 規格に従って、イントラ・フレームの逆量子化を実行する。

```
IppStatus ippiQuantInvIntra_H263_16s_C1I (Ipp16s *pDctCoeff, int quant);
```

引数

pDctCoeff	ブロックの始点へのポインタ

quant ビット・ストリームから読み出される量子化器

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiQuantInvIntra_H263_16s_C1I は、*pDctCoeff* から得られる DCT 係数を *quant* で乗算する。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNoError	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

QuantInv_H263

H263 規格に従って、非イントラ・フレームの逆量子化を実行する。

```
IppStatus ippiQuantInv_H263_16s_C1I (Ipp16s *pDctCoeff, int quant);
```

引数

pDctCoeff	ブロックの始点へのポインタ
quant	ビット・ストリームから読み出される量子化器

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiQuantInv_H263_16s_C1I は、*pDctCoeff* から得られる DCT 係数を *quant* で乗算する。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNoError	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

動き補償

この操作は、非イントラ型の各ブロックに適用される。処理の終了後、データは Ipp16s 型から Ipp8u 型に変換される。

予測されるブロック

MC16x16

予測される 16 × 16 ブロックの動き補償を実行する。

```
IppStatus ippiMC16x16_8u_C1 (const Ipp8u *pSrcRef, Ipp32s srcStep, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    Ipp32s mcType, IppRoundMode roundControl);
```

引数

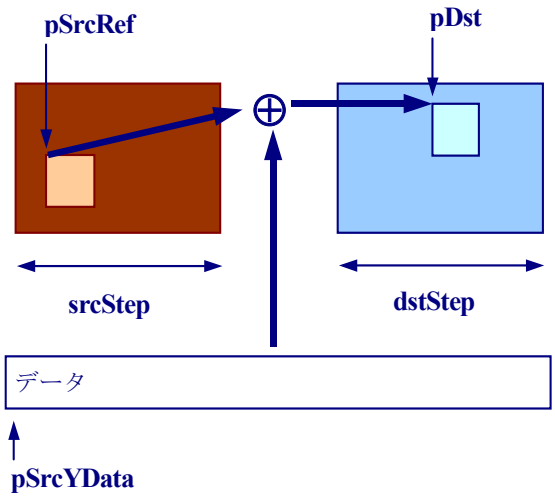
pSrcRef	リファレンス・イントラ・ブロックへのポインタ
srcStep	行のサイズ（バイト単位）、アライメントされたリファレンス・フレームの幅を指定する。
pSrcYData	逆 DCT の実行後に得られるデータへのポインタ
srcYDataStep	バイト数、逆 DCT の実行後に得られるアライメントされたデータの幅を指定する。
pDst	予測されるデスティネーション・ブロックへのポインタ
dstStep	行のサイズ（バイト単位）、アライメントされたデスティネーション・フレームの幅を指定する。
mcType	MC（動き補償）のタイプ IPPVC_MC_APX
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiMC16x16_8u_C1 は、予測される 16 × 16 ブロックの動き補償を実行する。

逆 DCT の実行後に得られるデータの値は、リファレンス・ブロックのデータに加算され、デスティネーション・ブロックに送られる（図 18-7 を参照）。動きベクトルはハーフ・ピクセルの精度でコード化されるため、丸めを実行する必要がある。

図 18-7　予測されるブロックの動き補償方式



戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

MC16x8

予測される 16 × 8 ブロックの動き補償を実行する。

```
IppStatus ippiMC16x8_8u_C1 (const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s
    mcType, IppRoundMode roundControl);
```

引数

pSrcRef	リファレンス・イントラ・ブロックへのポインタ
srcStep	行のサイズ（バイト単位）、アライメントされたリファレンス・フレームの幅を指定する。
pSrcYData	逆 DCT の実行後に得られるデータへのポインタ
srcYDataStep	バイト数、逆 DCT の実行後に得られるアライメントされたデータの幅を指定する。
pDst	予測されるデスティネーション・ブロックへのポインタ
dstStep	行のサイズ（バイト単位）、アライメントされたデスティネーション・フレームの幅を指定する。
mcType	MC（動き補償）のタイプ IPPVC_MC_APX
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiMC16x8_8u_C1 は、予測される 16 × 8 ブロックの動き補償を実行する。

逆 DCT の実行後に得られるデータの値は、リファレンス・ブロックのデータに加算され、デスティネーション・ブロックに送られる（図 18-7 を参照）。動きベクトルはハーフ・ピクセルの精度でコード化されるため、丸めを実行する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

MC8x16

予測される 8 × 16 ブロックの動き補償を実行する。

```
IppStatus ippiMC8x16_8u_C1 (const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s
    mcType, IppRoundMode roundControl);
```

引数

pSrcRef	リファレンス・イントラ・ブロックへのポインタ
srcStep	行のサイズ（バイト単位）、アライメントされたリファレンス・フレームの幅を指定する。
pSrcYData	逆 DCT の実行後に得られるデータへのポインタ
srcYDataStep	バイト数、逆 DCT の実行後に得られるアライメントされたデータの幅を指定する。
pDst	予測されるデスティネーション・ブロックへのポインタ
dstStep	行のサイズ（バイト単位）、アライメントされたデスティネーション・フレームの幅を指定する。
mcType	MC（動き補償）のタイプ IPPVC_MC_APX
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiMC8x16_8u_C1 は、予測される 8 × 16 ブロックの動き補償を実行する。

逆 DCT の実行後に得られるデータの値は、リファレンス・ブロックのデータに加算され、デスティネーション・ブロックに送られる（図 18-7 を参照）。動きベクトルはハーフ・ピクセルの精度でコード化されるため、丸めを実行する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

MC8x8

予測される 8 × 8 ブロックの動き補償を実行する。

```
IppStatus ippiMC8x8_8u_C1 (const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s
    mcType, IppRoundMode roundControl);
```

引数

pSrcRef	リファレンス・イントラ・ブロックへのポインタ
srcStep	行のサイズ（バイト単位）、アライメントされたリファレンス・フレームの幅を指定する。
pSrcYData	逆 DCT の実行後に得られるデータへのポインタ
srcYDataStep	バイト数、逆 DCT の実行後に得られるアライメントされたデータの幅を指定する。
pDst	予測されるデスティネーション・ブロックへのポインタ
dstStep	行のサイズ（バイト単位）、アライメントされたデスティネーション・フレームの幅を指定する。
mcType	MC（動き補償）のタイプ IPPVC_MC_APX
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。関数 ippiMC8x8_8u_C1 は、予測される 8 × 8 ブロックの動き補償を実行する。

逆 DCT の実行後に得られるデータの値は、リファレンス・ブロックのデータに加算され、デスティネーション・ブロックに送られる（図 18-7 を参照）。動きベクトルはハーフ・ピクセルの精度でコード化されるため、丸めを実行する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

MC8x4

予測される 8 × 4 ブロックの動き補償を実行する。

```
IppStatus ippiMC8x4_8u_C1 (const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s
    mcType, IppRoundMode roundControl);
```

引数

pSrcRef	リファレンス・イントラ・ブロックへのポインタ
srcStep	行のサイズ（バイト単位）、アライメントされたリファレンス・フレームの幅を指定する。
pSrcYData	逆 DCT の実行後に得られるデータへのポインタ
srcYDataStep	バイト数、逆 DCT の実行後に得られるアライメントされたデータの幅を指定する。
pDst	予測されるデスティネーション・ブロックへのポインタ
dstStep	行のサイズ（バイト単位）、アライメントされたデスティネーション・フレームの幅を指定する。
mcType	MC（動き補償）のタイプ IPPVC_MC_APX
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。関数 ippiMC8x4_8u_C1 は、予測される 8 × 4 ブロックの動き補償を実行する。

逆 DCT の実行後に得られるデータの値は、リファレンス・ブロックのデータに加算され、デスティネーション・ブロックに送られる（図 18-7 を参照）。動きベクトルはハーフ・ピクセルの精度でコード化されるため、丸めを実行する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

双方向予測されるブロック

MC16x16B

双方向予測されるブロックの動き補償を実行する。

```
IppStatus ippiMC16x16B_8u_C1 (const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    IppRoundMode roundControl);
```

引数

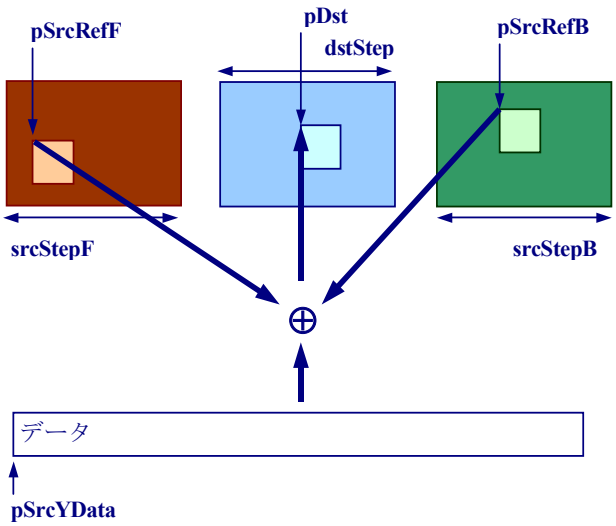
pSrcRefF	順方向リファレンス・ブロックへのポインタ
srcStepF	行のサイズ（バイト単位）、アライメントされた順方向リファレンス・フレームの幅を指定する。
mcTypeF	順方向 MC（動き補償）のタイプ IPPVC_MC_APX
pSrcRefB	逆方向リファレンス・ブロックへのポインタ
srcStepB	行のサイズ（バイト単位）、アライメントされた逆方向リファレンス・フレームの幅を指定する。
mcTypeB	逆方向 MC（動き補償）のタイプ IPPVC_MC_APX
pSrcYData	逆 DCT の実行後に得られるデータへのポインタ
srcYDataStep	バイト数、逆 DCT の実行後に得られるアライメントされたデータの幅を指定する。
pDst	予測されるデスティネーション・ブロックへのポインタ
dstStep	行のサイズ（バイト単位）、アライメントされたデスティネーション・フレームの幅を指定する。
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiMC16x16B_8u_C1 は、双方向予測される 16 × 16 ブロックの動き補償を実行する。

逆 DCT の実行後に得られるデータの値は、リファレンス・ブロックのデータに加算され、デスティネーション・ブロックに送られる（図 18-8 を参照）。動きベクトルはハーフ・ピクセルの精度でコード化されるため、丸めを実行する必要がある。

図 18-8　双方向予測されるブロックの動き補償方式



戻り値

`IppStsOk`	エラーがないことを示す。
`IppStsNullPtrErr`	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

MC16x8B

双方向予測される 16 × 8 ブロックの動き補償を実行する。

```
IppStatus ippiMC16x8B_8u_C1 (const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    IppRoundMode roundControl);
```

引数

pSrcRefF	順方向リファレンス・ブロックへのポインタ
srcStepF	行のサイズ（バイト単位）、アライメントされた順方向リファレンス・フレームの幅を指定する。
mcTypeF	順方向 MC（動き補償）のタイプ IPPVC_MC_APX
pSrcRefB	逆方向リファレンス・ブロックへのポインタ
srcStepB	行のサイズ（バイト単位）、アライメントされた逆方向リファレンス・フレームの幅を指定する。
mcTypeB	逆方向 MC（動き補償）のタイプ IPPVC_MC_APX
pSrcYData	逆 DCT の実行後に得られるデータへのポインタ
srcYDataStep	バイト数、逆 DCT の実行後に得られるアライメントされたデータの幅を指定する。
pDst	予測されるデスティネーション・ブロックへのポインタ
dstStep	行のサイズ（バイト単位）、アライメントされたデスティネーション・フレームの幅を指定する。
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiMC16x8B_8u_C1 は、双方向予測される 16 × 8 ブロックの動き補償を実行する。

逆 DCT の実行後に得られるデータの値は、リファレンス・ブロックのデータに加算され、デスティネーション・ブロックに送られる（図 18-8 を参照）。動きベクトルはハーフ・ピクセルの精度でコード化されるため、丸めを実行する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

MC8x16B

双方向予測される 8 × 16 ブロックの動き補償を実行する。

```
IppStatus ippiMC8x16B_8u_C1 (const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    IppRoundMode roundControl);
```

引数

pSrcRefF	順方向リファレンス・ブロックへのポインタ
srcStepF	行のサイズ（バイト単位）、アライメントされた順方向リファレンス・フレームの幅を指定する。
mcTypeF	順方向 MC（動き補償）のタイプ IPPVC_MC_APX
pSrcRefB	逆方向リファレンス・ブロックへのポインタ
srcStepB	行のサイズ（バイト単位）、アライメントされた逆方向リファレンス・フレームの幅を指定する。
mcTypeB	逆方向 MC（動き補償）のタイプ IPPVC_MC_APX
pSrcYData	逆 DCT の実行後に得られるデータへのポインタ
srcYDataStep	バイト数、逆 DCT の実行後に得られるアライメントされたデータの幅を指定する。
pDst	予測されるデスティネーション・ブロックへのポインタ
dstStep	行のサイズ（バイト単位）、アライメントされたデスティネーション・フレームの幅を指定する。
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiMC8x16B_8u_C1 は、双方向予測される 8 × 16 ブロックの動き補償を実行する。

逆 DCT の実行後に得られるデータの値は、リファレンス・ブロックのデータに加算され、デスティネーション・ブロックに送られる（図 18-8 を参照）。動きベクトルはハーフ・ピクセルの精度でコード化されるため、丸めを実行する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

MC8x8B

双方向予測される 8 × 8 ブロックの動き補償を実行する。

```
IppStatus ippiMC8x8B_8u_C1 (const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    IppRoundMode roundControl);
```

引数

pSrcRefF	順方向リファレンス・ブロックへのポインタ
srcStepF	行のサイズ（バイト単位）、アライメントされた順方向リファレンス・フレームの幅を指定する。
mcTypeF	順方向 MC（動き補償）のタイプ IPPVC_MC_APX
pSrcRefB	逆方向リファレンス・ブロックへのポインタ
srcStepB	行のサイズ（バイト単位）、アライメントされた逆方向リファレンス・フレームの幅を指定する。
mcTypeB	逆方向 MC（動き補償）のタイプ IPPVC_MC_APX
pSrcYData	逆 DCT の実行後に得られるデータへのポインタ
srcYDataStep	バイト数、逆 DCT の実行後に得られるアライメントされたデータの幅を指定する。
pDst	予測されるデスティネーション・ブロックへのポインタ
dstStep	行のサイズ（バイト単位）、アライメントされたデスティネーション・フレームの幅を指定する。
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiMC8x8B_8u_C1 は、双方向予測される 8 × 8 ブロックの動き補償を実行する。

逆 DCT の実行後に得られるデータの値は、リファレンス・ブロックのデータに加算され、デスティネーション・ブロックに送られる（図 18-8 を参照）。動きベクトルはハーフ・ピクセルの精度でコード化されるため、丸めを実行する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

MC8x4B

双方向予測される 8 × 4 ブロックの動き補償を実行する。

```
IppStatus ippiMC8x4B_8u_C1 (const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    IppRoundMode roundControl);
```

引数

pSrcRefF	順方向リファレンス・ブロックへのポインタ
srcStepF	行のサイズ（バイト単位）、アライメントされた順方向リファレンス・フレームの幅を指定する。
mcTypeF	順方向 MC（動き補償）のタイプ IPPVC_MC_APX
pSrcRefB	逆方向リファレンス・ブロックへのポインタ
srcStepB	行のサイズ（バイト単位）、アライメントされた逆方向リファレンス・フレームの幅を指定する。
mcTypeB	逆方向 MC（動き補償）のタイプ IPPVC_MC_APX
pSrcYData	逆 DCT の実行後に得られるデータへのポインタ
srcYDataStep	バイト数、逆 DCT の実行後に得られるアライメントされたデータの幅を指定する。

pDst	予測されるデスティネーション・ブロックへのポインタ
dstStep	行のサイズ（バイト単位）、アライメントされたデスティネーション・フレームの幅を指定する。
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiMC8x4B_8u_C1 は、双方向予測される 8 × 4 ブロックの動き補償を実行する。

逆 DCT の実行後に得られるデータの値は、リファレンス・ブロックのデータに加算され、デスティネーション・ブロックに送られる（図 18-8 を参照）。動きベクトルはハーフ・ピクセルの精度でコード化されるため、丸めを実行する必要がある。

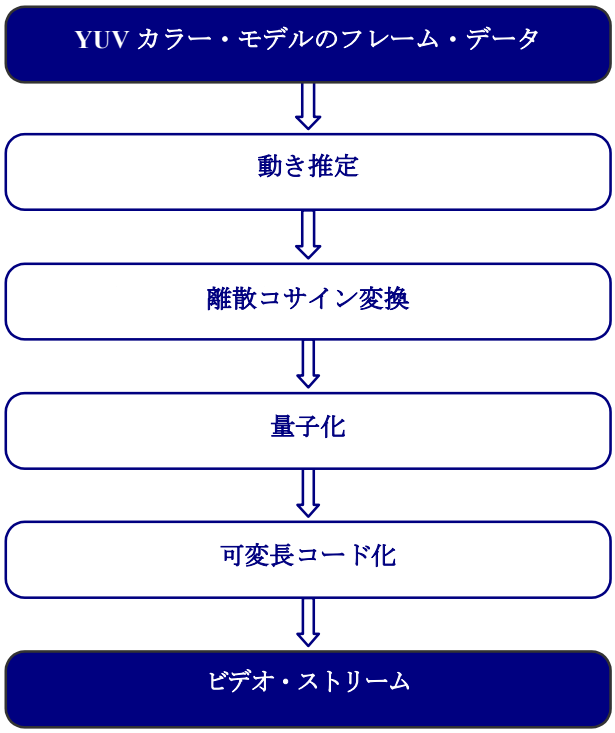
戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

ビデオ・データ・エンコード関数

本節では、図 18-9 に示す標準的なデコード・パイプラインに従って、MPEG ビデオ・エンコードの主な手順について説明する。

図 18-9　エンコード・パイプライン



動き推定と動き補償

エンコードのプロセスでは、最初に動き推定を実行する。予測される各非イントラ・ブロック *A* について、ブロック *B* が見つけられる。ブロック *A* が双方向予測される場合は、2 つのブロック *B* と *C* が見つけられる。これらのリファレンス・ブロックは、ブロック *A* に対して相似でなければならない。

予測されるブロック

GetDiff16x16

16 × 16 要素の現在の予測されるブロック
とリファレンス・ブロックの差を求める。

```
IppStatus ippiGetDiff16x16_8u16s_C1 (const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s
    mcType, Ipp32s roundControl);
```

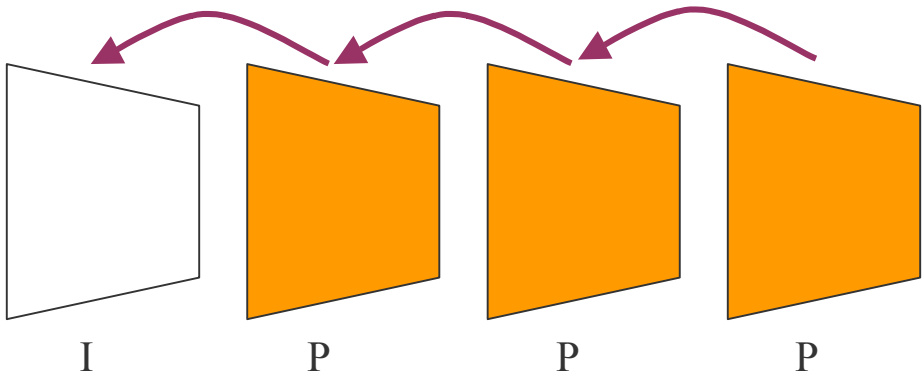
引数

pSrcCur	指定されたサイズの現在のブロックへのポインタ
srcCurStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
pSrcRef	指定されたサイズのリファレンス・ブロックへのポインタ
srcRefStep	リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
pDstDiff	現在のブロックとリファレンス・ブロックの差を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstDiffStep	デスティネーション・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
pDstPredictor	コード化ブロックに関する追加情報を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstPredictorStep	*pDstPredictor* ブロックのステップ（バイト単位）
mcType	次の動き補償のタイプ
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiGetDiff16x16_8u16s_C1 は、指定されたサイズの現在のブロックとリファレンス・ブロックの差を求める。リファレンス・ブロックは、動き補償のタイプに従って、前のブロックの特定のフレームに所属する（図 18-10）。処理の結果は、ブロック *pDstDiff* と *pDstPredictor* に格納される。*pDstPredictor* は、コード化ブロックに関する若干の追加情報を格納する。この情報は、現在のブロックを参照する後続のブロックのエンコードに使用される。この手法は、エンコード・エラーの数を減らすのに有効である。エンコードはハーフ・ピクセルの精度で実行されるため、丸めの種類を指定する必要がある

図 18-10 イントラ・フレームと予測されるフレーム



戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

GetDiff16x8

16 × 8 要素の現在の予測されるブロックと
リファレンス・ブロックの差を求める。

```
IppStatus ippiGetDiff16x8_8u16s_C1 (const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s
    mcType, Ipp32s roundControl);
```

引数

pSrcCur	指定されたサイズの現在のブロックへのポインタ
srcCurStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
pSrcRef	指定されたサイズのリファレンス・ブロックへのポインタ
srcRefStep	リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
pDstDiff	現在のブロックとリファレンス・ブロックの差を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstDiffStep	デスティネーション・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
pDstPredictor	コード化ブロックに関する追加情報を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstPredictorStep	*PDstPredictor* ブロックのステップ（バイト単位）
mcType	次の動き補償のタイプ
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiGetDiff16x8_8u16s_C1 は、指定されたサイズの現在のブロックとリファレンス・ブロックの差を求める。リファレンス・ブロックは、動き補償のタイプに従って、前のブロックの特定のフレームに所属する（図 18-10）。
処理の結果は、ブロック *pDstDiff* と *pDstPredictor* に格納される。*pDstPredictor* は、コード化ブロックに関する若干の追加情報を格納する。この情報は、現在のブロックを参照する後続のブロックのエンコードに使用される。

この手法は、エンコード・エラーの数を減らすのに有効である。エンコードはハーフ・ピクセルの精度で実行されるため、丸めの種類を指定する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

GetDiff8x8

**8 × 8 要素の現在の予測されるブロックと
リファレンス・ブロックの差を求める。**

```
IppStatus ippiGetDiff8x8_8u16s_C1 (const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s
    mcType, Ipp32s roundControl);
```

引数

pSrcCur	指定されたサイズの現在のブロックへのポインタ
srcCurStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
pSrcRef	指定されたサイズのリファレンス・ブロックへのポインタ
srcRefStep	リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
pDstDiff	現在のブロックとリファレンス・ブロックの差を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ

dstDiffStep	デスティネーション・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
pDstPredictor	コード化ブロックに関する追加情報を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstPredictorStep	*PDstPredictor* ブロックのステップ（バイト単位）
mcType	次の動き補償のタイプ
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiGetDiff8x8_8u16s_C1 は、指定されたサイズの現在のブロックとリファレンス・ブロックの差を求める。リファレンス・ブロックは、動き補償のタイプに従って、前のブロックの特定のフレームに所属する（図 18-10）。
処理の結果は、ブロック *pDstDiff* と *pDstPredictor* に格納される。*pDstPredictor* は、コード化ブロックに関する若干の追加情報を格納する。この情報は、現在のブロックを参照する後続のブロックのエンコードに使用される。この手法は、エンコード・エラーの数を減らすのに有効である。エンコードはハーフ・ピクセルの精度で実行されるため、丸めの種類を指定する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

GetDiff8x16

**8 × 16 要素の現在の予測されるブロックと
リファレンス・ブロックの差を求める。**

```
IppStatus ippiGetDiff8x16_8u16s_C1 (const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s
    mcType, Ipp32s roundControl);
```

引数

pSrcCur	指定されたサイズの現在のブロックへのポインタ
srcCurStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
pSrcRef	指定されたサイズのリファレンス・ブロックへのポインタ
srcRefStep	リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
pDstDiff	現在のブロックとリファレンス・ブロックの差を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstDiffStep	デスティネーション・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
pDstPredictor	コード化ブロックに関する追加情報を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstPredictorStep	*PDstPredictor* ブロックのステップ（バイト単位）
mcType	次の動き補償のタイプ
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiGetDiff8x16_8u16s_C1 は、指定されたサイズの現在のブロックとリファレンス・ブロックの差を求める。リファレンス・ブロックは、動き補償のタイプに従って、前のブロックの特定のフレームに所属する（図 18-10）。
処理の結果は、ブロック *pDstDiff* と *pDstPredictor* に格納される。*pDstPredictor* は、コード化ブロックに関する若干の追加情報を格納する。この情報は、現在のブロックを参照する後続のブロックのエンコードに使用される。この手法は、エンコード・エラーの数を減らすのに有効である。エンコードはハーフ・ピクセルの精度で実行されるため、丸めの種類を指定する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

GetDiff8x4

8 × 4 要素の現在の予測されるブロックと
リファレンス・ブロックの差を求める。

```
IppStatus ippiGetDiff8x4_8u16s_C1 (const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s
    mcType, Ipp32s roundControl);
```

引数

pSrcCur	指定されたサイズの現在のブロックへのポインタ
srcCurStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
pSrcRef	指定されたサイズのリファレンス・ブロックへのポインタ
srcRefStep	リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
pDstDiff	現在のブロックとリファレンス・ブロックの差を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstDiffStep	デスティネーション・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
pDstPredictor	コード化ブロックに関する追加情報を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstPredictorStep	*PDstPredictor* ブロックのステップ（バイト単位）
mcType	次の動き補償のタイプ
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiGetDiff8x4_8u16s_C1 は、指定されたサイズの現在のブロックとリファレンス・ブロックの差を求める。リファレンス・ブロックは、動き補償のタイプに従って、前のブロックの特定のフレームに所属する（図 18-10）。

処理の結果は、ブロック *pDstDiff* と *pDstPredictor* に格納される。*pDstPredictor* は、コード化ブロックに関する若干の追加情報を格納する。この情報は、現在のブロックを参照する後続のブロックのエンコードに使用される。この手法は、エンコード・エラーの数を減らすのに有効である。エンコードはハーフ・ピクセルの精度で実行されるため、丸めの種類を指定する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

双方向予測されるブロック

GetDiff16x16B

16 × 16 要素の現在の双方向予測されるブロックと 2 つのリファレンス・ブロックの平均の差を求める。

```
IppStatus ippiGetDiff16x16B_8u16s_C1 (const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
    pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, IppRoundMode roundControl);
```

引数

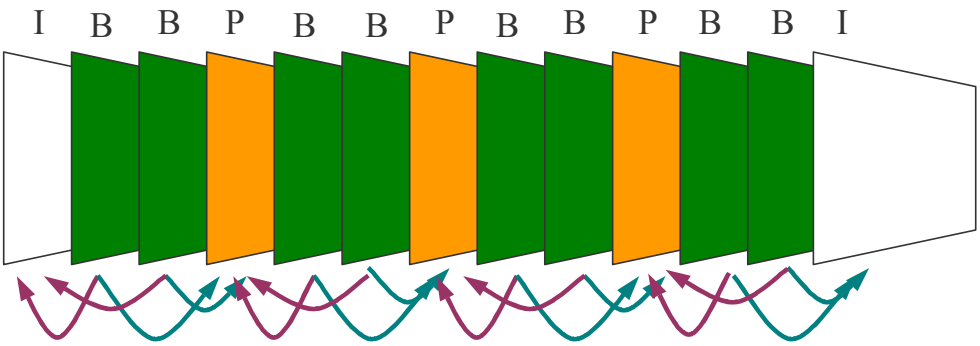
pSrcCur	指定されたサイズの現在のブロックへのポインタ
srcCurStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
pSrcRefF	指定されたサイズの順方向リファレンス・ブロックへのポインタ
srcRefStepF	順方向リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
mcTypeF	次の順方向の動き補償のタイプ
pSrcRefB	指定されたサイズの逆方向リファレンス・ブロックへのポインタ

srcRefStepB	逆方向リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
mcTypeB	次の逆方向の動き補償のタイプ
pDstDiff	現在のブロックとリファレンス・ブロックの差を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstDiffStep	デスティネーション・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiGetDiff16x16B_8u16s_C1 は、指定されたサイズの現在のブロックと 2 つのリファレンス・ブロックの平均の差を求める。リファレンス・ブロックのうち 1 つは順方向リファレンス・ブロックと呼ばれ、動き補償のタイプに従って前のフレームに所属する。もう 1 つのブロックは逆方向リファレンス・ブロックと呼ばれ、以下のフレームのうち 1 つに所属する (図 18-11)。処理の結果はブロック *pDstDiff* に格納される。エンコードはハーフ・ピクセルの精度で実行されるため、丸めの種類を指定する必要がある。

図 18-11 順方向参照と逆方向参照を使用する、イントラ・フレーム、予測されるフレーム、双方向予測されるフレーム



戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

GetDiff16x8B

16 × 8 要素の現在の双方向予測されるブロックと 2 つのリファレンス・ブロックの平均の差を求める。

```
IppStatus ippiGetDiff16x8B_8u16s_C1 (const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
    pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, IppRoundMode roundControl);
```

引数

pSrcCur	指定されたサイズの現在のブロックへのポインタ
srcCurStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
pSrcRefF	指定されたサイズの順方向リファレンス・ブロックへのポインタ
srcRefStepF	順方向リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
mcTypeF	次の順方向の動き補償のタイプ
pSrcRefB	指定されたサイズの逆方向リファレンス・ブロックへのポインタ
srcRefStepB	逆方向リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
mcTypeB	次の逆方向の動き補償のタイプ
pDstDiff	現在のブロックとリファレンス・ブロックの差を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstDiffStep	デスティネーション・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiGetDiff16x8B_8u16s_C1 は、指定されたサイズの現在のブロックと 2 つのリファレンス・ブロックの平均の差を求める。リファレンス・ブロックのうち 1 つは順方向リファレンス・ブロックと呼ばれ、動き補償のタイプに従って前のフレームに所属する。もう 1 つのブロックは逆方向リファレンス・ブロックと呼ばれ、以下のフレームのうち 1 つに所属する（図 18-11）。処理の結果はブロック *pDstDiff* に格納される。エンコードはハーフ・ピクセルの精度で実行されるため、丸めの種類を指定する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

GetDiff8x8B

8 × 8 要素の現在の双方向予測されるブロックと 2 つのリファレンス・ブロックの平均の差を求める。

```
IppStatus ippiGetDiff8x8B_8u16s_C1 (const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
    pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, IppRoundMode roundControl);
```

引数

pSrcCur	指定されたサイズの現在のブロックへのポインタ
srcCurStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
pSrcRefF	指定されたサイズの順方向リファレンス・ブロックへのポインタ
srcRefStepF	順方向リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
mcTypeF	次の順方向の動き補償のタイプ

pSrcRefB	指定されたサイズの逆方向リファレンス・ブロックへのポインタ
srcRefStepB	逆方向リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
mcTypeB	次の逆方向の動き補償のタイプ
pDstDiff	現在のブロックとリファレンス・ブロックの差を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstDiffStep	デスティネーション・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiGetDiff8x8B_8u16s_C1 は、指定されたサイズの現在のブロックと 2 つのリファレンス・ブロックの平均の差を求める。リファレンス・ブロックのうち 1 つは順方向リファレンス・ブロックと呼ばれ、動き補償のタイプに従って前のフレームに所属する。もう 1 つのブロックは逆方向リファレンス・ブロックと呼ばれ、以下のフレームのうち 1 つに所属する（図 18-11）。処理の結果はブロック *pDstDiff* に格納される。エンコードはハーフ・ピクセルの精度で実行されるため、丸めの種類を指定する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

GetDiff8x16B

8 × 16 要素の現在の双方向予測されるブロックと 2 つのリファレンス・ブロックの平均の差を求める。

```
IppStatus ippiGetDiff8x16B_8u16s_C1 (const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
    pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, IppRoundMode roundControl);
```

引数

pSrcCur	指定されたサイズの現在のブロックへのポインタ
srcCurStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
pSrcRefF	指定されたサイズの順方向リファレンス・ブロックへのポインタ
srcRefStepF	順方向リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
mcTypeF	次の順方向の動き補償のタイプ
pSrcRefB	指定されたサイズの逆方向リファレンス・ブロックへのポインタ
srcRefStepB	逆方向リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
mcTypeB	次の逆方向の動き補償のタイプ
pDstDiff	現在のブロックとリファレンス・ブロックの差を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstDiffStep	デスティネーション・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiGetDiff8x16B_8u16s_C1 は、指定されたサイズの現在のブロックと 2 つのリファレンス・ブロックの平均の差を求める。リファレンス・ブロックのうち 1 つは順方向リファレンス・ブロックと呼ばれ、動き補償のタイプに従って前のフレームに所属する。もう 1 つのブロックは逆方向リファレンス・ブロックと呼ばれ、以下のフレームのうち 1 つに所属する（図 18-11）。処理の結果はブロック *pDstDiff* に格納される。エンコードはハーフ・ピクセルの精度で実行されるため、丸めの種類を指定する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

GetDiff8x4B

8 × 4 要素の現在の双方向予測されるブロックと 2 つのリファレンス・ブロックの平均の差を求める。

```
IppStatus ippiGetDiff8x4B_8u16s_C1 (const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
    pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, IppRoundMode roundControl);
```

引数

pSrcCur	指定されたサイズの現在のブロックへのポインタ
srcCurStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
pSrcRefF	指定されたサイズの順方向リファレンス・ブロックへのポインタ
srcRefStepF	順方向リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
mcTypeF	次の順方向の動き補償のタイプ

pSrcRefB	指定されたサイズの逆方向リファレンス・ブロックへのポインタ
srcRefStepB	逆方向リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
mcTypeB	次の逆方向の動き補償のタイプ
pDstDiff	現在のブロックとリファレンス・ブロックの差を格納する、指定されたサイズのデスティネーション・ブロックへのポインタ
dstDiffStep	デスティネーション・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
roundControl	ハーフ・ピクセルの近似に使用する丸めの種類を指定するパラメータ。0 または 1。

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiGetDiff8x4B_8u16s_C1 は、指定されたサイズの現在のブロックと 2 つのリファレンス・ブロックの平均の差を求める。リファレンス・ブロックのうち 1 つは順方向リファレンス・ブロックと呼ばれ、動き補償のタイプに従って前のフレームに所属する。もう 1 つのブロックは逆方向リファレンス・ブロックと呼ばれ、以下のフレームのうち 1 つに所属する（図 18-11）。処理の結果はブロック *pDstDiff* に格納される。エンコードはハーフ・ピクセルの精度で実行されるため、丸めの種類を指定する必要がある。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

SAD16x16

現在のブロックとリファレンス・ブロックの差の絶対値の合計を求める。

```
IppStatus ippiSAD16x16_8u32s (const Ipp8u* pSrcCur, Ipp32s srcCurStep, const
    Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp32s* pDst, Ipp32s mcType);
```

引数

pSrcCur	指定されたサイズの現在のブロックへのポインタ
srcCurStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
pSrcRef	指定されたサイズのリファレンス・ブロックへのポインタ
srcRefStep	リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
pDst	デスティネーション整数へのポインタ
mcType	動き補償のタイプ

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiSAD16x16_8u32s は、現在のブロックのすべての要素とリファレンス・ブロックの対応する要素の差の絶対値の合計を求める。処理の結果は整数 *pDst* に格納される。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。
IppStsStepErr	*srcCurStep* または *srcRefStep* の値が 0 またはそれより小さい場合のエラーを示す。

Variance16x16

現在のブロックの分散を求める。

```
IppStatus ippiVariance16x16_8u32s (const Ipp8u* pSrc, int srcStep, Ipp32s*
    Var);
```

引数

pSrc	指定されたサイズの現在のブロックへのポインタ
srcStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
Var	デスティネーション整数へのポインタ

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiVariance16x16_8u32s は、現在のブロックの分散を求める。結果は整数 *Var* に格納される。

現在のブロックの i^{th} 番目の行と j^{th} 番目の列の交点に位置するピクセルを *block[i,j]* と呼ぶことにする。ここで、次の式が成り立つ。

$$Var = \sum_{i=0}^{15}\sum_{j=0}^{15} block[i,j]^2 - \frac{\left(\sum_{i=0}^{15}\sum_{j=0}^{15} block[i,j]\right)^2}{256}$$

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。
IppStsStepErr	*srcStep* が 0 またはそれより小さい場合のエラーを示す。

SqrDiff16x16

現在のブロックとリファレンス・ブロックの差の 2 乗の合計を求める。

```
IppStatus ippiSqrDiff16x16_8u32s (const Ipp8u* pSrc, int srcStep, const Ipp8u
    pRef, int refStep, Ipp32s mcType, Ipp32s* pSqrDiff);
```

引数

pSrc	指定されたサイズの現在のブロックへのポインタ
srcStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
pRef	指定されたサイズのリファレンス・ブロックへのポインタ
refStep	リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
mcType	動き補償のタイプ
pSqrDiff	デスティネーション整数へのポインタ

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiSqrDiff16x16_8u32s は、現在のブロックのすべての要素とリファレンス・ブロックの対応する要素の差の 2 乗の合計を求める。処理の結果は整数 **pSqrDiff* に格納される。

現在のブロックの i^{th} 番目の行と j^{th} 番目の列の交点に位置するピクセルを *block[i,j]* と呼び、リファレンス・ブロックの i^{th} 番目の行と j^{th} 番目の列の交点に位置するピクセルを *ref_block[i,j]* と呼ぶことにする。ここで、次の式が成り立つ。

$$*pSqrDiff = \sum_{i=0}^{15} \sum_{j=0}^{15} (block[i,j] - ref\_block[i,j])^2$$

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。

IppStsStepErr	*srcStep* または *refStep* が 0 またはそれより小さい場合のエラーを示す。

SqrDiff16x16B

現在の双方向予測されるブロックと 2 つのリファレンス・ブロックの平均の差の 2 乗の合計を求める。

```
IppStatus ippiSqrDiff16x16B_8u32s(const Ipp8u* pSrc, int srcStep, const Ipp8u
    pRefF, int refStepF, Ipp32s mcTypeF, const Ipp8u pRefB, int refStepB,
    Ipp32s mcTypeB, Ipp32s* pSqrDiff);
```

引数

pSrc	指定されたサイズの現在のブロックへのポインタ
srcStep	現在のブロックのステップ、ブロックの幅（バイト単位）を指定する。
pRefF	指定されたサイズの順方向リファレンス・ブロックへのポインタ
refStepF	順方向リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
mcTypeF	順方向リファレンス・ブロックの動き補償のタイプ
pRefB	指定されたサイズの逆方向リファレンス・ブロックへのポインタ
refStepB	逆方向リファレンス・ブロックのステップ、ブロックの幅（バイト単位）を指定する。
mcTypeB	逆方向リファレンス・ブロックの動き補償のタイプ
pSqrDiff	デスティネーション整数へのポインタ

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiSqrDiff16x16B_8u32s は、8 × 8 要素の現在のブロックのすべての要素と 2 つのリファレンス・ブロックの対応する要素の差の 2 乗の合計を求める。処理の結果は整数 *pSqrDiff* に格納される。

現在のブロックの i^{th} 番目の行と j^{th} 番目の列の交点に位置するピクセルを *block[i,j]* と呼び、順方向リファレンス・ブロックの i^{th} 番目の行と j^{th} 番目の列の交点に位置するピクセルを *f_block[i,j]* と呼び、逆方向リファレンス・ブロックの i^{th} 番目の行と j^{th} 番目の列の交点に位置するピクセルを *b_block[i,j]* と呼ぶことにする。次の式が成り立つ。

$$*pSqrDiff = \sum_{i=0}^{15}\sum_{j=0}^{15}\left(block[i,j] - \frac{f\_block[i,j] + b\_block[i,j]}{2}\right)^2$$

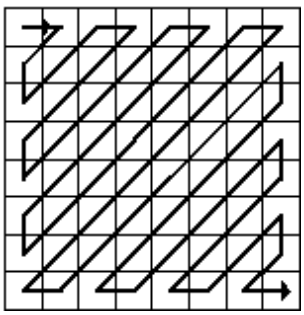
戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。
IppStsStepErr	*srcStep*、*refStepF* または *refStepB* の値が 0 またはそれより小さい場合のエラーを示す。

量子化

ブロック内の DCT 係数は、反転された重み付け行列から得られる対応する値で乗算された後、量子化器のスケール係数で除算され、頻度の低い値を減らせば量子化される。要素がスキャン・シーケンス内の最初の AC 係数から遠くなるほど (図 18-12)、画像の表現とその後のデコードに対するその要素の値の重要性は小さくなる。

図 18-12 スキャン・シーケンス



QuantIntra_MPEG2

MPEG-2 規格に従って、指定された量子化行列を使用してイントラ・ブロックの DCT 係数の量子化をインプレースで実行する。

```
IppStatus ippiQuantIntra_MPEG2_16s_C1I (Ipp16s *pDctCoeff, Ipp32s quant, const
    Ipp32f *pInvQuantMatrix, Ipp32s *Count);
```

引数

pDctCoeff	DCT 係数のブロックへのポインタ
quant	量子化器
pInvQuantMatrix	反転された量子化係数行列へのポインタ
Count	量子化の実行後のスキャン・シーケンス内の最後の 0 でないブロック係数の位置

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。インプレース関数 ippiQuantIntra_MPEG2_16s_C1I は、ブロック内の DCT 係数を反転された量子化行列の要素で乗算し、その値を量子化器で除算する。最後の 0 でない係数の位置は、その後の検討のために *Count* に格納される。ポインタ *pInvQuantMatrix* が NULL の場合は、デフォルトの行列が使用される。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。
IppStsNullPtrErr	0 による除算が発生した場合のエラーを示す。

QuantIntra_MPEG4

MPEG-4 規格に従って、指定された量子化行列を使用してイントラ・ブロックの DCT 係数の量子化をインプレースで実行する。

```
IppStatus ippiQuantIntra_MPEG4_16s_C1I (Ipp16s *pDctCoeff, Ipp32s quant, const
    Ipp32f *pInvQuantMatrix, Ipp32s *Count);
```

引数

pDctCoeff	DCT 係数のブロックへのポインタ
quant	量子化器
pInvQuantMatrix	反転された量子化係数行列へのポインタ
Count	量子化の実行後のスキャン・シーケンス内の最後の 0 でないブロック係数の位置

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。インプレース関数 ippiQuantIntra_MPEG4_16s_C1I は、ブロック内の DCT 係数を反転された量子化行列の要素で乗算し、その値を量子化器で除算する。最後の 0 でない係数の位置は、その後の検討のために *Count* に格納される。ポインタ *pInvQuantMatrix* が NULL の場合は、デフォルトの行列が使用される。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。
IppStsNullPtrErr	0 による除算が発生した場合のエラーを示す。

QuantIntra_H263

H263 規格に従って、指定された量子化行列を使用してイントラ・ブロックの DCT 係数の量子化をインプレースで実行する。

```
IppStatus ippiQuantIntra_H263_16s_C1I (Ipp16s *pDctCoeff, Ipp32s quant, Ipp32s
    *Count);
```

引数

pDctCoeff	DCT 係数のブロックへのポインタ
quant	量子化器
Count	量子化の実行後のスキャン・シーケンス内の最後の 0 でないブロック係数の位置

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。インプレース関数 ippiQuantIntra_H263_16s_C1I は、ブロック内の DCT 係数を反転された量子化行列の要素で乗算し、その値を量子化器で除算する。最後の 0 でない係数の位置は、その後の検討のために *Count* に格納される。ポインタ *pInvQuantMatrix* が NULL の場合は、デフォルトの行列が使用される。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。
IppStsNullPtrErr	0 による除算が発生した場合のエラーを示す。

Quant_MPEG2

MPEG-2 規格に従って、指定された量子化行列を使用して非イントラ・ブロックの DCT 係数の量子化をインプレースで実行する。

```
IppStatus ippiQuant_MPEG2_16s_C1I (Ipp16s *pDctCoeff, Ipp32s quant, const
    Ipp32f *pInvQuantMatrix, Ipp32s *Count);
```

引数

pDctCoeff	DCT 係数のブロックへのポインタ
quant	量子化器
pInvQuantMatrix	反転された量子化係数行列へのポインタ
Count	量子化の実行後のスキャン・シーケンス内の最後の 0 でないブロック係数の位置

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。インプレース関数 ippiQuant_MPEG2_16s_C1I は、ブロック内の DCT 係数を反転された量子化行列の要素で乗算し、その値を量子化器で除算する。最後の 0 でない係数の位置は、その後の検討のために *Count* に格納される。ポインタ *pInvQuantMatrix* が NULL の場合は、デフォルトの行列が使用される。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。
IppStsNullPtrErr	0 による除算が発生した場合のエラーを示す。

Quant_MPEG4

MPEG-4 規格に従って、指定された量子化行列を使用して非イントラ・ブロックの DCT 係数の量子化をインプレースで実行する。

```
IppStatus ippiQuant_MPEG4_16s_C1I (Ipp16s *pDctCoeff, Ipp32s quant, const
    Ipp32f *pInvQuantMatrix, Ipp32s *Count);
```

引数

pDctCoeff	DCT 係数のブロックへのポインタ
quant	量子化器
pInvQuantMatrix	反転された量子化係数行列へのポインタ
Count	量子化の実行後のスキャン・シーケンス内の最後の 0 でないブロック係数の位置

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。インプレース関数 ippiQuant_MPEG4_16s_C1I は、ブロック内の DCT 係数を反転された量子化行列の要素で乗算し、その値を量子化器で除算する。最後の 0 でない係数の位置は、その後の検討のために *Count* に格納される。ポインタ *pInvQuantMatrix* が NULL の場合は、デフォルトの行列が使用される。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。
IppStsNullPtrErr	0 による除算が発生した場合のエラーを示す。

Quant_H263

H263 規格に従って、指定された量子化行列を使用して非イントラ・ブロックの DCT 係数の量子化をインプレースで実行する。

```
IppStatus ippiQuant_H263_16s_C1I (Ipp16s *pDctCoeff, Ipp32s quant, Ipp32s
    *Count);
```

引数

pDctCoeff	DCT 係数のブロックへのポインタ
quant	量子化器
Count	量子化の実行後のスキャン・シーケンス内の最後の 0 でないブロック係数の位置

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。インプレース関数 ippiQuant_H263_16s_C1I は、ブロック内の DCT 係数を反転された量子化行列の要素で乗算し、その値を量子化器で除算する。最後の 0 でない係数の位置は、その後の検討のために *Count* に格納される。ポインタ *pInvQuantMatrix* が NULL の場合は、デフォルトの行列が使用される。

戻り値

IppStsOk	エラーがないことを示す。
IppStsNullPtrErr	少なくとも 1 つの入力ポインタが NULL の場合のエラーを示す。
IppStsNullPtrErr	0 による除算が発生した場合のエラーを示す。

ハフマン・エンコード関数

量子化の実行後、ブロック内の DCT 係数をエンコードする必要がある。エンコードは、標準コード・テーブルを使用した可変長コード化（VLC）によって実現される。標準コード・テーブルは、エントロピーで制約されている。つまり、ダウンロード可能ではなく、限られた範囲のビット・レート向けに最適化されている。

VCL テーブルの作成

EncodeTableInitAlloc

実行レベルのエンコード・テーブルを作成する。

```
IppStatus ippiEncodeTableInitAlloc (const Ipp32s *pSrcTable,
    IppVCHuffmanSpec_32s** pDstSpec);
```

引数

pSrcTable	ソース・テーブルへのポインタ
pDstSpec	デスティネーション・エンコード・テーブルへの二重ポインタ

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiEncodeTableInitAlloc は、実行レベルのエンコード・テーブルを作成する。処理の結果はブロック *pDstSpec* に格納される。

戻り値

IppStsOk	エラーがないことを示す。

ブロック・エンコード関数

PutIntraBlock

イントラ・ブロックのエンコード、再編成、ビット・ストリーム内への配置を実行する。

```
IppStatus ippiPutIntraBlock(Ipp32u** pBitStream, Ipp32s* pOffset, Ipp16s*
    block, Ipp32s* dc_dct_pred, IppVCHuffmanSpec_32s* DC_Tbl,
    IppVCHuffmanSpec_32s* AC_Tbl, Ipp32s* scan, Ipp32s count);
```

引数

pBitStream	ビット・ストリーム内の現在の位置への二重ポインタ
pOffset	*pBitStream* が指すビットとコードの始点の間のオフセットへのポインタ
block	ブロックへのポインタ
dc_dct_pred	DC 係数に加算される値へのポインタ。この値は、MPEG 規格によって定義される特殊なテーブルから読み出される。
DC_Tbl	DC 係数（すなわち、DCT 係数の中で最初の係数）のコードを格納するテーブルへのポインタ
AC_Tbl	AC 係数（すなわち、最初の係数を除くすべての DCT 係数）の実行レベル・コードを格納するテーブルへのポインタ
scan	スキャン行列へのポインタ
count	最後の 0 でない係数の番号

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiPutIntraBlock は、予測されるブロックと双方向予測されるブロックに適用され、8 × 8 DCT 係数のブロックをエンコードする。

ippiPutIntraBlock 関数は、値の計算に **dc_dct_pred* の値を使用し、エンコードに *DC_Tbl* を使用して、DC 係数をエンコードする。次に、この関数は、*AC_Tbl* を使用して AC 係数をエンコードする。

ポインタ *scan* は、MPEG 規格に規定されたスキャン行列を指す。簡単なスキャン行列とシーケンスは、図 18-12 を参照のこと。エンコードの実行後、エンコードされたデータはリニア・シーケンスからスキャン・シーケンスに再編成される。

戻り値

IppStsOk	エラーがないことを示す。

PutNonIntraBlock

非イントラ・ブロックのエンコード、再編成、ビット・ストリーム内への配置を実行する。

```
IppStatus ippiPutNonIntraBlock(Ipp32u** pBitStream, Ipp32s* pOffset, Ipp16s*
    block, IppVCHuffmanSpec_32s* AC_Tbl, Ipp32s* scan, Ipp32s count);
```

引数

pBitStream	ビット・ストリーム内の現在の位置への二重ポインタ
pOffset	*pBitStream* が指すビットとコードの始点の間のオフセットへのポインタ
block	ブロックへのポインタ
AC_Tbl	AC 係数（すなわち、最初の係数を除くすべての DCT 係数）の実行レベル・コードを格納するテーブルへのポインタ
scan	スキャン行列へのポインタ
count	最後の 0 でない係数の番号

説明

この関数は、ippvc.h ヘッダ・ファイルの中で宣言される。
関数 ippiPutNonIntraBlock は、非イントラ・ブロックに適用され、8 × 8 DCT 係数のブロックをエンコードする。この関数は、*AC_Tbl* を使用して、DC 係数と AC 係数をエンコードする。

ポインタ *scan* は、MPEG 規格に規定されたスキャン行列を指す。簡単なスキャン行列とシーケンスは、図 18-12 を参照のこと。エンコードの実行後、エンコードされたデータはリニア・シーケンスからスキャン・シーケンスに再編成される。

戻り値

IppStsOk	エラーがないことを示す。

特殊な事例の処理

IPP に実装されている数値演算関数の中には、すべての引数について定義されていないものがある。ここでは、入力引数が関数定義の範囲を外れているとき、あるいはあいまいな出力結果をまねく可能性があるときに、対応する IPP 画像処理関数がその状況をどのように処理するかについて説明する。

以下の表 18-4 に、各種関数について特殊な事例をまとめ、その関数から返ってくるステータス・コードと結果値の一覧を示す。Err で終わるステータス・コード (ippStsNoErr ステータスを除く) は、エラーを示す。エラーが発生すると、関数の実行は中断される。その他のすべてのステータス・コードは、入力引数が範囲外であることを示す。この場合は、関数の実行は続けられ、ステータス・コードに対応する結果の値が返される。

表 18-4　IPP 画像処理関数についての特殊な事例

関数の基本名	データ・タイプ	特殊な事例	結果値	ステータス・コード
ippiSqrt	16s	Sqrt (x <0)	0	ippStsSqrtNegArg
	32f	Sqrt (x <0)	NAN_32F	ippStsSqrtNegArg
ippiDiv	8u	Div (0/0)	0	ippStsDivByZero
		Div (x/0)	IPP_MAX_8U	ippStsDivByZero
	16s	Div (0/0)	0	ippStsDivByZero
		Div (x/0), x>0	IPP_MAX_16S	ippStsDivByZero
		Div (x/0), x<0	IPP_MIN_16S	ippStsDivByZero
	32f	Div (0/0)	NAN_32F	ippStsDivByZero
		Div (x/0), x>0	INF_32F	ippStsDivByZero
		Div (x/0), x<0	INF_NEG_32F	ippStsDivByZero
				continued
ippiDiv	16sc	Div (0/0)	0	ppStsDivByZero
		Div (x/0),	0	ippStsDivByZero

続く

表 18-4　IPP 画像処理関数についての特殊な事例

関数の基本名	データ・タイプ	特殊な事例	結果値	ステータス・コード
	32sc	Div (0/0)	0	ippStsDivByZero
		Div (x/0), x>0	IPP_MAX_32S	ippStsDivByZero
		Div (x/0), x<0	IPP_MIN_32S	ippStsDivByZero
	32fc	Div (0/0)	NAN_32F	ippStsDivByZero
		Div (x/0), x>0	INF_32F	ippStsDivByZero
		Div (x/0), x<0	INF_NEG_32F	ippStsDivByZero
ippiDivC	all	Div(x/const), const=0	-	ippStsDivByZeroErr
ippiLn	8u	Ln (0)	0	ippStsLnZeroArg
	16s	Ln (0)	0	ippStsLnZeroArg
		Ln (x<0)	0	ippStsLnNegArg
	32f	Ln (x<0)	NAN_32F	ippStsLnNegArg
		Ln(x<IPP_MINABS_32F)	INF_NEG_32F	ippStsLnZeroArg
ippiExp	8u	overflow	IPP_MAX_8U	ippStsNoErr
	16s	overflow	IPP_MAX_16S	ippStsNoErr
	32f	overflow	INF_32F	ippStsNoErr

ここで、x は入力値を表す。それぞれの定数の定義については、第 2 章の「画像のデータ・タイプと範囲」を参照のこと。

別々のデータ・タイプを、同じ数値演算関数のいくつかの変種で処理すると、引数値が同じでも、異なる結果の出る場合がある。ただし、特定の関数と固定データ・タイプについて言えば、特殊な事例の処理は、さまざまな descriptors をその名前に含んでいる関数のどの変種でも同じである。例えば、16s データを操作する対数関数 ippiLn におけるゼロ引数値の扱いは、そのすべての変種 ippiLn_16s_C1RSfs, ippiLn_16s_C3RSfs, ippiLn_16s_C1IRSfs, ippiLn_16s_C3IRSfs で同じである。

参考文献

IPP の画像処理関数を使用する際に参考になる文献の一覧を紹介する。この一覧がすべてではない。さらに詳しく知るための出発点である。[Rogers85]、[Rogers90]、[Foley90] の各文献は、画像処理とコンピュータ・グラフィックスに関する情報が豊富で、さまざまな数式やコード例も収録している。

[Akansu96] A.Akansu 著、M.Smith (編集) 『*Subband and Wavelet transform. Design and Applications*』Kluwer Academic Publishers、1996 年

[AP922] 『*A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Streaming SIMD Extensions and MMX™ Instructions*』Application Note AP922、Intel Corp. 資料番号 742474、1999 年

[APMF] 『*Fast Algorithms for Median Filtering*』Application Note、Intel Corp. 資料番号 79835、2001 年

[Borge86] G. Borgefors 著『*Distance Transformations in Digital Images*』Computer Vision, Graphics, and Image Processing 34、1986 年

[Bragg] Dennis Bragg 著『*A simple color reduction filter*』Graphic Gems III: 20-22

[Canny86] J. Canny 著『*A Computational Approach to Edge Detection*』IEEE Trans. on Pattern Analysis and Machine Intelligence 8(6)、1986 年

[Davis97] J.Davis、Bobick共著『*The Representation and Recognition of Action Using Temporal Templates*』MIT Media Lab Technical Report 402、1997 年

[Davis99] J.Davis、G.Bradski共著『*Real-Time Motion Template Gradients Using Intel® Computer Vision Library*』IEEE ICCV'99 FRAME-RATE WORKSHOP、1999 年

[Feig92] E. Feig、S. Winograd 共著『*Fast Algorithms for the Discrete Cosine Transform*』IEEE Trans. Signal Processing, vol. 40, no. 9, pp.2174 ～ 2193、1992 年 9 月

[Foley90] James D. Foley、Andries van Dam、Steven K. Feiner、John F. Hughes 共著『*Computer Graphics — Principles and Practice*』第 2 版、Addison Wesley、1990 年

[IEEE] 『*IEEE Standard Specifications for the Implementations of 8X8 Inverse Discrete Cosine Transform*』IEEE #1180 (1997 年)

[IPL] 『*Intel Image Processing Library Reference Manual*』Intel Corp. 資料番号 663791

[ISO11172] ISOCD 11172. Information Technology.『*Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s*』(1993 年)

[ISO13818] ISO/IEC 13818. Information Technology.『*Coding of Moving Pictures and Associated*』(1994 年 11 月)

[ISO14496] International Standard ISO/IEC 14496-2. Information Technology. 『*Coding of Audio-Visual Objects - Part 2: Visual*』

[ISO14496A] ISO/IEC 14496-2:1999/Amd.1:2000(E). Information Technology. 『*Coding of Audio-Visual Objects. Part2:Visual. Amendment 1: Visual Extensions*』(2000 年 1 月)

[ISO10918] International Standard ISO/IEC 10918-1『*Digital Compression and Coding of Continuous Tone Still Images,* Appendix A – *Requirements and guidelines*』

[ISO15444] International Standard ISO/IEC 15444-1『*JPEG 2000 Image coding system,* part 1*: Core coding system*』

[ITUH263] ITU-T Recommendation H.263. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS.『*Infrastructure of audiovisual services - Coding of moving video. Video coding for low bit rate communication*』(1998 年 2 月)

[ITU709] ITU-R Recommendation BT.709『*Basic Parameter Values for the HDTV Standard for the Studio and International Programme Exchange*』[旧 CCIR Rec.709] ITU、ジュネーブ、スイス、1990 年

[Jaehne95] Jaehne, Bernd 著『*Digital Image Processing*』第 3 版、Springer-Verlag, Berlin 1995 年

[Jaehne97] Jaehne, Bernd 著『*Practical Handbook on Image Processing for Scientific Applications*』CRC Press、ニューヨーク、1997 年

[Jack01] Jack, Keith 著『*Video Demystified: a Handbook for the Digital Engineer*』LLH Technology Publishing、第 3 版、2001 年

[Myler93] H.Myler, A.Weeks 著『*Computer Imaging Recipes in C*』Prentice Hall、1993 年

[Randy97] Randy Crane 著『*A Simplified Approach to Image Processing*』Prentice Hall PTR、1997 年

[Rao90] K.R. Rao、P. Yip 共著『*Discrete Cosine Transform. Algorithms, Advantages, Applications*』Academic Press, Inc、ロンドン、1990 年

[Ritter96] G.Ritter, J.Wilson 著『*Computer Vision. Algorithms in Image Algebra*』CRC Press、1996 年

[Rogers85] David Rogers 著『*Procedural Elements for Computer Graphics*』McGraw-Hill、1985 年

[Rogers90] David Rogers、J.Alan Adams 共著『*Mathematical Elements for Computer Graphics*』McGraw-Hill、1990 年

[Shanley98] Tom Shanley 著『*Pentium Pro and Pentium II System Architecture*』Addison-Wesley、1998 年

[Schumacher] Dale A. Schumacher 著『*A comparison of digital halftoning techniques*』Graphic Gems III: 57-71

[Serra82] J.Serra 著『*Image Analysis and Mathematical Morphology*』Academic Press、London、1982 年

[Thomas] Spencer W. Thomas、Rod G. Bogart 共著『*Color dithering*』Graphic Gems II: 72-77

[Ulichney93] R.Ulichney 著『*Digital halftoning*』MIT press、1993 年

[Wolberg96] G.Wolberg 著『*Digital Image Warping*』IEEE Computer Society Press、1996 年

用語集

DCT	離散コサイン変換（Discrete Cosine Transform）の略。第 10 章の「Discrete Cosine Transform」を参照のこと。
MMX® テクノロジ	マルチメディアや通信に使用するアプリケーションのパフォーマンス改善を目的としてインテル ® アーキテクチャに加えられた拡張機能の 1 つ。新しいデータ・タイプを 4 つ、64 ビット MMX テクノロジ・レジスタを 8 つ、新しい SIMD 命令を 57 個使用する。
RGB	Red-Green-Blue （赤 - 緑 - 青）の略。赤チャネル、緑チャネル、青チャネルを使用する、3 チャネルのカラー・モデル。
RGBA	Red-Green-Blue-Alpha （赤 - 緑 - 青 - アルファ）の略。赤チャネル、緑チャネル、青チャネル、アルファ・チャネル（透過度チャネル）を使用する、4 チャネルのカラー・モデル。
ROI	「処理対象領域」を参照のこと。
3 チャネル・モデル	3 つのカラー・チャネルを使用するカラー・モデル。RGB カラー・モデルなど。
4 チャネル・モデル	4 つのカラー・チャネルを使用するカラー・モデル。RGBA カラー・モデルなど。
アルファ・チャネル	RGBA モデルなどのカラー・モデルに使用できるカラー・チャネル。「透過度チャネル」とも言う。
インプレース操作	いくつかの入力画像の 1 つが出力画像となるような操作。
カラー・ツイスト・マトリックス	一方のカラー空間に含まれている各ピクセル成分に乗算をすることにより、他方のカラー空間に含まれている各成分を調べるときに使用する。
グレー・スケール・イメージ	輝度チャネルを 1 つ使用して、輝度の違いを白黒の濃淡の違いで表した画像。

行優先順	C 言語で配列を格納するときの標準的な方法。同じ配列に含まれているすべての行がひとつながりに格納されるようなメモリ表現となる。例えば、配列 a[3][4] の場合は、要素 a[1][0] が要素 a[0][3] のすぐあとに続く。
算術演算	画像のピクセル値を対象に行う、加算、減算、乗算、除算、2 乗といった演算。
収縮	各出力ピクセルを、それに対応する入力ピクセルとその 8- 近傍の最大値に設定するモルフォロジー演算。
処理対象領域	演算または処理が行われる矩形画像領域。
ストリーミング SIMD 拡張命令	次世代プロセッサ用としてインテル・アーキテクチャ命令セットに加えられた拡張機能。パックド・データを処理する汎用浮動小数点命令、キャッシャビリティ制御機能のある新たなパックド整数命令、状態管理命令といった各種の命令群を含んでいる。これらの命令を使用することによって浮動小数点データと整数データを大量に処理するアプリケーションのパフォーマンスがかなり改善される。
絶対カラー	カラー空間における各ピクセルの座標により規定されている色。画像処理用のインテル・インテグレーテッド・パフォーマンス・プリミティブでは、絶対カラーの付いた画像を使用する。
単項演算	1 つの入力画像を対象に行う演算。ほかに入力パラメータを指定することも可能。
二項演算	2 つの入力画像を対象に行う演算。ほかに入力パラメータを指定することも可能。
非インプレース操作	入力画像以外の画像が出力になるような操作。「インプレース操作」を参照のこと。
ピクセル指向順序	ピクセルごとのすべてのカラー・チャネルの値がひとかたまりになるような順番（例 : RGBRGB...）で画像情報を格納すること。
ピクセル深度	画像に含まれているピクセルごとに各チャネルの輝度を決定するビット数。
プレーン指向順序	1 つのカラー・チャネルのすべてのデータが、それとは別のチャネルのすべてのデータのあとに続くようなかたち

	（例 : RRRRRGGGGGBBBBB...）で画像情報を格納すること。したがって、チャネルごとに別々の「プレーン」が 1 つずつ出来る。
膨張	各出力ピクセルを、それに対応する入力ピクセルとその 8– 近傍（8 neighbors）の最小値に設定するモルフォロジー演算。
飽和	飽和演算では、数値がそのデータ・タイプとしてのデータ範囲の上限を上回った場合には、上回ったぶんがあふれて飽和し、それ以上大きな数値にはならない。例えば、7FFFh より大きい符号付きワードは 7FFFh に飽和する。数値がデータ範囲の下限を下回った場合には、下回ったぶんがあふれて飽和し、それより小さな数値にはならない。例えば、8000h より小さい符号付きワードは 8000h に飽和する。
モルフォロジー演算	「収縮」「膨張」「収縮 + 膨張」といった演算。
要素単位の演算	1 つのベクトルの各要素を対象に、同じ演算を実行するか、複数のベクトルの同じ位置にある各要素を演算の入力として使用する要素単位の演算。
リニア・フィルタリング	本書では「2D たたみ込み演算」のこと。
リニア画像変換	本書では「離散コサイン変換（DCT）」のこと。

索引

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

あ

い

う

え

お

か

き

け

こ

さ

の

は

ひ

ふ

ほ

ま

め

も

インテル株式会社
〒300-2635 茨城県つくば市東光台 5-6
http://www.intel.co.jp/

2004 年 4 月

A70805-3304J