GIMP and Film Production
v 1.0

http://film.gimp.org

Caroline Dahllöf
caro@rhythm.com

Calvin Williamson
calvinw@mindspring.com

Manish Singh
yosh@gimp.org

Garry Osgood
gosgood@idt.net

September 22, 2000

Contents

Chapter 1

Introduction

This paper discusses GIMP (General Image Manipulation Program) and its possible use in film and production environments. GIMP is a freely available open source paint program (http://www.gimp.org). It can be downloaded with source code, and built and installed on most Unix and Windows machines. It has a standard set of painting and image editing tools, similar to Adobe Photoshop. It supports plug-ins and scripting.

Because the source code is available, GIMP can be customized to fit the needs of production. Experimental versions which support 16-bits per channel have been built and proven useful in production for over a year (see Appendix B).

With the unstable future of many of the current 16-bit off-the-shelf paint packages on the Unix platform, GIMP has a definite advantage for the film industry. It provides an opportunity to build and contribute to a stable and customizable solution for many of the particular needs of production.

This document describes GIMP and gives some details about the upcoming architectural changes for future versions. It also discusses ways interested parties can help contribute to the GIMP community.

The following is an outline of the topics we cover:

- Chapter 2 "GIMP 2.0 Design" describes the basic architecture of GIMP 2.0.
- Chapter 3 "GEGL" reviews the most concretely realized component of GIMP 2.0, the GEGL library. GEGL will be the heart of the GIMP 2.0 image processing engine.
- Chapter 4 "Schedule" is a list of tasks for GIMP 2.0 which describes the main parts of the next major version of gimp. It gives some rough schedule estimates for the tasks.
- Chapter 5, "Conclusion" summarizes our document.

In addition, the appendices cover some useful extra material:

- Appendix A "GIMP Versions, Licenses" covers GIMP versions 0.54 to 1.2 and the experimental HOLLYWOOD branch. It also describes GIMP's license.
- Appendix B "GIMP at Rhythm & Hues Studios" goes into details of the use of the GIMP HOLLYWOOD branch in a production setting.
- Appendix C "Contributing to GIMP" This is a short description of how things get done in open source projects.
- Appendix D "More Information" contains URLs to web pages with more information.

Chapter 2

GIMP 2.0 Design

Though the upcoming 1.2 release of GIMP contains many new features, most development effort since the 1.0 version has involved improved UI and tools (Appendix A), with little improvement in underlying architecture. Internally GIMP is still a monolithic application with a plug-in library wrapped around it. It was never broken into libraries or manageable pieces.

This makes it difficult to add features like color management, CMYK support, different color models and data types. Without breaking GIMP into a series of libraries, moving forward with any major architectural enhancements is impossible. So the plan for GIMP after 1.2 release is to break GIMP into modules, separate the UI from the engine code, and give GIMP the flexibility to handle the next stage of functionality. Unfortunately there is no way to avoid this reworking of GIMP's current architecture.

2.1 GimpCon

In June 2000 the first official GIMP Developers Conference took place in Berlin. At the meeting many of the core developers discussed the future direction of GIMP.

The main topic of the conference was a new architecture for GIMP and a corresponding set of libraries. The remaining sections in this chapter describe the new architecture and design.

2.2 GIMP 2.0 Architecture

GIMP 2.0 will consist of the following libraries:

- *gegl* – Image processing library.
- *libgimpcore* – Core code, includes main GIMP objects and classes.
- *libgimpcoreui* – Private widgets needed by app, not exported to plug-ins.

- *libgimpui* – Widgets available for both app and plug-ins.
- *libgimpwidget* – Utility widgets used in building the other UI libraries.
- *libgimpapp* – Actual GIMP app.
- *libgimpext* – Communication library between plug-ins and application.
- *libgimpiface* – Interfaces for libgimpui and libgimpcore.
- *libgimpjunk* – Various utilities.

2.2.1 GEGL

Pixel manipulation will be done using GEGL (Generic Graphics Library – Chapter 3). GEGL abstracts the color model and color depth (8-bit, 16-bit, float) and provides generic data types for channels or colors. The algorithms implemented using these abstract types are converted to the actual C code using a code-generator. That way it's easier to add additional data types and color models and optimizations.

Chapter 3 has a more detailed description of this library.

2.2.2 GIMP Core and Plug-in Interfaces

Built on top of this, the GIMP core will implement objects like images, tools, layers, channels, brushes etc. The interfaces of these high level objects will be defined in the appropriate library, and implementations of these objects will be provided for use by plug-ins as well. The relevant code to implement the objects for the plug-in library will be auto-generated from the object descriptions. This will give plug-in developers an object-oriented interface instead of the simple procedural approach GIMP uses now.

This architecture also allows reuse of UI-elements from the core UI libraries in plug-ins and provides a clean model-view separation. By exporting the internal object structure to the plug-ins it will become possible to implement operators, tools or complete layer types as plug-ins. For example one could create plug-in vector tools for vector layers in with this design.

2.2.3 Layer Trees

The simple composite of layers into a projection from GIMP 1.x will become with the help of GEGL, a chain (or tree) of operations, allowing things like effect and vector layers to be part of the composite as well.

Each node (comparable to a layer in current gimp) may compute its data internally as pixels, vectors, text, or whatever is appropriate for that node, and just needs to implement a well-defined interface so it can be plugged into the computation chain. In this way anything that does image processing can be used as part of the chain: color corrections, blur filters as well as affine transformations and more complex effects are all viewed in the same way.

2.2.4 Common Object Request Broker Architecture

The communication between the core application and the plug-ins will be done using CORBA. The interface between the actual objects and the ORB will be generated using the object descriptions mentioned above. The developer will only see GTK+-objects and will not have to deal with the shoals of CORBA. GIMP 2.0 will use a network-transparent client-server system.

2.2.5 Multi-threading

Another highlight will be multi-threading of image processing computations. This allows complicated composites and projections of the layers to be computed efficiently. When an update of the projection becomes necessary, the system locks the necessary resources and gives the operation to the batch-renderer to run in its own thread. This way the user interface, and object system stays responsive.

2.2.6 The Application

The GIMP application is a small layer on top of all the libraries and just holds the libraries together. This design will make it possible to implement other applications like video-editors or compositors with the same set of libraries.

Most of this chapter is taken from http://www.gimp.org/gimpcon/review.html.

Chapter 3

GEGL

GEGL (Generic Graphics Library) is a new image processing library to be used by GIMP 2.0. It will support different color models and data types (8bit,16bit,float) in a general way, and allow new models and types to be added easily. It will include support for common image processing operations, and provide a way to create chains of operations and apply these in a memory efficient way. Early versions will include operations needed by GIMP 2.0, and later versions will add a full library of image processing operations and color models. GEGL is available from http://www.gegl.org.

3.1 High Level Design

GEGL is based on the GTK+ object model, an object oriented system that supports inheritance, has an advanced signal and type system, and provides a reference counting scheme for memory management.

The main classes for GEGL are for images, image processing operators, and color models. Operators and images inherit from a common abstract base class, so image operation chains (or trees) can be set up with nodes representing either images or image operations. Color and data conversions between inputs and outputs is handled automatically if desired during computation of a chain.

Each color model implements at least converters to and from an absolute CIE XYZ color model. Specific color models may implement direct conversions between common or closely related color models as well.

Image processing on a chain is done by passing a destination image buffer and region of interest and evaluating the chain to fill in the destination.

3.2 Low Level Design

GEGL will include a comprehensive set of operators for point, area, geometric and statistics operations. For these, different data types (i.e. bit depths) for

channels and color models will be handled through generic programming techniques. This involves using a Generic Image Language (GIL), a simple image manipulation and algorithm description language designed as part of GEGL.

Image algorithms will be written where possible using GIL and particular data type and color model cases will be generated from their GIL description. Though many image processing algorithms are well suited to this style of generic coding, it will be necessary to allow partial specialization techniques where a generic approach is not suitable.

Memory management will include a cache and swapping system. Image data will typically consist of tiles. Multi-threaded computation of operations is planned where possible. For efficiency, region of interests and domains of definitions will be calculated to avoid doing unnecessary computations.

3.3 Current State

GEGL is currently in a development stage, and work is ongoing for most of the major classes.

The abstract base image and operator classes for creating and processing chains of image operations have been written. The Generic Image Language (GIL) has been implemented for the case of point operators using lex/yacc to generate code for some basic common color models (RGB, Gray) and data types (float, 8bit, 16bit). Current work on AreaOps in GIL is beginning. The color model inheritance classes and accompanying data/colors pace converters which provide support for converting color models/data types during image processing has been written as well.

The memory management classes for caching and tiling, as well as efficient computation of chains of operations using regions of interests and domains of definition are beginning design. For details of GEGL progress, see http://www.gegl.org.

Chapter 4

Schedule

The schedule for achieving GIMP 2.0 is broken into three main parts.

The first part involves the development of GEGL. Since GEGL is a low-level library to be used by GIMP, work for it can proceed somewhat independent of GIMP.

Once GEGL is far enough along, a simple skeleton of imaging and painting tools will be developed using GEGL. This stripped down version is called MicroGIMP and will include some simple paint, retouch and editing tools, enough to provide first tests of each of the major areas of functionality for GIMP: painting, editing, compositing, color conversions.

After MicroGIMP is built and tested as a small production tool, the rest of the major GIMP libraries will be built. These libraries include the various GIMP UI libs and the core code for GIMP's graphics objects (Images, Layers, Channels, Drawables). GIMPs task management (rendering pipeline), libraries for plugin communication as well as GIMPs component for scripting (PDB, Procedural Database) are all included here.

Schedule for GIMP 2.0				
Task	Description	Library	Cost	
MicroGIMP				
MicroGimp-skeleton	Create stripped down GIMP with basic simple toolbox ui.	gimpcore	40h	1w
MicroGimp-basic classes	Write initial simple versions of Image, Layer, Drawable classes for stripped down GIMP.	gimpcore	40h	1w
MicroGimp-paint core	Write the initial paint core code for microgimp using GEGL.	gimpcore	80h	2w
MicroGimp-tools	Write first test tools: paint, clone, simple edit tools	gimpcore	120h	3w
		Sub-total	280h	7w
GEGL				
Area Operation	Classes for common area operations in GEGL (convolve, kernel ops)	gegl	120h	3w
Geometric Operations	Classes for common geometric operations such as scaling, transformation.	gegl	120h	3w
GIL Specialization	Mechanism for allowing partial specialization in GIL code	gegl	120h	3w
Point Operations	Classes for LUT operations	gegl	120h	3w
Preprocessor for codegen	Generates colormodel and data type code from GIL specifications	gegl	80h	2w
Memory Management	Cache and virtual memory management classes	gegl	300h	7.5w
Multi-thread support	Multi-threaded support for image operations.	gegl	200h	5w
		Sub-total	1060h	26.5w
GIMP				
Image, Drawable, Layer, Channel	Complete Image, Drawable, Layer, Channel classes. The basic core classes used in GIMP	gimpcore	160h	4w
Widgets UI	Build utility widgets for use in other UI libs	gimpwidgets	200h	5w
Core UI	Private widgets not shared with plugins	gimpcoreui	120h	3w
Gimp UI	Widgets shared with plugins	gimpui	200h	5w
Painting tools	Paint, Airbrush, Erase, Blur etc	gimpcore, gimpui	120h	3w
Editing tools	Rect Select, Free select, Bezier etc	gimpcore, gimpui	120h	3w

Schedule for GIMP 2.0				
Task	Description	Library	Cost	
GIMP(cont)				
Resize tools	Scale, Resize, Transform...	gimpcore, gimpui	80h	2w
Image Adjustment tools	Brightness, Levels, Histogram	gimpcore, gimpui	120h	3w
GIMP Render Pipeline	GIMP task management code to manage calls to gegl, ui tasks, io	gimpcore, gegl	120h	3w
Generic UI	Widgets for multiple data types	gimpwidgets, gimpcoreui, gimpui	120h	3w
Undo	Build undo history mechanism	gimpcore	100h	2.5w
Plugin libraries	Corba based design for communication between plugins and GIMP	gimpext, gimpiface	300h	7.5w
		Sub-total	1760h	44w
		Total	3100h	77.5w

Chapter 5

Conclusion

We think that GIMP 2.0 is the best alternative for solving 16bit paint needs in the film and production world.

5.1 Commercial Paint Packages

Several current 16bit Unix paint packages have uncertain futures. They are costly to install and usually involve working without full control over image data types, and display look tables.

5.2 Call for Participation with GIMP

We hope that you will join us in contributing to the development of GIMP. GIMP is a great example of an open source program whose advancement and progress is clearly beneficial for the graphics community at large. From a very practical point of view, it can be made to solve many of the problems that turn up with paint programs in production settings. It is a solution that can last as well.

Appendix A

GIMP Versions, Licenses

A.1 History of GIMP

Spencer Kimball and Peter Mattis wrote the first version of GIMP, released in 1995 as GIMP 0.54. This original version worked with the Motif UI toolkit, and was later replaced with an open source UI toolkit written specifically for GIMP called the GIMP Toolkit (GTK+, see http://www.gtk.org/).

The next main release was GIMP version 0.99 which came out in 1997. This version included new tile memory management, transparency layers, and a plug-in architecture. In May 1998 version 1.0 of GIMP was released, with the stable version 1.0.4 shortly after that.

The GIMP 1.1 series of developer releases has continued throughout 1999 and 2000 and is currently at version 1.1.25 (Sept. 2000). The current development versions are being debugged and stabilized with the goal of releasing GIMP 1.2 based on these in the very near future.

A.2 GIMP 1.2

This is the next stable version due for release. This version contains many changes since the last stable version, 1.0.4. These include new tools (dodge/burn, smudge, ink), improved progress and status bars, many UI improvements (drag and drop support), better preferences, much improved Bezier path tools, and support for editable brushes. For more details on this version of GIMP see http://sven.gimp.org/1.1/.

A.3 Film Version of GIMP

The film version of GIMP supports both 16-bit and floating point channel data. This version was written as a test for introducing high color resolution support into future versions of GIMP, and to investigate the viability of using GIMP

for film production work. Work on this version was done as a branch of the main GIMP project. This branch is called HOLLYWOOD (the CVS tag for it). HOLLYWOOD is based on the GIMP 1.0.4 code base, and the necessary changes were made to that version to allow it to work for 16-bit and float channels. The HOLLYWOOD branch was updated to work with a more recent version of the GIMP UI toolkit (GTK+1.2) as well. More information about the film version of GIMP is available at http://film.gimp.org.

This film version of GIMP is being used currently at Rhythm and Hues Studios. (see Appendix B).

A.4 GIMP License

GIMP is an open source software project developed for the most part under the GNU Public License (GPL) and the Library GNU Public License (LGPL). The source code for the application itself is licensed under the GPL. The source code for libgimp, the library which communicates with plug-ins and which plug-ins link with, is licensed under the LGPL.

You may download, install, build, make changes to the source code as provided by these licenses (http://www.gnu.org/).

Appendix B

GIMP at Rhythm & Hues Studios

Rhythm & Hues is a character animation and visual effects studio located in Los Angeles. The company's work is featured in recent films like "Babe", "Mouse-Hunt", and "The Flintstones in Viva Rock Vegas". Other work includes commercials for Hankook Tires, Game Boy, and the Coca Cola "Polar Bear" series. The studio has around 400 employees. It uses SGIs for computer graphics work.

The HOLLYWOOD version of GIMP has been used in production at Rhythm & Hues for the past year and a half.

B.1 Why Rhythm & Hues uses GIMP

Rhythm & Hues has a history of writing proprietary software tools for production needs. These were written because commercial versions didn't exist at the time, or available tools could not be altered for production needs. Also it is expensive to install commercial versions of software on large numbers of machines.

Having access to the source code for software has some obvious advantages as well, among them the ability to alter tools for exact needs of production tasks, and to insure that tools work together in a pipeline without data loss.

This is why GIMP was a good choice for Rhythm & Hues. It can be installed, compiled and supported by in-house programmers in much the same way proprietary software can. This is in contrast to commercial paint solutions, which rarely can be configured to match the Rhythm & Hues image data types.

Other big advantages of having source code for GIMP include being able to install and use multiple film display look up tables for viewing film images on monitors. TDs can view images with a variety of these tables depending on the type of job, or film stock.

B.2 How GIMP is used at Rhythm & Hues

GIMP is used at Rhythm & Hues in a number of different ways, from simple retouching and cloning to more specialized fur painting.

Here are some standard uses for GIMP in production

- *Dusting and touching up frames*
- *Texture painting*
- *Painting mattes*
- *Painting maps*
- *Rig and wire removal in scenes where procedural methods are impossible.*

  To make this possible new features were added to GIMP. In this case the ability to flip through frames was added
- *Together with other in-house programs.*

  Rhythm & Hues uses GIMP in conjunction with its in-house fur program and there are plans to integrate GIMP closely with the compositor to make the "talking animal" pipeline even more efficient.

B.3 Some GIMP Problems

After using GIMP in film production for a year and a half, it is clear there are also certain areas where GIMP needs improvement to become a full-featured and mature paint tool for film work:

- *Alpha channel access and editing*

  Direct access to the matte or alpha channel of a RGBA image is very important for film work. It is currently only possible to view and edit the alpha channel of an image in GIMP only indirectly and with some trouble.
- *Color Channel visibility and editing*

  It is difficult to edit single color channels (r,g,b) and cut and paste easily between color channels. Swapping color channels is hard as well.
- *Memory usage*

  Too much memory is used for some simple operations, and this is especially a when large images are involved. Just two 2k layers of images usually becomes difficult to manipulate and use.
- *Speed*

  Some operations are slow, and could use more efficient algorithms.
- *Resolution independence*

  It should be easy to re-use image operations at several resolutions. Spatial parameters should scale correctly to make this work.

B.4 Future of GIMP at Rhythm & Hues

Rhythm & Hues is excited about GIMP's future and will continue to support and contribute to the GIMP community. These contributions are made available to everyone through the normal open source process. Additions, improvements, bug fixes and changes made by Rhythm and Hues are released publicly back to the GIMP project.

It is hoped that the unique opportunity that GIMP provides is clear motivation for other production companies, software developers and users to contribute to and promote this project as well.

Appendix C

Contributing to GIMP

The GIMP project has many developers, all with different interests and located around the world. To work within this setting, new contributors should be willing to participate in the normal development process of the project, and demonstrate their interest and committed to the advancement of the project.

This starts with communication via emails, GIMP news groups, developer lists, documentation, proposals, and IRC as well. GIMP interest groups at conferences and SIGGRAPH are other places where developers and users can meet.

For the most part there is no direct management in the traditional sense for open source projects. No managers tell anyone they must do something. Rather people and interested parties offer to contribute. This means it is up to potential contributors to learn about what is going on by looking at documentation (user and programmer) and studying code and reading emails.

Once developers prove they can contribute (usually by coding things like bug fixes as patches, or writing plug-ins) they graduate to working on libraries and similar things by finding areas that need work or maintenance, and by volunteering to do that work.

Just as GIMP development must address the needs of the software itself it must also improve the development process within the GIMP project. This involves the need for concrete design documents for each module of GIMP, and schedules and task lists that describe ongoing status of the library or module. An example of this can be seen for the GEGL library, and hopefully as other parts of GIMP 2.0 modules are built, they will include similar documentation as well.

Please feel free to contact the authors with any questions or issues and ideas about the above and becoming involved with GIMP and GEGL.

Appendix D

More Information

For more information please look at the following URLs:

- *http://www.gimp.org* This is GIMP's home page.
- *http://www.gegl.org* This is GEGL's home page. It includes more documnetation about its different compoments.
- *http://www.gimp.org/gimpcon* This is the site for the GIMP Conference that has held in Berlin this past June.
- *http://plugins.gimp.org/gimp2* Contains information about new libraries for GIMP 2.0