WGT/2 Reference Manuals

WGT5_WC.LIB

This is the main WGT system library. It provides the user with everything from VGA detection routines to texture-mapped polygons. Every program using WGT must link this library file.

getmemfree

Function: Returns the number of bytes available to the program.

Declaration: int getmemfree (void)

Remarks: Use this function to determine if there is enough free memory for a malloc call or any other dynamic requirements during program execution. Each machine will have a different upper limit based on the amount of memory installed. Values returned may be as high as 2 gigabytes.

Parameters: None

Return Value: The number of bytes free.

Examples: 34, 63

installkbd

Function: Installs a keyboard handler which permits multiple keypresses at the same time.

Declaration: void installkbd (void)

Remarks: Once installed, this keyboard handler will store the status of all keypresses in an array of 128 short integers. The array is called kbdon. Each element of the array is either a 0 or a 1. Each key on the keyboard is assigned a scancode which is represented by the same element in the array. To determine if a key has been pressed, simply check to see if the corresponding element of kbdon is equal to 1.

You must uninstall the keyboard handler before attempting any normal keyboard input routines or before leaving the program. To determine the scancodes of the keys you want, refer to a DOS programmer's manual, or run the included SCANCODE.EXE utility.

A variable called keypressed is increased each time a key is pressed. You may set this variable to 0 and then wait until it reaches 1 (or more) to determine if a key has been pressed. At this point you may check the kbdon array to determine which keys have been pressed. Reset the variable to 0 again after checking the array.

Parameters: None.

Return Value: None.

See also:uninstallkbd

Examples: 56, 57, 58, 59, 62, 63, 65, 67

lib2buf

Function: Allocates memory for a file, and loads it into memory.

Declaration: void *lib2buf (char *filename)

Remarks: This allows any type of file to be loaded into memory from the WGT Data Library file. It gets the required memory first, and then returns a pointer to the data. It can be used for many things, such as text, digital sound files, calculated tables, and any other data you might need for the duration of your program..

What is done with the data loaded is up to you. If the data is text, you will need to process it from the pointer into a usable form.

Parameters: filename - The name of the file inside the WGT Data Library file which will be loaded into memory.

Return Value: A pointer to the data which was loaded.

See Also: setlib, setpassword

Examples: 31

mdeinit

Function: Removes the custom mouse handler.

Declaration: void mdeinit (void)

Remarks: WGT installs a custom mouse handler when the mouse is initialized with minit. This routine MUST be called before your program ends when using the mouse routines. If the custom mouse interrupt is not removed, moving the mouse when inside a rodent-free application will usually lock the computer.

Parameters: None

Return Value: None

See Also: minit

Examples: 12, 13, 14, 16, 18, 19, 20, 22, 23, 25, 30, 34, 36, 37, 39, 40, 47, 60, 61, 64, 66

minit

Function: Initializes the mouse driver.

Declaration: short minit (unsigned short mouseMode)

Remarks: This command initializes the mouse handler in FULLSCREEN OS/2 only!. If you want to use the mouse in DIVE or PM, take a look at example #20 (DIVE version).You must call this first if you want to use the mouse, also YOU MUST CALL THIS BEFORE SETTING THE VIDEO MODE! Unfortunatley the native OS/2 mouse driver does not APPEAR to support VGA mode ?!? However we have provided an example (#23 FullScreen) to show how to make up for it, by using the mouse routines.

Upon initialization, a custom mouse thread is installed. Whenever an action occurs with the mouse, such as it being moved, or a button being depressed, some variables are updated.

The variables mouse.mx, mouse.my, and mouse.but are updated to contain the information about the mouse. mouse.mx and mouse.my are the coordinates on the screen. mouse.mx is in the range 0 to 320, and mouse.my ranges 0 to 200.NOTE: The mouse routines do not work in resolutions higher than 320x200.

If you are using the mouse routines in text mode, they will still report values in the ranges mentioned above. To change the values into character values, you must divide the coordinates by a number. For example, if you are in 80 column mode, the actual character column would be mouse.mx/4, since 320/4 gives you 80 possible values. Also, make a copy of the mouse.mx and mouse.my variables and divide them instead. This is because mouse.mx and mouse.my may change at any time from the custom mouse interrupt.

The mouse.but variable contains information for all of the buttons. Each button is represented by a bit within this number.

Left button: bit 1 (value of 1)
Right button: bit 2 (value of 2)
Middle button bit 3 (value of 4)

Therefore to see if a certain button is depressed,

```
if (mouse.but & 1)  button=LEFT;
if (mouse.but & 2)  button=RIGHT;
if (mouse.but & 4)  button=MIDDLE;
```

You can create defines for each button to make your code easier to understand.

Parameters: None

Return Value: Number of buttons on the mouse if the mouse driver is found.
If the number is less than 1, no mouse driver is present.

See Also: mdeinit, moff, mon, mouseshape, msetbounds, msetspeed, msetthreshhold, noclick

Examples: 12, 13, 14, 16, 18, 19, 20, 22, 23, 25, 30, 34, 36, 37, 39, 47, 60, 61, 64, 66

moff

Function: Turns off the display of the mouse cursor.

Declaration: void moff (void)

Remarks: This routine currently does nothing, unless set by wsetmouseoff.

Parameters: None

Return Value: None

See Also: minit, mon

Examples: FullScreen Example #47

mon

Function: Turn on the display of the mouse cursor.

Declaration: void mon (void)

Remarks: As the mouse driver does not support VGA, This routine currently does nothing, unless set by wsetmouseon

Parameters: None

Return Value: None

See Also: minit, moff

Examples: FullScreen Example #47

msetbounds

Function: Sets the movement boundaries of the mouse cursor.

Declaration: void msetbounds (short x, short y, short x2, short y2)

Remarks: The cursor's movement will be restricted to the specified region after this routine is called. This routine is used to control the user's actions by limiting where they can click on the screen.

Parameters: x - The left X coordinate of the mouse boundary.

y - The top Y coordinate of the mouse boundary.

x2 - The right X coordinate of the mouse boundary.

y2 - The bottom Y coordinate of the mouse boundary.

Return Value: None

Examples: 12, 13, 19, 20, 60, 61, 66

msetxy

Function: Sets the screen location of the mouse cursor.

Declaration: void msetxy (short x, short y)

Remarks: You may set the location of the mouse cursor on the screen with this command. Use this whenever you want to force the user to move to a certain region of the screen.

Parameters: x - Horizontal screen coordinate.

y - Vertical screen coordinate.

Return Value: None

Examples: 37, 60, 61

noclick

Function: Waits until all buttons are released.

Declaration: void noclick (void)

Remarks: This routine waits in a loop until all the mouse buttons are released. It is useful if you want the user to click on a button, perform an action, and wait for another button click. Without this command, the user could hold down the button and the program will think you pressed it a bunch of times. Put this command in after you have found the user pressed a button.

For example:

```
if (mouse.but == 1)
{
  ctr++;              /* Keep track of how many times you press the left button */
  noclick ();         /* Wait until user lets go */
}
```

Parameters: None

Return Value: None

See Also: minit, moff, mon

Examples: 13, 19, 20, 36

setlib

Function: Sets the WGT Data Library name.

Declaration: void setlib (char *libraryname)

Remarks: WGT has the ability to store all of your graphics or data files into one large library file. Use the utility called WGTLIB to manage your WGT Data Library files. These data library files are not the library files made by WLIB, but rather those created by WGTLIB. Set the name of the data library file in the beginning of your program, and all WGT commands which load something will use the library instead of individual files. Pass NULL as libname to disable library files.

Parameters: libraryname - Filename of the WGT Data Library file.

Return Value: None

See Also: lib2buf, setpassword

Examples: 31

setpassword

Function: Sets the WGT Data Library password.

Declaration: void setpassword (char *password)

Remarks: WGT library files can be protected by a password.
If you choose the incorrect password, files cannot be loaded in. This prevents other WGT users from taking your graphics and sound files. Use NULL to indicate no password. If you change your library file partway through the execution of your program and it uses a different password, you should call setpassword immediately after setlib.

Parameters: password - The password used to access the currently open WGT Data Library file.

Return Value: None

See Also: lib2buf, setlib

Examples: 31

uninstallkbd

Function: Removes the WGT custom keyboard handler.

Declaration: void uninstallkbd (void)

Remarks: Any program which calls the installkbd command must at some point call this routine as well. If the program fails to remove the keyboard handler, strange things may happen once the program ends (usually the system locks up).

Parameters: None.

Return Value: None.

See Also: installkbd

Examples: 56, 57, 58, 59, 62, 63, 65, 67

vga256

Function: Initializes WGT/2 and more (SEE BELOW).

Declaration: void vga256 (unsigned char videoBuffer,
short width,short height,unsigned short currentEnv)

Remarks: This command will in the case of a fullscreen OS/2 program, change the video mode into the closest mode selected by width and height on the current machine, which is usually 320x200 pixels with 256 colors. Mode 0x13 is the most popular mode for games because it is very fast and easy to program. vga256 must be called before any other drawing commands.

Upon initialization, vga256 sets a pointer called abuf to the visual screen of the VGA card. abuf will contain a pointer to the current graphics page throughout your program. The pointer returned in videoBuffer will ALWAYS point to video memory. For fullscreen OS/2 programs set currentEnv with WGTDRAWMODE_FSVGA.

In Dive/PM programs YOU pass the pointer to the video buffer where you want the default drawing to go (along with the correct size of that buffer). For currentEnv you would set WGTDRAWMODE_FSVGA

Parameters: None

Return Value: None

See Also: vgadetected, wgetmode, wsetmode

Examples: 1-29

vgadetected

Function: Determines if a VGA card is available.

Declaration: char vgadetected (void)

Remarks: If a VGA card is found, this routine returns 1, otherwise your program should stop immediately since WGT's graphics commands will not function. This is not necessary for DIVE or PM mode programs.

Parameters: None

Return Value: 1 = VGA card found
0 = VGA card not found

See Also: vga256, wgetmode, wsetmode

Examples: 1-29

wallocblock

Function: Allocates enough memory for an image of given dimensions and returns a pointer.

Declaration: block wallocblock (short width, short height)

Remarks: This call is exactly the same as the wnewblock command except for the fact that the memory is left uninitialized. This is useful when you want to create a virtual buffer but do not wish to copy the screen's contents into it yet. It is most commonly used when allocating virtual screens larger than the visual screen.

Parameters: width - Desired image width in pixels

height - Desired image height in pixels

Return Value: Pointer to allocated memory

See also:wnewblock, wsetscreen

Examples: 34, 63

wbar

Function: Draws a filled rectangle.

Declaration: void wbar (short x1, short y1, short x2, short y2)

Remarks: (x1,y1) defines the upper left corner of the bar, and (x2,y2) defines the lower right. The rectangle will be clipped to the clipping boundaries if needed.

Parameters: x1 - The left X coordinate of the bar.

y1 - The top Y coordinate of the bar.

x2 - The right X coordinate of the bar.

y2 - The bottom Y coordinate of the bar.

Return Value: None

See Also: wrectangle

Examples: 4, 18, 27, 29, 42, 54, 55, 60, 61

wbezier

Function: Calculates the points of a bezier curve.

Declaration: void wbezier (tpolypoint *rawpts, short numraw, tpolypoint *curvepts, short numcurve)

Remarks: wbezier takes a list of polygon points and creates a smooth curve between them. It puts the points of the curve into another vertex list that can be used with WGT's polygon routines.

Parameters: rawpts - an array of initial data points.

numraw - contains the number of data points in the rawpts array.

curvepts - the array which contains the points of the bezier curve.

numcurve - the number of points on the curve. Greater values of numcurve will produce smoother bezier curves. numcurve must always be greater than numraw.

Return Value: None

See Also: whollowpoly

Examples: 33

wbutt

Function: Draws a 3-dimensional button.

Declaration: void wbutt (short x1, short y1, short x2, short y2)

Remarks: wbutt draws a 3D button by using different colors for shading around the sides. It is useful for creating user interfaces which often require a screen which is appealing to the eye. This routine uses colors 253 to 255 for shading the button. These colors must be of decreasing intensity for the button to look correct.

Parameters: x1 - The left X coordinate of the button.

y1 - The top Y coordinate of the button.

x2 - The right X coordinate of the button.

y2 - The bottom Y coordinate of the button.

Return Value: None

See Also: wbar, wrectangle

Examples: 4, 13, 20, 58, 59

wchecktimerticks()

Function: Tells you how many ticks with the Hi-Res timer have occured since the last wresetticks(). You need to have winitimer and wstarttimer called before (of course).

Declaration: unsignd long wchecktimerticks()

Remarks: wcircle draws a hollow circle using the given location and radius.

Parameters: x_center - The X coordinate of the center of the circle.

y _center - The Y coordinate of the center of the circle.

radius - The radius of the circle. It must be greater than 0.

Return Value: None

See Also: wellipse, wfill_circle, wfill_ellipse

Examples: 4, 8

wcircle

Function: Draws a circle with a given radius.

Declaration: void wcircle (short x_center, short y_center, short radius)

Remarks: wcircle draws a hollow circle using the given location and radius.

Parameters: x_center - The X coordinate of the center of the circle.

y _center - The Y coordinate of the center of the circle.

radius - The radius of the circle. It must be greater than 0.

Return Value: None

See Also: wellipse, wfill_circle, wfill_ellipse

Examples: 4, 8

wclip

Function: Sets the current clipping area for all graphics pages.

Declaration: void wclip (short x1, short y1, short x2, short y2)

Remarks: (x1,y1) defines the upper left corner of the clipping area, and (x2,y2) defines the lower right. All of WGT's drawing commands will draw within this boundary unless mentioned otherwise. Set the coordinates to the visual screen size to 'disable' clipping.

Parameters: x1 - The left X coordinate of the clipping region.

y1 - The top Y coordinate of the clipping region.

x2 - The right X coordinate of the clipping region.

y2 - The bottom Y coordinate of the clipping region.

Return Value: None

Examples: 4, 8, 11, 12, 17, 32, 61

wcls

Function: Clears the currently selected graphics page with a specified color number.

Declaration: void wcls (unsigned char col)

Remarks: This command fills the current drawing page with the color given. Clipping is not performed.

Parameters: col - The color used to fill the screen.

Return Value: None

See Also: wnormscreen, wsetscreen

Examples: 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 17, 18, 20, 27, 31, 32, 33, 42, 46, 51, 58, 59, 62, 65, 67

wcolrotate

Function: Cycles the colors between two indices in a palette.

Declaration: void wcolrotate (unsigned char start, unsigned char finish, short dir, color *pal)

Remarks: This routine takes the colors from start to finish and shifts them once in the direction specified by dir. This effect can be used to simulate animation.

dir = 0, rotates up.
dir = 1, rotates down.

This function does NOT set the palette. You must call wsetpalette in order to see the changes. This allows you to define multiple rotation areas at a time, by several calls to wcolrotate.

Parameters: start - First palette entry in rotation.

finish - Last palette entry in rotation.

dir - Direction of rotation.

pal - A pointer to the palette

Return Value: None

See Also: wsetpalette, wsetrgb

Examples: 5, 20, 27

wcopyscreen

Function: Copies a section of one block onto a different block.

Declaration: void wcopyscreen (short x1, short y1, short x2, short y2, block source, short destx, short desty, block destination)

Remarks: This command copies a region of one block to a new location. The source and destination blocks may be the same as long as the source and destination region do not overlap. It is commonly used to copy from one page to another.

The area defined by (x1,y1,x2,y2) is copied from the image source, and is placed with the upper left corner at (dx,dy) on the screen referred to by destination. For example, to copy screen second (entirely) to the screen first, type wcopyscreen(0,0,319,199,second,0,0,first).

The visual page can be accessed by replacing the destination or source with NULL.

For example to copy firstscr to the visual page:
```
wcopyscreen (0, 0, 319, 199, firstscr, 0, 0, NULL);
```
or the other way around:
```
wcopyscreen (0, 0, 319, 199, NULL, 0, 0, firstscr);
```

You should also note that in the last example, firstscr must already be allocated from another image function such as wnewblock, or wloadblock. It is not the same as firstscr = wnewblock (0, 0, 319, 199) because wcopyscreen assumes there is already memory allocated for firstscr.

You can also copy a region of one screen to a different location on another. For example,
wcopyscreen(30, 20, 180, 190, first, 60, 90, NULL);

Full clipping is performed, so you do not have to worry about the coordinates being completely within the destination screen. Source and destination images do not required the same dimensions.

Parameters:

x1	- The left X coordinate of the source region.
y1	- The top Y coordinate of the source region.
x2	- The right X coordinate of the source region.
y2	- The bottom Y coordinate of the source region.
source	- A pointer to the source image.
destx	- The X coordinate in the destination region.
desty	- The Y coordinate in the destination region.
destination	- A pointer to the destination image.

Return Value: None

Examples: 16, 18, 19, 20, 23, 25, 29, 42, 54, 55, 60, 61

wdeinitpoly

Function: Deinitializes the WGT polygon system and frees internal buffers.

Declaration: void wdeinitpoly (void)

Remarks: Every program which uses a polygon routine must call this before it exits. Internal buffers are deallocated and memory is returned to the heap space. This call must be matched with a previous call to winitpoly or else the program will crash.

Parameters: None

Return Value: None

See Also: winitpoly

Examples: 42, 51, 54, 55

wdeinit_triangle_renderer

Function: Deinitializes the WGT triangle rendering system and frees internal buffers.

Declaration: void wdeinit_triangle_renderer (void)

Remarks: Every program which uses a triangle routine must call this before it exits. Internal buffers are deallocated and memory is returned to the heap space. This call must be matched with a previous call to winit_triangle_renderer or else the program will crash.

Parameters: None

Return Value: None

See Also: winit_triangle_renderer, wtriangle_flat_shaded_texture, wtriangle_gouraud, wtriangle_gouraud_shaded_texture, wtriangle_solid, wtriangle_texture, wtriangle_translucent_gouraud, wtriangle_translucent_texture

Examples: 3D_CAM

wdissolve

Function: Dissolves a block onto the active display page, with a user-defined pattern.

Declaration: void wdissolve (block sourceimage, short *pattern, short speed)

Remarks: The block described by sourceimage is dissolved onto the active screen using the pattern specified by pattern. Pattern is an array of short integers. It can be defined by the program called dissolve.exe, included with WGT. After saving a pattern with this program, include the code generated into your program, and pass the array to this procedure. The speed is the amount to delay after each pixel is copied.

Parameters: sourceimage - Pointer to image which will replace active screen after dissolve.

pattern - Array of integers used to define fade pattern.

speed - Delay in milliseconds between pixel updates.

Return Value: None

Examples: 17

wdonetimer

Function: Removes the custom timer interrupt.

Declaration: void wdonetimer (void)

Remarks: This command will reset the speed of the system timer to its original speed, and remove the custom timer interrupt.

Parameters: None

Return Value: None

See Also: winittimer, wsettimerspeed, wstarttimer, wstoptimer

Examples: 3, 11, 46, 55, 56, 57, 58, 59, 61, 62, 63, 68

wdraw_scanpoly

Function: Draws a concave polygon which was previously scan converted with the wscan_convertpoly routine.

Declaration: void wdraw_scanpoly (short x, short y, wscanpoly *scanpoly,
void (*customline)(short x1, short x2, short y))

Remarks: The polygon must have been scan converted by the wscan_convertpoly routine before you call this routine.

X and Y are added to each vertex before drawing the polygon, allowing you to move the shape to a different location.

customline is a pointer to a function which will draw a horizontal line. If you set this to NULL, it will use the whline routine. customline accepts three parameters, the first two being the x coordinates of the line. The third is the y coordinate of the horizontal line.

You can write your own horizontal line routines that don't fill the line with a solid color. Some possibilities include plasma, wall papering, or fill patterns.

Parameters: x - Horizontal offset added to each vertex.

y - Vertical offset added to each vertex.

scanpoly - the scan converted polygon structure containing the polygon to draw.

customline - A pointer to a function which draws a horizontal line.

Return Value: None

See Also: wdeinitpoly, wfree_scanpoly, winitpoly, wscan_convertpoly

Examples: 51

wellipse

Function: Draws a hollow ellipse.

Declaration: void wellipse (short x_center, short y_center, short x_radius,
short y_radius)

Remarks: Draws a hollow ellipse with center (x_center,y_center) using radius (x_radius,y_radius). Both x_radius and y_radius must be greater than 0.

Parameters: x_center - Center of ellipse along horizontal axis.

y_center - Center of ellipse along vertical axis.

x_radius - Horizontal radius in pixels.

y_radius - Vertical radius in pixels.

Return Value: None

See Also: wcircle, wfill_circle, wfill_ellipse

Examples: 4

wfade_between

Function: Gradually fades in a group of colors from one palette to another palette using the specified speed.

Declaration: void wfade_between (unsigned char start, unsigned char finish,
short speed, color *pal1, color *pal2)

Remarks: The colors from start to finish are faded from pal1 to pal2. The speed may range from 0 to 32767, and is similar to using the DELAY command between each change of the colors.

Parameters: start - Start index of palette array for fade range.

finish - End index of palette array for fade range.

speed - Delay in milliseconds between color updates.

pal1 - Starting palette.

pal2 - Ending palette.

Return Value: None

See Also: wfade_between_once, wfade_in, wfade_in_once, wfade_out, wfade_out_once

Examples: 49

wfade_between_once

Function: Fades in a group of colors from one palette to another, performing only 1/64th of the fade operation.

Declaration: void wfade_between_once (unsigned char start, unsigned char finish, color *pal1, color *pal2)

Remarks: The colors from start to finish are faded from pal1 to pal2. This routine only performs one step of the fade to allow other operations to be performed while the colors are fading. It must be called at least 64 times to complete the fade between the palettes. The palette is not set by this routine, so you must use wsetpalette to do so. After this is called 64 times, pal1 contains pal2, so be sure to preserve pal1 in another palette array if you don't want to lose it.

Parameters: start - Start index of palette array for fade range.

finish - End index of palette array for fade range.

pal1 - Starting palette.

pal2 - Palette which will result from fade.

Return Value: None

See Also: wfade_between, wfade_in, wfade_in_once, wfade_out, wfade_out_once

Examples: 49

wfade_in

Function: Gradually fades in a group of colors from black to a palette variable using a specified speed.

Declaration: void wfade_in (unsigned char start, unsigned char finish,
short speed, color *pal)

Remarks: The colors from start to finish are faded in starting from black. The speed may range from 0 to 32767, and is similar to using the DELAY command between each change of the colors.

Parameters: start - Start index of palette array for fade range.

finish - End index of palette array for fade range.

speed - Delay in milliseconds between color updates.

pal - Colors will fade from black to the ones contained in this array.

Return Value: None

See Also: wfade_between, wfade_between_once, wfade_in_once, wfade_out, wfade_out_once

Examples: 6, 20

wfade_in_once

Function: Fades in a group of colors from black to a palette variable, performing only 1/64th of the fade operation.

Declaration: void wfade_in_once (unsigned char start, unsigned char finish,
color *pal, color *temppal)

Remarks: The colors from start to finish are faded from black to pal. This routine only performs one step of the fade to allow other operations to be performed while the colors are fading. It must be called at least 64 times to complete the fade. temppal is a temporary palette array which holds the initial values for the fade operation. Usually it would contain a totally black palette. The palette is not set by this routine, so you must use wsetpalette to do so.

Parameters: start - Start index of palette array for fade range.

finish - End index of palette array for fade range.

pal - Colors will fade from black to the ones contained in this array.

temppal - A temporary buffer which is filled with black colors.

Return Value: None

See Also: wfade_between, wfade_between_once, wfade_in, wfade_out, wfade_out_once

Examples: 6

wfade_out

Function: Gradually fades a group of colors from a palette variable to black, using a specified speed.

Declaration: void wfade_out (unsigned char start, unsigned char finish,
short speed, color *pal)

Remarks: The colors from start to finish are faded out to black starting from the palette given. The speed may range from 0 to 32767, and is similar to using the DELAY command between each change of the colors.

Parameters: start - Start index of palette array for fade range.

finish - End index of palette array for fade range.

speed - Delay in milliseconds between color updates.

pal - Palette will fade from these colors to black.

Return Value: None

See Also: wfade_between, wfade_between_once, wfade_in, wfade_in_once, wfade_out_once

Examples: 6, 64

wfade_out_once

Function: Fades out a group of colors from a palette variable to black, performing only 1/64th of the fade operation.

Declaration: void wfade_out_once (unsigned char start, unsigned char finish,
color *pal)

Remarks: The colors from start to finish are faded from pal to black. This routine only performs one step of the fade to allow other operations to be performed while the colors are fading. It must be called at least 64 times to complete the fade. The palette is not set by this routine, so you must use wsetpalette to do so.

Parameters: start - Start index of palette array for fade range.

finish - End index of palette array for fade range.

pal - Colors will fade from this palette to black.

Return Value: None

See Also: wfade_between, wfade_between_once, wfade_in, wfade_in_once, wfade_out

Examples: 6

wfastputpixel

Function: Plots a pixel at the given screen coordinate without clipping.

Declaration: void wfastputpixel (short x, short y)

Remarks: This routine operates the same as wputpixel only it ignores clipping. This is the fastest pixel plotting method.

Parameters: x - Pixel coordinate along horizontal axis.

y - Pixel coordinate along vertical axis.

Return Value: None

See Also: wgetpixel, wputpixel

Examples: 2, 39, 51

wfill_circle

Function: Draws a filled circle given a center and radius.

Declaration: void wfill_circle (short x, short y, short radius)

Remarks: (x,y) is the center of the circle to be drawn. Radius is the size in pixels horizontally. It must be greater than 0.

Parameters: x - Pixel coordinate along horizontal axis.

y - Pixel coordinate along vertical axis.

radius - Radius of circle in pixel units.

Return Value: None

See Also: wcircle, wellipse, wfill_ellipse

Examples: 4, 10, 12, 18, 27

wfill_ellipse

Function: Draws a filled ellipse given a center, a horizontal radius, and a vertical radius.

Declaration: void wfill_ellipse (short x_center, short y_center, short x_radius,
short y_radius)

Remarks: Draws a filled ellipse with center (x_center,y_center) using radius (x_radius,y_radius). Both x_radius and y_radius must be greater than 0.

Parameters: x_center - Center of ellipse on horizontal axis.

y_center - Center of ellipse on vertical axis.

x_radius - Radius of ellipse along horizontal axis in pixels.

y_radius - Radius of ellipse along vertical axis in pixels.

Return Value: None

See Also: wcircle, wellipse, wfill_circle

Examples: 4

wflashcursor

Function: Flash the cursor using the values set by wsetcursor and the global variable curspeed.

Declaration: void wflashcursor (void)

Remarks: This routine uses the cursor coordinates set by the global variables xc and yc. Curspeed is another global variable which can be set to change the flashing speed.

wflashcursor will display the software emulated cursor, delay according to curspeed, erase the cursor, delay again, and then return. It will erase the cursor and return immediately if a key is pressed.

Parameters: None

Return Value: None

See Also: wstring

Examples: 14

wfline

Function: Draws a line between two points, without clipping.

Declaration: void wfline (short x1, short y1, short x2, short y2)

Remarks: This routine is similar to wline, except that no clipping is performed on the line. This results in a slightly faster routine, although care must be taken to ensure that the endpoints are within screen boundaries.

Parameters: x1 - Horizontal coordinate of first endpoint.

y1 - Vertical coordinate of first endpoint.

x2 - Horizontal coordinate of second endpoint.

y2 - Vertical coordinate of second endpoint.

Return Value: None

See Also: wline, wstyleline

Examples: 4, 16, 20, 25

wflipblock

Function: Flips a block in one of two directions. The block is physically changed in memory, not on the screen.

Declaration: void wflipblock (block image, short direction)

Remarks: This procedure will flip a previously stored image either vertically or horizontally. The stored image is physically changed, and therefore the results will not be seen until the block is displayed with wputblock or another image transfer routine.

direction is defined as the following:

vertical = 0
horizontal = 1

Parameters image - Pointer to the image which will be flipped.

direction- Indicates axis along which the flip is performed.

Return Value: None

See Also: wfreeblock, wnewblock, wputblock, wresizeblock

Examples: 10, 19

wfree_scanpoly

Function: Frees memory used by a polygon buffer.

Declaration: void wfree_scanpoly (wscanpoly *scanpoly)

Remarks: When you use wscan_convertpoly, the polygon is stored in a buffer which must be deallocated before you use the buffer again. This routine will free the memory used by one of these scan converted polygons.

Parameters: scanpoly - Buffer in which the polygon was stored, and will now be deallocated.

Return Value: None

See Also: wdeinitpoly, wdraw_scanpoly, winitpoly, wscan_convertpoly

Examples: 51

wfreeblock

Function: Releases memory used to store an image.

Declaration: void wfreeblock (block image)

Remarks: When memory is allocated for a block, it is allocated dynamically using malloc. Most commands that return a block variable allocate this memory for you. To release this memory, use wfreeblock. As with malloc, you cannot use the same pointer twice in a row without freeing the memory.

For example, the following code would produce memory errors:

```
pic = wloadblock ("myblock.blk");
pic = wloadblock ("myblock.blk");
...
wfreeblock (pic);
```

It should read:

```
pic = wloadblock ("myblock.blk");
wfreeblock (pic);
pic = wloadblock ("myblock.blk");
...
wfreeblock (pic);
```

Parameters: image - Pointer to the image which is to be deallocated.

Return Value: None

See Also: wallocblock, wnewblock

Examples: 9, 10, 11, 15, 16, 17, 18, 19, 20, 21, 24, 31, 32, 33, 34, 35, 42, 47, 48, 49, 51, 53, 64, 65, 66

wfreefont

Function: Frees memory previously allocated for a font.

Declaration: void wfreefont (wgtfont font)

Remarks: When a bitmapped font is loaded with wloadfont, memory is allocated dynamically. This is similar in operation to the block commands. Use wfreefont to release the memory for the font.

Parameters: font - Pointer to font buffer which will be deallocated.

Return Value: None

See Also: wloadfont

Examples: 40, 41, 47

wfreesprites

Function: Frees memory from an array of blocks.

Declaration: void wfreesprites (block *image_array, short start, short end)

Remarks: When a sprite file is loaded, several blocks are allocated at once. This routine will free all of the blocks in the array from start to end. It should be noted that this routine will deallocate the images, but will not set the pointers to NULL. You must do this yourself if required.

Example usage:

```
block sprites[1001];
...
wfreesprites (sprites, 0, 1000);
```

Parameters: image_array - Array containing sprite pointers.

start - Start index for block deallocation.

end - End index for block deallocation.

Return Value: None

See Also: wloadsprites

Examples: 19, 20, 22, 23, 25, 29, 31, 45, 56, 58, 59, 62, 63, 65, 66, 67, 68

wget_capslock

Function: Returns the state of the Capslock key.

Declaration: short wget_capslock (void)

Remarks: This routine is used to report the state of the caps lock key. It cannot be used when the custom keyboard interrupt is installed.

Parameters: None

Return Value 0 = Capslock is off
1 = Capslock is on

See Also: wset_capslock

Examples: 44

wget_imagebit

Function: Retrieves a byte of information from the end of allocated memory from an image.

Declaration: unsigned char wget_imagebit (block image)

Remarks: This routine retrieves a value from the last allocated byte of memory for that image. The value returned indicates whether or not the first bit is set. The value indicates the following information:

If the result is 1, the image contains some pixels of color index 0. This means that an XRAY putblock would be effective to make the image see-through for parallax scrolling and other such effects. If the status is 0, a normal putblock would be the fastest way of blasting it to the screen. None of the images contain this status unless either wupdate_imagebytes or wset_imagebyte have been used previously.

Parameters: image - Pointer to the image in memory.

Return Value: Status bit

See Also: wset_imagebyte, wget_imagebyte, wupdate_imagebytes

Examples: None

wget_imagebyte

Function: Retrieves a byte of information from the end of allocated memory from an image.

Declaration: unsigned char wget_imagebyte (block image)

Remarks: This routine retrieves a value from the last allocated byte of memory for that image and it is recommended the the information is used in the following format:

Bit 0 - Indicates Xray or Normal block
Bits 1-7 are unused.

If bit 0 is set (status is 1), the image contains some pixels of color index 0. This means that an XRAY putblock would be effective to make the image see-through for parallax scrolling and other such effects. If the status is 0, a normal putblock would be the fastest way of blasting it to the screen. None of the images contain this byte unless either wupdate_imagebytes or wset_imagebyte have been used previously.

Parameters: image - Pointer to the image in memory.

Return Value: Status byte

See Also: wset_imagebyte, wget_imagebit, wupdate_imagebytes

Examples: None

wget_lalt

Function: Returns the state of the left Alt key.

Declaration: short wget_lalt (void)

Remarks: This routine is used to report the state of the left Alt key. It cannot be used when the custom keyboard interrupt is installed.

Parameters: None

Return Value: 0 = Left Alt is released
1 = Left Alt is depressed

See Also: wget_ralt

Examples: 44

wget_lctrl

Function: Returns the state of the Left Control key.

Declaration: short wget_lctrl (void)

Remarks: This routine is used to report the state of the left control key. It cannot be used when the custom keyboard interrupt is installed.

Parameters: None

Return Value: 0 = Left Control is released
1 = Left Control is depressed

See Also: wget_rctrl

Examples: 44

wget_lshift

Function: Returns the state of the Left Shift key.

Declaration: short wget_lshift (void)

Remarks: This routine is used to report the state of the left shift key. It cannot be used when the custom keyboard interrupt is installed.

Parameters: None

Return Value: 0 = Left Shift is released
1 = Left Shift is depressed

See Also: wget_rshift

Examples: 44

wget_numlock

Function: Returns the state of the Numlock key.

Declaration: short wget_numlock (void)

Remarks: This routine is used to report the state of the numlock key. It cannot be used when the custom keyboard interrupt is installed.

Parameters: None

Return Value: 0 = Numlock is off
1 = Numlock is on

See Also: wset_numlock

Examples: 44

wget_ralt

Function: Returns the state of the Right Alt key.

Declaration: short wget_ralt(void)

Remarks: This routine is used to report the state of the right Alt key. It cannot be used when the custom keyboard interrupt is installed.

Parameters: None

Return Value: 0 = Right Alt is released
1 = Right Alt is depressed

See Also: wget_lalt

Examples: 44

wget_rctrl

Function: Returns the state of the Right Control key.

Declaration: short wget_rctrl (void)

Remarks: This routine is used to report the state of the right control key. It cannot be used when the custom keyboard interrupt is installed.

Parameters: None

Return Value: 0 = Right Control is released
1 = Right Control is depressed

See Also: wget_lctrl

Examples: 44

wget_rshift

Function: Returns the state of the Right Shift key.

Declaration: short wget_rshift (void)

Remarks: This routine is used to report the state of the right shift key. It cannot be used when the WGT keyboard interrupt is installed.

Parameters: None

Return Value: 0 = Right Shift is released
1 = Right Shift is depressed

See Also: wget_lshift

Examples: 44

wget_scrlock

Function: Returns the state of the Scroll Lock key.

Declaration: short wget_scrlock (void)

Remarks: This routine is used to report the state of the scroll lock key. It cannot be used when the custom keyboard interrupt is installed.

Parameters: None

Return Value: 0 = Scroll Lock is off
1 = Scroll Lock is on

See Also: wset_scrlock

Examples: 44

wgetblockheight

Function: Returns height of a previously stored block.

Declaration: short wgetblockheight (block image)

Remarks: This routine returns the height of the block in pixels. If the block has not been allocated, the result is undefined.

Parameters: image - Pointer to image which will be measured.

Return Value: Height of the block, in pixels.

See Also: wgetblockwidth, wnewblock

Examples: 11, 19, 57, 64, 66, 67

wgetblockwidth

Function: Returns width of a previously stored block.

Declaration: short wgetblockwidth (block image)

Remarks: This function returns the width of the block in pixels. If the block has not been allocated, the result is undefined.

Parameters: image - Pointer to image which will be measured.

Return Value: Width of the block, in pixels.

See Also: wgetblockheight, wnewblock

Examples: 11, 19, 29, 57, 64, 66, 67

wgetmode

Function: Returns the current video mode number.

Declaration: short wgetmode (void)

Remarks: This is used to store the current video state before changing into graphics mode. You can use wsetmode to restore the mode before ending your program.

Parameters: None

Return Value: The mode number of the current video mode.

See Also: vga256, vgadetected, wsetmode

Examples: 1-29

wgetpixel

Function: Gets the pixel value at the given screen coordinate.

Declaration: unsigned char wgetpixel (short x, short y)

Remarks: This reads the color of a pixel from the current page and returns the color value at the coordinate given.

Parameters: x - Horizontal coordinate of pixel to check.

y - Vertical coordinate of pixel to check.

Return Value: Color value of the pixel, between 0 and 255.

See Also: wfastputpixel, wputpixel

Examples: 2, 60

wgettextheight

Function: Returns the maximum height of the string.

Declaration: short wgettextheight (char *string, wgtfont font)

Remarks: This function is used for justification of strings. If font is NULL, it always returns 8, which is the height of the default font. If font is a pointer to a bitmapped font, the height of the tallest character in the string is returned.

Parameters: string - The string to measure.

font - Pointer to the font used in measurement.

Return Value: The height in pixels of the tallest letter in the string.

See Also: wgettextwidth

Examples: 41

wgettextwidth

Function: Returns the width of the string.

Declaration: short wgettextwidth(char *string, wgtfont font)

Remarks: This function is used for justification of strings. If font is NULL, the width is 8 for each character. If font is a pointer to a bitmapped font, the width of each character may vary.

Parameters: string - The string to be measured.

font - Pointer to the font used in measurement.

Return Value: The total width in pixels of the string.

See Also: wgettextheight

Examples: 41

wgouraudpoly

Function: Draws a Gouraud shaded convex polygon.

Declaration: void wgouraudpoly (tpolypoint *vertexlist, short numvertex,
short x, short y)

Remarks: The wgouraudpoly routine draws a Gouraud shaded polygon by shading the polygon between the colors at each vertex. Each vertex is has the offset (x,y) added to it before the polygon is drawn. vertexlist is an array of vertices, which contains numvertex vertices.

The tpolypoint structure is defined as:

```
typedef struct
{
  short x, y;       /* Coordinate on the screen */
  short sx, sy;     /* Coordinate on the texture */
} tpolypoint;
```

wgouraudpoly uses the sx variable to store the color of the vertex. This means each vertex has a color value (0-255). For Gouraud shading, the polygon uses color numbers between the ones stored at the vertices. For example, if one vertex is color 5, and another is color 30, the polygon will use colors 5-30. Therefore in order to make a shaded polygon, those color values must be smoothly blended from one to the other. You can use the blend function in the WGT Sprite Editor to create a blended range of colors.

Parameters: vertexlist - Array of vertices defining the polygon.

numvertex - Total number of vertices in the array.

x - Horizontal offset added to each vertex.

y - Vertical offset added to each vertex.

Return Value: None

See Also: wdeinitpoly, whollowpoly, winitpoly, wsolidpoly, wtexturedpoly

Examples: 42, 54, 55

wgtprintf

Function: Prints text using the standard printf format.

Declaration: void wgtprintf (short x, short y, wgtfont font, char *fmt, ...)

Remarks: Allows the user to print to the screen using the same format as the printf command from ANSI C. The text is printed at the screen coordinates (x,y) and uses the font. The format is the same as the printf command, so you can easily print numbers within strings very easily.

Example:
wgtprintf (30, 40, bigfont, "Hello %s! You are %hi years old.", name, years);

If font is NULL, it uses the default 8x8 font.

Note: Special characters such as newline, bell, or carriage return are NOT interpreted. wgtprintf will display the character used by these codes.

Parameters: x - Horizontal location for string display.

y - Vertical location for string display.

font - Pointer to font which will be used.

fmt - String for formatted output (see printf).

... - Variables used for formatted string output.

Return Value: None

See Also: wouttextxy

Examples: 12, 20, 26, 34, 35, 41, 49, 51, 58, 59, 63, 67

whline

Function: Draws a horizontal line between two points.

Declaration: void whline (short x1, short x2, short y)

Remarks: This routine will draw horizontal lines faster than the normal wline routine. Be careful when passing parameters because the two x coordinates are passed before the single y coordinate (different format than wline).

Parameters: x1 - Horizontal coordinate of first endpoint.

x2 - Horizontal coordinate of second endpoint.

y - Vertical coordinate for the whole line.

Return Value: None

See Also: wfline, wline, wstyleline

Examples: 46

whollowpoly

Function: Draws a hollow polygon.

Declaration: void whollowpoly (tpolypoint *vertexlist, short numvertex,
short x, short y, short closemode)

Remarks: The whollowpoly routine draws a hollow polygon using the current color. Each vertex is has the offset (x,y) added to it before the polygon is drawn.
vertexlist is an array of vertices, which contains numvertex vertices. closemode tells whether the polygon is open or closed. If closemode is 1, the first and last points are connected.

The tpolypoint structure is defined as:

```
typedef struct
{
  short x, y;        /* Coordinate on the screen */
  short sx, sy;      /* Coordinate on the texture */
} tpolypoint;
```

whollowpoly does not use the sx,sy variables. While this is wasteful of memory, it allows you to switch between polygon rendering methods without altering your vertex data variables.

Parameters: vertexlist - Array of vertices defining the polygon.

numvertex - Total number of vertices in the array.

x - Horizontal offset added to each vertex.

y - Vertical offset added to each vertex.

closemode - Indicates if the polygon is open or closed.

Return Value: None

See Also: wgouraudpoly, wsolidpoly, wtexturedpoly

Examples: 33, 42, 54, 55

winitpoly

Function: Initializes the WGT polygon system to a maximum number of scanlines.

Declaration: void winitpoly (short maxrows)

Remarks: Before performing any polygon routines this function must be called to initialize the internal buffers. Simply pass the maximum number of scanlines which will be used when manipulating polygons. Usually this number refers the the screen dimension for height (WGT_SYS.yres).

A call to wdeinitpoly must be made before the program exits in order to free the buffers used by the polygon system. This should also be called if you want to re-initialize the system to a different number of scanlines at some point.

Parameters: maxrows - The maximum number of scanlines which will be used for any polygon routines.

Return Value: None

See Also: wdeinitpoly

Examples: 42, 54, 51, 55

winittimer

Function:	Installs the custom timer
Declaration:	void winittimer (void)
Remarks:	You must have IBM's Hi-Resolution Timer (Standard on OS/2 4.x+) for these routines to work.
Parameters:	None
Return Value:	None
See Also:	wdonetimer, wsettimerspeed, wstarttimer, wstoptimer
Examples:	3, 11, 46, 55, 56, 57, 58, 59, 61, 62, 63, 68

winit_triangle_renderer

Function: Initializes the WGT triangle rendering system to a maximum number of scanlines.

Declaration: void winit_triangle_renderer (short maxrows)

Remarks: Before performing any triangle routines this function must be called to initialize the internal buffers. Simply pass the maximum number of scanlines which will be used when manipulating triangles. Usually this number refers the the screen dimension for height (WGT_SYS.yres).

A call to wdeinit_triangle_renderer must be made before the program exits in order to free the buffers used by the polygon system. This should also be called if you want to re-initialize the system to a different number of scanlines at some point.

Parameters: maxrows - The maximum number of scanlines which will be used for any triangle routines.

Return Value: None

See Also: wdeinit_triangle_renderer, wtriangle_flat_shaded_texture, wtriangle_gouraud, wtriangle_gouraud_shaded_texture, wtriangle_solid, wtriangle_texture, wtriangle_translucent_gouraud, wtriangle_translucent_texture

Examples: 3D_CAM

whline

Function: Draws a line between two points.

Declaration: void wline (short x1, short y1, short x2, short y2)

Remarks: (x1,y1) defines the first endpoint of the line. (x2,y2) defines the second endpoint of the line. The line is clipped if needed.

Parameters: x1 - Horizontal coordinate of first endpoint.

y1 - Vertical coordinate of first endpoint.

x2 - Horizontal coordinate of second endpoint.

y2 - Vertical coordinate of second endpoint.

Return Value: None

See Also: wfline, wstyleline

Examples: 1, 4, 5, 6, 9, 10, 12, 14, 15, 17, 18, 19, 23, 28, 29, 36, 47, 60, 67

wloadblock

Function: Loads a block file from disk into memory.

Declaration: block wloadblock (char *filename)

Remarks: wloadblock automatically allocates the appropriate amount of memory, by calling malloc, and loads the data into the space allocated. It then returns a pointer to this memory so you may perform other actions on the block. You do should NOT call wnewblock before you load the block. wnewblock is used for reading a block off the current video page only.

Block files are the simplest of graphics file formats. A block has the following format:

width : short integer
height : short integer
data : width*height bytes (no compression)

No palette data is stored in a block. You must load palettes separately with wloadpalette if you use block files.

Parameters: filename - Full pathname of image to be loaded.

Return Value: A pointer to the block loaded. If the block could not be found on disk, it returns NULL.

See Also: wsaveblock

Examples: 35, 47

wloadbmp

Function: Loads a BMP image and converts it into block format.

Declaration: block wloadbmp (char *filename, color *pal)

Remarks: Given a filename, this function loads the BMP image and converts it to block format (image may be mono, 2/4/8 or 24-bit color). 24-bit color images are converted to greyscale.

Parameters: filename - Full pathname of image to be loaded.

pal - Color array for image palette.

Return Value: Pointer to the loaded BMP, in block format. If the BMP could not be found on disk, it returns NULL.

See Also: wsavebmp

Examples: 31, 35

wloadcel

Function: Loads a CEL file (AutoDesk Animator format) from disk into memory.

Declaration: block wloadcel (char *filename, color *palette)

Remarks: CEL files are created with AutoDesk Animator. The newer version, Animator Pro, is not compatible with WGT's CEL format. You should use the POCO program oldcel included with Animator Pro to save the old format supported by WGT.

wloadcel automatically allocates the appropriate amount of memory by calling malloc, and loads the data into the space allocated. It then returns a pointer to this memory so you may perform other actions on the block. You should NOT call wnewblock before you load the block.

Parameters: filename - Full pathname of image to be loaded.

palette - Color array for image palette.

Return Value: Pointer to the loaded cel, in block format. If the CEL could not be found on disk, it returns NULL.

See Also: wsavecel

Examples: 35, 42

wloadfont

Function: Loads a bitmapped font from disk and returns a pointer to it.

Declaration: wgtfont wloadfont (char *filename)

Remarks: Fonts can be used with wouttextxy, wgtprintf, or wstring. Custom fonts are created with the WGT Sprite Editor.

Example:
myfont = wloadfont ("tiny.wfn");

Bitmapped fonts require memory to be allocated when loading them, so you should use wfreefont to release this memory when you are finished using the font.

Parameters: filename - Full pathname of font to load.

Return Value: Pointer to font in memory.

See Also: wfreefont, wouttextxy, wstring

Examples: 26, 40, 41, 47

wloadiff

Function: Loads an Amiga IFF/LBM format image and its associated palette.

Declaration: block wloadiff (char *filename, color *pal)

Remarks: Given a filename and a palette buffer, this function will load an IFF or LBM format image and return the pointer to it. The image may be of any dimensions or color depth.

If the call is successful, the pointer will indicate where the image is stored (uncompressed) in memory. An unsuccessful call will return NULL as the pointer.

Parameters: filename - The full pathname (or filename) of the image to be loaded

pal - The image palette will be stored in this buffer

See Also: none

Examples: 64

wloadpak

Function: Loads a compressed PAK file from disk into memory.

Declaration: block wloadpak (char *filename)

Remarks: wloadpak automatically allocates the appropriate amount of memory by calling malloc, and loads the data into the space allocated. It then returns a pointer to this memory so you may perform other actions on the block. You do should NOT call wnewblock before you load the block.

PAK files are a custom format, only used within WGT applications. It uses a run-length encoding compression scheme, and compresses a bit better than the PCX format. No palette information is saved.

Parameters: filename - Full pathname for image to be loaded.

Return Value: Pointer to the loaded PAK file, in block format. If the PAK could not be found on disk, it returns NULL.

See Also: wsavepak

Examples: 35, 61, 64

wloadpalette

Function: Loads a palette file from disk into a palette variable.

Declaration: void wloadpalette (char *filename, color *pal)

Remarks: Palette files created by WGT or popular graphics programs may be loaded into the variable specified by palette. Changes are not made to the currently displayed palette. A good example of compatibility would be the AutoDesk Animator. It's .COL files are of this type. Many commercial games use these files as well. Each file must be exactly 768 bytes, or it is an incorrect format.

The format of a palette file is:

```
unsigned char red1
unsigned char green1
unsigned char blue1
unsigned char red2
unsigned char green2
unsigned char blue2
...
unsigned char red256
unsigned char green256
unsigned char blue256
```

These values are repeated for each of the 256 palette registers, giving you a 768 byte file.

Parameters: filename - Full pathname for palette file to be loaded.

pal - Array in which to load the palette.

Return Value: None

See Also: wsavepalette, wsetpalette, wsetrgb

Examples: 35, 39, 54, 55

wloadpcx

Function: Loads a PCX file from disk into memory.

Declaration: block wloadpcx (char *filename, color *pal)

Remarks: wloadpcx automatically allocates the appropriate amount of memory by calling malloc, and loads the data into the space allocated. It then returns a pointer to this memory so you may perform other actions on the block. You do should NOT call wnewblock before you load the block. PCX files are created from many drawing programs. They use a run-length encoding compression scheme and unlike the PAK format, it includes palette information. Images may be any dimensions and any color depth up to 8-bit.

Parameters: filename - Full pathname for PCX image to be loaded.

pal - Array in which the PCX's palette will be stored.

Return Value: Pointer to the loaded PCX, in block format. If the PCX could not be found on disk, it returns NULL.

See Also: wsavepcx

Examples: 11, 21, 24, 31, 32, 34, 35, 48, 49, 53, 64

wloadsprites

Function: Loads in a sprite file.

Declaration: short wloadsprites (color *pal, char *filename, block *image_array,
short start, short end)

Remarks: wloadsprites will load a sprite file created with the WGT Sprite Editor. pal is the variable to store the palette of the sprites into. The command only loads sprites with numbers start to end. image_array is an array of blocks to load the sprites into. Your array must be large enough to hold the number of sprites between start and end.

Example usage:

```
block mysprites[800];
...
ok = wloadsprites (palette, "game.spr", mysprites, 0, 500);
```

Furthermore, you can "chain" sprite files together by giving an offset of the block array. For example to load another file at the end of the previous file,

```
ok = wloadsprites (palette, "stuff.spr", &mysprites[501], 0, 200);
```

Parameters: pal - Array of colors used in sprites.

filename - Full pathname of sprite file to load from.

image_array - Array of pointers to the sprites loaded.

start - Start index of sprites to load.

end - End index of sprites to load.

Return Value: -1 = unsuccessful
0 = successful

See Also: wfreesprites, wsavesprites

Examples: 19, 20, 22, 23, 25, 29, 31, 45, 56, 57, 58, 59, 60, 61, 62, 63, 65, 66, 67, 68

wnewblock

Function: Captures a section of the screen into a block.

Declaration: block wnewblock (short x1, short y1, short x2, short y2)

Remarks: Dynamic memory is used to automatically calculate the size of the data, and to store it in the standard block format. See wloadblock for a description of this format.

The size of the block is calculated, and then malloc is allocated to reserve enough memory for the block. The section of the current page is copied into this memory, and a pointer is returned to the new block. (x1,y1) defines the upper left corner of the region, and (x2,y2) defines the lower right corner.

example:

```
block mypicture;
void main (void)
{
  ...
  mypicture = wnewblock (50, 50, 60, 60);
  ...
}
```

Parameters: x1 - Horizontal coordinate of left edge.

y1 - Vertical coordinate of upper edge.

x2 - Horizontal coordinate of right edge.

y2 - Vertical coordinate of lower edge.

Return Value: A pointer the to block. If there is not enough memory to create the block, it returns NULL.

See Also: wallocblock, wflipblock, wfreeblock, wgetblockheight, wgetblockwidth, wloadblock, wputblock, wsaveblock

Examples: 9, 10, 15, 16, 17, 18, 19, 20, 24, 27, 33, 42, 43, 47, 51, 54, 55, 67

wnormscreen

Function:	Sets the active page back to the original visual page.
Declaration:	void wnormscreen (void)
Remarks:	This is used to return all drawing operations to the normal (visual) screen.
Parameters:	None
Return Value:	None
See Also:	wfreeblock, wnewblock, wsetscreen
Examples:	11, 15, 16, 17, 18, 20, 24, 33, 34, 56, 60, 61, 62, 67

wouttextxy

Function: Outputs a string of text.

Declaration: void wouttextxy (short x, short y, wgtfont font, char *string)

Remarks: The coordinate system may be be character or pixel based, depending on the state of the text grid system. See the wtextgrid procedure for more information. It uses the font for displaying bitmapped fonts which were loaded with wloadfont.

In place of the wgtfont, use NULL to display text with the default 8x8 font.

Parameters: x - Horizontal coordinate for string display.

y - Vertical coordinate for string display.

font - Pointer to font which will be used.

string - String to display.

See Also: wgtprintf, wloadfont, wstring, wtextgrid

Return Value: None

Examples: 4, 7, 13, 14, 15, 20, 24, 34, 48

wputblock

Function: Displays a block onto the current graphics page.

Declaration: void wputblock (short x, short y, block image, short method)

Remarks: This routine is used in a variety of situations for displaying bitmapped imaged onto the screen. It can be used to show single sprites, or full screen pictures which have been loaded with the set of image commands. X and y are the coordinates on the screen to show the bitmap image, and method can be either:

0 = Normal
1 = XRay

The XRay mode uses the 0 color as a "see-through" color. Any occurrence of this color is not drawn on the screen. This is useful for drawing sprites which overlay a background screen.

Parameters: x - Horizontal coordinate for image display.

y - Vertical coordinate for image display.

image - Pointer to image which will be displayed.

method - Technique used to display image.

Return Value: None

See Also: wloadblock, wloadbmp, wloadcel, wloadpak, wloadpcx, wnewblock

Examples: 9, 10, 11, 15, 18, 19, 20, 21, 29, 31, 33, 34, 35, 45, 47, 48, 49, 56, 57, 58, 59, 60, 61, 62, 64, 65, 67

wputblock_shade

Function: Displays a block onto the current graphics page, using a shade table.

Declaration: void wputblock_shade (short x, short y, block image,
unsigned char *shadetable, short mode)

Remarks: This routine is used for displaying bitmapped imaged onto the screen with a special lighting effect. X and y are the coordinates on the screen to show the bitmap image.

For the first three shade methods, shadetable is an array of 256 characters which defines how each color is processed. In general, each color turns into a different color depending on the value in this array. For instance if a pixel from image was color 34, the color written to the screen would be shadetable[34].

For the SHADE_TRANSLUCENT method, shadetable must be an array of 65536 characters. This technique takes a pixel from the image AND the destination screen and combines them into a number between 0 and 65535. The new color is taken from this location in the table.

For the SHADE_MONO method, shadetable is a pointer to a single character. The value of this character will be used as a color to draw the shape. This is useful for highlighting an irregular object.

mode tells which special lighting method to use on the image. It can be one of the following:

SHADE_NORMAL (0) Each pixel in the image is mapped to the new color in the shade table, including color 0.

SHADE_XRAY (1) Each pixel in the image is mapped to the new color in the shade table, but color 0 is not drawn. Note that if a color is set to 0 after passing through shadetable, it is drawn.

SHADE_SHADOW (2) For each pixel in the image which is not color 0, a pixel is taken from the destination screen and passed through shadetable. This lets you alter an area on a previously drawn screen in the shape of the image. The actual colors contained in image are not used. This method is only concerned about the pixel if it is greater than color 0.

SHADE_TRANSLUCENT(3) For each pixel in the image which is not color 0, a pixel is taken from both the destination screen and the source image and passed through shadetable. This is usually for creating translucent images where you can see part of the background and foreground at the same time.

SHADE_MONO(4) For each pixel in the image which is not color 0, a pixel is drawn with the color in shadetable.

Parameters: x - Horizontal coordinate for image display.

y - Vertical coordinate for image display.

image - Pointer to image which will be displayed.

shadetable - Pointer to the shade table

method - Technique used to display image.

Return Value: None

See Also: wputblock, wresize_shade

wputpixel

Function: Plots a pixel at the given screen coordinate.

Declaration: void wputpixel (short x, short y)

Remarks: Draws a point in the current drawing color (set by wsetcolor) at the coordinate (x,y) of the current graphics page.

Parameters: x - Horizontal coordinate of pixel to set.

y - Vertical coordinate of pixel to set.

Return Value: None

See Also: wfastputpixel, wgetpixel

Examples: 2, 8, 67

wreadpalette

Function: Reads the current palette into a palette variable.

Declaration: void wreadpalette (unsigned char start, unsigned char finish,
color *pal)

Remarks: Reads the palette registers between start and finish and stores them into the palette variable. This is used to capture the palette being used at the time.

Parameters: start - Start index of palette array to set.

finish - End index of palette array to set.

pal - Array of colors to be altered.

Return Value: None

See Also: wloadpalette, wsavepalette, wsetcolor, wsetpalette, wsetrgb

Examples: 4, 27, 40, 47, 48, 49

wrectangle

Function: Draws a rectangle using the current color and graphics page.

Declaration: void wrectangle (short x1, short y1, short x2, short y2)

Remarks: (x1,y1) defines the upper left corner of the rectangle, and (x2,y2) defines the lower right.

Parameters: x1 - Horizontal coordinate of left edge.

y1 - Vertical coordinate of upper edge.

x2 - Horizontal coordinate of right edge.

y2 - Vertical coordinate of lower edge.

Return Value: None

See Also: wbar

Examples: 4, 60

wregionfill

Function: Fills an area of the screen bounded by any color.

Declaration: void wregionfill (short x, short y)

Remarks: Fills an enclosed area starting at (x,y), which is bounded by pixels of any color other than the one at the original point. Full clipping is performed.

Parameters: x - Horizontal coordinate of starting pixel.

y - Vertical coordinate of starting pixel.

Return Value: None

Examples: 8

wremap

Function: Remaps the colors of a block or the visual screen to a new palette.

Declaration: void wremap (color *pal1, block image, color *pal2)

Remarks: pal1 is the palette belonging to the block image.
pal2 contains the new palette the block will use after remapping. This changes every pixel in the block from the normal palette, to the closest color in the new palette pal2. If image is NULL, the visual screen is used.

Parameters: pal1 - Array of colors normally used for image.

image - Image to be remapped.

pal2 - Palette to remap block with.

Return Value: None

See Also: wloadblock, wloadpalette

Examples: 48

wresize

Function: Draws a previously stored image on the screen to fit within a given boundary.

Declaration: void wresize (short x1, short y1, short x2, short y2, block image,
short mode)

Remarks: This routine will stretch a bitmapped image to a different size than it was originally stored in. The bitmap is displayed on the current graphics page. If the new size is larger than the original, the image becomes blockier. If the new size is smaller, some of the pixels within the image are removed. Resizing is generally used in special effects sequences, not within a game animation loop. It is too slow to draw several resized bitmaps on the screen at once, so you should resize each image beforehand and store each one as a block. mode is similar to the mode in wputblock.

mode = 0 : Normal
mode = 1 : X-ray (color 0 is not copied)

Parameters: x1 - Horizontal coordinate of left edge.

y1 - Vertical coordinate of upper edge.

x2 - Horizontal coordinate of right edge.

y2 - Vertical coordinate of lower edge.

image - Pointer to image which will be resized.

mode - Technique used when resizing (xray or normal)

Return Value: None

See Also: wresize_column, wresize_shade, wvertres, wwarp

Examples: 11, 19

wresize_column

Function: Resizes a single column of a bitmap to a new height on the screen.

Declaration: void wresize_column (short x, short y, short y2, block image,
short column, short mode)

Remarks: This routine will stretch a single column from a bitmapped image to a different height than it was originally stored in. The column is displayed on the current graphics page. If the new size is larger than the original, the image becomes blockier. If the new size is smaller, some of the pixels within the image are removed. mode is similar to the mode in wputblock.

mode = 0 : Normal
mode = 1 : X-ray (color 0 is not copied)

Parameters: x - Horizontal coordinate of column on screen.

y - Top y coordinate of column on screen.

y2 - Botton y coordinate of column on screen.

image - Pointer to image which will be resized.

column - Horizontal coordinate of column on image.
This must be less than the width of image.

mode - Technique used when resizing (xray or normal)

Return Value: None

See Also: wresize, wresize_shade, wvertres

Examples: None

wresize_shade

Function: Draws a previously stored image on the screen to fit within a given boundary, using a shade table.

Declaration: void wresize_shade (short x1, short y1, short x2, short y2, block image, unsigned char *shadetable, short mode)

Remarks: This routine will stretch a bitmapped image to a different size than it was originally stored in, while performing a special light effect on the image.

For the first three shade methods, shadetable is an array of 256 characters which defines how each color is processed. In general, each color turns into a different color depending on the value in this array. For instance if a pixel from image was color 34, the color written to the screen would be shadetable[34].

For the SHADE_TRANSLUCENT method, shadetable must be an array of 65536 characters. This technique takes a pixel from the image AND the destination screen and combines them into a number between 0 and 65535. The new color is taken from this location in the table.

For the SHADE_MONO method, shadetable is a pointer to a single character. The value of this character will be used as a color to draw the shape. This is useful for highlighting an irregular object.

mode tells which special lighting method to use on the resized image. It can be one of the following:

SHADE_NORMAL (0) Each pixel in the image is mapped to the new color in the shade table, including color 0.

SHADE_XRAY (1) Each pixel in the image is mapped to the new color in the shade table, but color 0 is not drawn. Note that if a color is set to 0 after passing through shadetable, it is drawn.

SHADE_SHADOW (2) For each pixel in the image which is not color 0, a pixel is taken from the destination screen and passed through shadetable. This lets you alter an area on a previously drawn screen in the shape of the image. The actual colors contained in image are not used. This method is only concerned about the pixel if it is greater than color 0.

SHADE_TRANSLUCENT(3) For each pixel in the image which is not color 0, a pixel is taken from both the destination screen and the source image and passed through shadetable. This is usually for creating translucent images where you can see part of the background and foreground at the same time.

SHADE_MONO(4) For each pixel in the image which is not color 0, a pixel is drawn with the color in shadetable.

Parameters: x1 - Horizontal coordinate of left edge.

y1 - Vertical coordinate of upper edge.

x2 - Horizontal coordinate of right edge.

y2 - Vertical coordinate of lower edge.

image - Pointer to image which will be resized.

shadetable - Pointer to the shade table

mode - Technique used when resizing.

Return Value: None

See Also: wresize, wresize_column, wvertres

Examples: 68

wretrace

Function: Waits for the vertical retrace.
Declaration: void wretrace (void)

Remarks: Use the wretrace command when you get flickering graphics. This command waits for the vertical retrace on your monitor, so you can write to the screen while it is not refreshing the screen.

It will not work perfectly unless you are writing small amounts to the screen, but it is worth a try if you get flickering screens. This routine can also be used for timing (obtaining constant speeds on different CPUs).

Parameters: None

Return Value: None

Examples: 3, 5, 20, 21, 22, 23, 24, 25, 27, 29, 37, 39, 51, 54, 56, 57, 59, 60, 61, 65

wsaveblock

Function: Saves a block to a disk file in raw format.

Declaration: short wsaveblock (char *filename, block image)

Remarks: A file is created which holds the data stored at the pointer given by image. This allows your programs to store images that can be loaded at a later time.

Parameters: filename - Full pathname for image to save.

image - Pointer to the image which will be saved.

Return Value: 0 if successful, 1 if it could not save the file

See Also: wfreeblock, wloadblock, wnewblock

Examples: 35

wsavebmp

Function: Saves a block image to disk in BMP format.

Declaration: short wsavebmp (char *filename, block image, color *pal)

Remarks: Given a filename and a pointer to a block, this function saves the image in BMP format (8-bit color).

Parameters: filename - Full pathname of image to save.

image - Pointer to actual image.

pal - Palette to save with image.

Return Value: None

See Also: wloadbmp

Examples: 35

wsavecel

Function: Saves a CEL file (AutoDesk Animator format) to disk.

Declaration: short wsavecel (char *filename, block image, color *pal)

Remarks: A file is created which holds the data stored at the pointer given by image. This allows your programs to store images that can be loaded at a later time.

Parameters: filename - Full pathname of image to save.

image - Pointer to actual image in memory.

pal - Palette to save with image.

Return Value: 0 if succesful, 1 if it could not save the file

See Also: wloadcel

Examples: 35

wsavepak

Function: Saves a block to disk in compressed format.

Declaration: short wsavepak (char *filename, block image)

Remarks: This is similar to wsaveblock except that the image is in a file which is compressed. The compression used is "similar" to PCX format (sometimes better). The more simplistic the image, the smaller the file. The image still requires the same amount of memory when loading it in, but takes up less disk space.

Parameters: filename - Full pathname of image to save.

image - Pointer to image.

Return Value: 0 if succesful, 1 if it could not save the file

See Also: wloadblock, wloadpak

Examples: 35

wsavepalette

Function: Save a palette variable's contents to a file.

Declaration: void wsavepalette (char *filename, color *pal)

Remarks: Creates a 768 byte palette file to disk from a variable specified by palette. This is useful for storing different palettes for games and their various screens.

Parameters: filename - Full pathname of palette file to save.

pal - Actual palette to save.

Return Value: None

See Also: wloadpalette, wsetpalette, wsetrgb

Examples: 35

wsavepcx

Function: Saves a block to disk in the PCX compressed format.

Declaration: short wsavepcx (char *filename, block image, color *pal)

Remarks: This is similar to wsaveblock except that the image is in a file which is compressed. The compression used is the PCX format. The more simplistic the image, the smaller the file. The image still requires the same amount of memory when loading it in, but takes up less disk space.

Parameters: filename - Full pathname of PCX file to save.

image - Pointer to image in memory.

pal - Palette to save with image.

Return Value: None

See Also: wloadblock, wloadpak, wloadpcx

Examples: 35

wsavesprites

Function: Saves a sprite file using a palette array and a set of blocks.

Declaration: short wsavesprites (color *pal, char *filename, block *image_array,
short start, short end)

Remarks: wsavesprites will save a sprite file compatible with the WGT Sprite Editor. pal is the array of color info used for the images. image_array is the array of blocks containing the sprites. This function will save from index "start" to whatever value you specify as the "end" sprite in the array. Your array must be large enough to hold the given number of sprites between start and end or a page fault will occur.

Parameters: pal - Array of colors used in sprites.

filename - Full pathname of sprite file to save.

image_array - Array of pointers to the sprites in memory.

start - Lowest index of sprites to save.

end - Highest index of sprites to save.

Return Value: -1 = unsuccessful
0 = successful

See Also: wloadsprites

Examples: 45

wscan_convertpoly

Function: Scan converts a concave polygon into a rendered polygon buffer.

Declaration: void wscan_convertpoly (tpolypoint *vertexlist, short numvertex,
wscanpoly *scanpoly)

Remarks: Concave polygons allow multiple horizontal line segments on each row of the polygon. This means you can have any number of vertices in any position and the polygon will still be drawn correctly.

Rendered polygons are only stored in an allocated buffer. They are not drawn until wdraw_scanpoly is called. The benefit of storing the polygon in a buffer is you can draw the polygon in a different location on the screen without doing the mathematical calculations again.

wscanpoly is a structure to hold the polygon buffer. It is defined as:

```
typedef struct
{
  char **pointbuffer;          /* Holds a buffer of x coordinates for each row
                                  of the polygon. */
  unsigned short *numpoints;  /* Holds the size of the point buffer in bytes. */
} wscanpoly;
```

Parameters: vertexlist - Array of vertices that define the concave polygon.

numvertex - Total number of vertices in the array.

scanpoly - Array of vertices defining the rendered polygon.

Return Value: None

See Also: winitpoly, wdeinitpoly, wdraw_scanpoly, wfree_scanpoly

Examples: 51

wset_capslock

Function: Sets the state of the capslock key.

Declaration: void wset_capslock (short state)

Remarks: This routine will set the state of the capslock key and change the keyboard LEDS appropriately.

Parameters: state - Indicates on/off status of key.
0=off 1=on

Return Value: None

See Also: wget_capslock, wset_numlock, wset_scrlock

Examples: 44

wset_imagebyte

Function: Stores a byte of information at the end of allocated memory for an image.

Declaration: void wset_imagebyte (block image, unsigned char status)

Remarks: This routine stores a value as the last allocated byte of memory for that image and it is recommended the the information is used in the following format:

Bit 0 - Indicates Xray or Normal block
Bits 1-7 are unused.

If bit 0 is set (status is 1), the image contains some pixels of color index 0. This means that an XRAY putblock would be effective to make the image see-through for parallax scrolling and other such effects. If the status is 0, a normal putblock would be the fastest way of blasting it to the screen.

Parameters: image - Pointer to the image in memory.

status - A single byte of information to be stored.

Return Value: None

See Also: wget_imagebyte, wget_imagebit, wupdate_imagebytes

Examples: None

wset_numlock

Function: Sets the state of the numlock key.

Declaration: void wset_numlock (short state)

Remarks: This will set the state of the numlock key and change the keyboard LEDS appropriately.

Parameters: state - Indicates on/off status of key.
0=off 1=on

Return Value: None

See Also: wget_numlock, wset_capslock, wset_scrlock

Examples: 44

wset_scrlock

Function: Sets the state of the scrlock key.

Declaration: void wset_scrlock (short state)

Remarks: This will set the state of the scroll lock key and change the keyboard LEDS appropriately.

Parameters: state - Indicates on/off status of key.
0=off 1=on

Return Value: None

See Also: wget_scrlock, wset_capslock, wset_numlock

Examples: 44

wsetcolor

Function: Sets the current drawing color.

Declaration: void wsetcolor (unsigned char col)

Remarks: Col may be in the range 0 to 255. All drawing procedures used after setting this will use the color you choose.

Parameters: col - Color index to use for graphics primitives.

Return Value: None

See Also: wloadpalette, wsavepalette, wsetpalette, wsetrgb

Examples: 1, 2, 4, 5, 6, 8, 9, 10, 12, 14, 15, 16, 17, 18,
19, 20, 23, 25, 27, 28, 29, 33, 36, 39, 42, 46, 47, 51, 53, 54, 55, 60, 61, 66, 67

wsetcursor

Function: Sets the shape of the software emulated text cursor.

Declaration: void wsetcursor (short start, short end)

Remarks: start and end are the y coordinates of the upper and lower lines for a cursor 8 pixels wide. The X coordinates are fixed to 8 pixels wide. The cursor is not the hardware text cursor, but rather a software simulation which flashes during text input. A solid box would be (0,7).

Parameters: start - Upper coordinate of cursor.

end - Lower coordinate of cursor.

See Also: wflashcursor, wstring

Return Value: None

Examples: 14

wsetmode

Function: Sets the video mode.

Declaration: void wsetmode (short mode)

Remarks: This is mainly used to restore the video mode. The most common mode to restore is 0x03, which is 80x25 Text mode.

Parameters: mode - Video mode to switch into.

Return Value: None

See Also: vga256, vgadetected, wgetmode

Examples: 1-29

wsetpalette

Function: Sets a group of colors from a specified palette variable.

Declaration: void wsetpalette (unsigned char start, unsigned char finish,
color *pal)

Remarks: Colors from start to finish are set using the values stored in the palette variable. The variable type color is specific to WGT and may not be used elsewhere. It is defined as follows:

```
typedef struct {
  unsigned char r, g, b;
} color;
```

To define a palette, you must make an array of colors like this:

```
color palette[256];
```

Parameters: start - Start index of palette array to set.

finish - End index of palette array to set.

pal - Array of colors to use.

Return Value: None

See Also: wloadpalette, wreadpalette, wsavepalette, wsetcolor, wsetrgb

Examples: 4, 5, 6, 7, 11, 13, 19, 20, 21, 22, 23, 24, 25, 27, 31, 32, 34, 35, 39, 40, 42, 45, 47, 48, 49, 51, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 63, 64, 65, 66, 67

wsetrgb

Function: Sets a color's red, green, and blue values.

Declaration: void wsetrgb (unsigned char col, unsigned char red,
unsigned char green, unsigned char blue, color *pal)

Remarks: Col can range from 0 to 255. The Red, Green, and Blue values are in the range 0 to 63 giving a total of 262144 color combinations possible (64*64*64).

The colors do NOT change until you call the wsetpalette command. This way you can set a number of colors and change them all simultaneously.

Parameters: col - Index within color array to set.

red - Red level to use (0-63).

green - Green level to use (0-63).

blue - Blue level to use (0-63).

pal - Array of colors to set.

Return Value: None

See Also: wloadpalette, wsavepalette, wsetcolor, wsetpalette

Examples: 4, 5, 6, 7, 13, 20, 40, 42, 47, 51, 55

wsetscreen

Function: Makes all drawing procedures use a specified destination video buffer.

Declaration: void wsetscreen (block image)

Remarks: Sets the active drawing buffer to image. Every WGT graphics routine will write to this page instead of the visual screen. This is used to create pictures in memory which will be displayed later using wputblock or wcopyscreen. Clipping values are adjusted to meet the new virtual screen dimensions.

The screen must be first allocated by using wnewblock or a block load function. It can be any size.

If image is NULL, the active buffer is set to the visual screen. (equivalent to wnormscreen)

Parameters: image - Pointer to screen which becomes the active screen.

Return Value: None

See Also: wfreeblock, wnewblock

Examples: 15, 16, 17, 18, 19, 20, 33, 34, 42, 54, 55, 60, 61, 63, 67

wskew

Function: Skews a block sideways.

Declaration: void wskew (short x, short y, block image, short degrees)

Remarks: Shows the bitmap image on the current page at (x,y) using a skew value of degrees. Each row of the bitmap is pushed either left or right of the previous. If degrees is negative, it will be skewed in the opposite direction.

Parameters: x - Horizontal display coordinate.

y - Vertical display coordinate.

image - Pointer to image to be skewed.

degrees - Number of degrees to skew image.

Return Value: None

Examples: 27

wsline

Function: Stores the points of a line into an array instead of plotting them on the screen.

Declaration: void wsline (short x1, short y1, short x2, short y2, short *y_array)

Remarks: Simply stores the y coordinate values in the appropriate place in the array for every x coordinate on a line. y_array is an array of short integers. See wwarp for more info.

Parameters: x1 - Horizontal coordinate of first endpoint.

y1 - Vertical coordinate of first endpoint.

x2 - Horizontal coordinate of second endpoint.

y2 - Vertical coordinate of second endpoint.

y_array - Buffer used to store points in the line.

Return Value: None

See Also: wwarp

Examples: 32

wsolidpoly

Function: Draws a filled convex polygon.

Declaration: void wsolidpoly (tpolypoint *vertexlist, short numvertex,
short x, short y,
void (*customline)(short x1, short x2, short y))

Remarks: The wsolidpoly routine draws a filled convex polygon using the current color. Each vertex is has the offset (x,y) added to it before the polygon is drawn. vertexlist is an array of vertices, which contains numvertex vertices. customline is the name of the horizontal line routine to call for each line in the polygon. If NULL is given, it will use whline. You can create your own line routines to make other unique effects such as fill patterns using the wstyleline command, image copying using memcpy, or shadebob techniques.

The tpolypoint structure is defined as:

```
typedef struct
{
  short x,y;  /* Coordinate on the screen */
  short sx,sy; /* Coordinate on the texture */
} tpolypoint;
```

wsolidpoly does not use the sx,sy variables. While this is wasteful of memory, it allows you to switch between polygon rendering methods without altering your vertex data variables.

Customline is a function which accepts three parameters. The first two are the x coordinates of a horizontal line, and the third is the y coordinate.

Parameters: vertexlist - Array of vertices defining the polygon.

numvertex - Total number of vertices in the array.

x - Horizontal offset to add to each vertex.
y - Vertical offset to add to each vertex.
customline - Address of custom horizontal line routine.

Return Value: none

See Also: wdeinitpoly, whollowpoly, winitpoly

Examples: 42, 51, 54, 55

wstarttimer

Function: Installs a custom timer virtual interrupt at the given speed.

Declaration: void wstarttimer (int speed)

Remarks: sets the speed at which the timer will 'tick' in milliseconds.

wtimerproc is defined as:
typedef void (*wtimerproc)(void);

Parameters: timer - The procedure name of the new timer routine

speed - The new timer speed

Return Value: None

See Also: wdonetimer, winittimer, wsettimerspeed, wstoptimer

Examples: 3, 11, 46, 55, 56, 57, 58, 59, 61, 62, 63, 68

wstoptimer

Function: Resets a custom timer interrupt to an empty procedure.

Declaration: void wstoptimer (void)

Remarks: This command will remove the custom timer procedure and replace it with a procedure which does nothing. The timer interrupt itself is still present, and must be removed with wdonetimer.

Parameters: None

Return Value: None

See Also: wdonetimer, winittimer, wsettimerspeed, wstarttimer

Examples: 3, 11, 46, 55, 56, 57, 58, 59, 61, 62, 63, 68

wstring

Function: Reads in a string of text from the keyboard and displays both the text and a simulated cursor on the graphics page.

Declaration: void wstring (short x, short y, char *string, char *legal,
short maxlength)

Remarks: This procedure uses the flashing cursor while you input the string. Any characters you wish to be able to type in must be included in LEGAL. For example, if you want a yes or no answer only, make LEGAL="YNyn". num is the maximum number of letters in the string. x and y are the coordinates of the initial cursor position.

STRING must be a pointer to char:

```
char *filename;
```

Along with legal:

```
char *legal_chars = " ABCDEFGHIJKLMNOPQRSTUVWXYZ
                    abcdefghijklmnopqrstuvwxyz1234567890_.";
```

Then, you must allocate memory for the string, like this:

```
filename = (char *) malloc (13);
```

Add 1 to the length for the null terminator.

Call the string procedure:

```
wstring (10, 22, filename, legal_chars, 12);
```

After you're done with the string, free the memory by

```
free (filenameout);
```

INSERT,DELETE,HOME,END and the arrow keys are functional, and backspace is destructive.

If you press insert, the cursor shape changes and toggles between insert and overwrite modes.

Parameters:

x	- Horizontal coordinate for text input.
y	- Vertical coordinate for text input.
string	- Character string input buffer.
legal	- String of legal input characters.
maxlength	- Maximum number of characters to input.

Return Value: None

See Also: wflashcursor, wgtprintf, wouttextxy

Examples: 14

wstyleline

Function: Draws a line from between two points with a line pattern.

Declaration: void wstyleline (short x, short y, short x2, short y2,
unsigned short style)

Remarks: (x1,y1) defines the first endpoint of the line. (x2,y2) defines the second endpoint of the line. pattern is a number between 0 and 65535 which defines the line pattern. The style can be thought of as an array of 16 bits, each telling if a pixel is on (1) or off (0).

For example, a dashed line would have half of the pixels turned on, and half of them turned off. This would be represented as 1111111100000000 in binary.

Of course you cannot use binary numbers within your C program, so you must first convert them to hexadecimal. To convert this number, split the digits into groups of four. Below is a table of the binary group on the left, and the hexadecimal equivalent on the right.

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Some examples of this conversion are:

Binary: 1111 1111 0000 0000
Hexadecimal: F F 0 0 = 0xFF00

Binary: 1000 1000 1001 1001
Hexadecimal: 8 8 9 9 = 0x8899

This conversion process is also important when creating user defined mouse cursors with the mouseshape routine.

Parameters: x - Horizontal coordinate of first endpoint.

y - Vertical coordinate of first endpoint.

x2 - Horizontal coordinate of second endpoint.

y2 - Vertical coordinate of second endpoint.

style - Bitpattern of the line.

Return Value: None

See Also: wfline, wline

Examples: 4

wtextbackground

Function: Sets the background text color.

Declaration: void wtextbackground (unsigned char col)

Remarks: Col is in the range 0 to 255. The background color behind text may be turned off by calling wtexttransparent.

Parameters: col - Color index to use for text background.

Return Value: None

See Also: wouttextxy, wstring, wtextcolor, wtextcursor, wtextgrid, wtexttransparent

Examples: 7, 14, 20, 26, 53, 58, 59

wtextcolor

Function: Selects the foreground character color.

Declaration: void wtextcolor (unsigned char col)

Remarks: Col is in the range 0 to 255. The foreground color of text (the actual letters) may be turned off by calling wtexttransparent.

Parameters: col - Color index to use for text output.

Return Value: None

See Also: wouttextxy, wstring, wtextbackground, wtextcursor, wtextgrid, wtexttransparent

Examples: 4, 7, 12, 13, 14, 15, 20, 24, 26, 34, 35, 41, 48, 49, 51, 53, 58, 59, 63, 67

wtextgrid

Function: Switches the text coordinate system between character and pixel based.

Declaration: void wtextgrid (short state)

Remarks: xt may be output in regular graphics coordinates in pixels, or a text mode system which ligns the strings to character boundaries. If STATE is set to 1, all input and output values for WGT text functions are expected to be character based. Characters will be 'snapped' to the grid as if it were a text mode. 0 will use the regular graphics system coordinates.

Parameters: state - Indicates text gridlock on/off status.

Return Value: None

See Also: wouttextxy, wstring

Examples: 7

wtexttransparent

Function: Sets text output to suppress the display of the foreground or background.

Declaration: void wtexttransparent (short mode)

Remarks: This will turn off the display of either the foreground or the background, or set it to show both.

The values possible are:
TEXTFG (0): Foreground turned on
TEXTBG (1): Background turned on
TEXTFGBG(2): Both turned on

Parameters: mode - Indicates visibility status of text fore/background.

Return Value: None

See Also: wtextbackground, wtextcolor

Examples: 7, 12, 13, 14, 20, 26, 34, 48, 49, 51, 53, 58, 59, 63

wtexturedpoly

Function: Draws a texture mapped convex polygon.

Declaration: void wtexturedpoly (tpolypoint *vertexlist, short numvertex,
short x, short y, block image, short mode)

Remarks: The wtexturedpoly routine draws a texture mapped convex polygon using the texture bitmap image. Each vertex is has the offset (x,y) added to it before the polygon is drawn. vertexlist is an array of vertices, which contains numvertex vertices.

Mode can either be NORMAL (0), or XRAY (1) , which allows for see-through textures. The tpolypoint structure is defined as:

```
typedef struct
{
  short x,y;  /* Coordinate on the screen */
  short sx,sy; /* Coordinate on the texture */
} tpolypoint;
```

(sx,sy) contains the offset within the texture bitmap of the vertex. Each vertex in the polygon has a corresponding coordinate on the texture bitmap.

Parameters: vertexlist - Array of vertices which define the polygon.

numvertex - Total number of vertices in array.

x - Horizontal offset added to each vertex.

y - Vertical offset added to each vertex.

image - Pointer to block used as a texture.

mode - Indicates copy mode (xray or normal).

Return Value: None

See Also: wgouraudpoly, whollowpoly, wsolidpoly

Examples: 42

wtriangle_flat_shaded_texture

Function: Draws a flat shaded texture mapped triangle.

Declaration: void wtriangle_flat_shaded_texture (tpoint *ttri, block texture,
short shade, block shadetable)

Remarks: The wtriangle_flat_shaded_texture routine draws a flat shaded texture mapped triangle on the screen using texture as the image, and shadetable for the lighting table. shadetable is an array of unsigned characters. It can be defined as follows:

unsigned char shadetable[NUMSHADES][256];
shade tells which table of 256 characters to use as the lighting table. It can range from 0 to NUMSHADES - 1.

When drawing the triangle, each pixel is taken from the image texture, then a new color is looked up from shadetable[shade][color] and displayed on the screen.
This allows you to remap the texture to a different palette as it is drawn.

texture must have a width of 256, and has a maximum height of 256. This is the only limitation of the rendering system. You can place multiple pictures on a single texture block and show them by assigning different texture coordinates.

ttri is an array of 3 vertices. Each vertex has the following structure:

```
typedef struct
  {
   short x, y;              /* Coordinate on the screen */
   long sx, sy;             /* Coordinate on the texture */
   short col;               /* Shade/color  value */
  } tpoint;
```

For this routine, col is not used.

Before using this routine, you must initialize the triangle rendering engine using winit_triangle_renderer.

Parameters: ttri - Pointer to 3 tpoints which hold the vertex information.

texture - Image containing the texture (max 256x256)

shade - Tells the base offset into the shadetable

shadetable - Array of unsigned characters to use for remapping

Return Value: None

See Also: winit_triangle_renderer, wdeinit_triangle_renderer, wtriangle_gouraud, wtriangle_gouraud_shaded_texture, wtriangle_solid, wtriangle_texture, wtriangle_translucent_gouraud, wtriangle_translucent_texture

Examples: 3D_CAM

wtriangle_gouraud

Function: Draws a Gouraud shaded triangle.

Declaration: void wtriangle_gouraud (tpoint *gtri)

Remarks: The wtriangle_gouraud routine draws a Gouraud shaded triangle on the screen. gtri is an array of 3 vertices. Each vertex has the following structure:

```
typedef struct
  {
   short x, y;              /* Coordinate on the screen */
   long sx, sy;             /* Coordinate on the texture */
   short col;               /* Shade/color  value */
  } tpoint;
```

For this routine, sx and sy are not used.

The colors within the Gouraud shaded triangle are in the range 0 to 255. The triangle will use all the colors between the three col values of the vertices.

Before using this routine, you must initialize the triangle rendering engine using winit_triangle_renderer.

Parameters: gtri - Pointer to 3 tpoints which hold the vertex information.

Return Value: None

See Also: winit_triangle_renderer, wdeinit_triangle_renderer, wtriangle_flat_shaded_texture, wtriangle_gouraud_shaded_texture, wtriangle_solid, wtriangle_texture, wtriangle_translucent_gouraud, wtriangle_translucent_texture

Examples: 3D_CAM

wtriangle_gouraud_shaded_texture

Function: Draws a gouraud shaded texture mapped triangle.

Declaration: void wtriangle_gouraud_shaded_texture (tpoint *ttri, block texture,
block shadetable)

Remarks: The wtriangle_gouraud_shaded_texture routine draws a Gouraud shaded texture mapped triangle on the screen using texture as the image, and shadetable for the lighting table. shadetable is an array of unsigned characters. It can be defined as follows:

unsigned char shadetable[NUMSHADES][256];

When drawing the triangle, each pixel is taken from the image texture, then a new color is looked up from shadetable[shade][color] and displayed on the screen.
This allows you to remap the texture to a different palette as it is drawn.

texture must have a width of 256, and has a maximum height of 256. This is the only limitation of the rendering system. You can place multiple pictures on a single texture block and show them by assigning different texture coordinates.

ttri is an array of 3 vertices. Each vertex has the following structure:

```
typedef struct
  {
   short x, y;              /* Coordinate on the screen */
   long sx, sy;             /* Coordinate on the texture */
   short col;               /* Shade/color  value */
  } tpoint;
```

shade ranges from 0 to NUMSHADES - 1 and it is assumed the shadetable is set up correctly. The triangle will use all the shades between the three col values of the vertices.

Before using this routine, you must initialize the triangle rendering engine using winit_triangle_renderer.

Parameters: ttri - Pointer to 3 tpoints which hold the vertex information.

texture - Image containing the texture (max 256x256)

shadetable - Array of unsigned characters to use for remapping

Return Value: None

See Also: winit_triangle_renderer, wdeinit_triangle_renderer, wtriangle_flat_shaded_texture, wtriangle_gouraud, wtriangle_solid, wtriangle_texture, wtriangle_translucent_gouraud, wtriangle_translucent_texture

Examples: 3D_CAM

wtriangle_solid

Function: Draws a filled triangle.

Declaration: void wtriangle_solid (tpoint *ftri)

Remarks: The wtriangle_solid routine draws a filled triangle on the screen using the current drawing color. ftri is an array of 3 vertices. Each vertex has the following structure:

```
typedef struct
  {
   short x, y;              /* Coordinate on the screen */
   long sx, sy;             /* Coordinate on the texture */
   short col;               /* Shade/color  value */
  } tpoint;
```

For this routine, sx, sy, and col are not used.

Before using this routine, you must initialize the triangle rendering engine using winit_triangle_renderer.

Parameters: ftri - Pointer to 3 tpoints which hold the vertex information.

Return Value: None

See Also: winit_triangle_renderer, wdeinit_triangle_renderer, wtriangle_flat_shaded_texture, wtriangle_gouraud, wtriangle_gouraud_shaded_texture, wtriangle_texture, wtriangle_translucent_gouraud, wtriangle_translucent_texture

Examples: 3D_CAM

wtriangle_texture

Function: Draws a texture mapped triangle.

Declaration: void wtriangle_texture (tpoint *ttri, block texture)

Remarks: The wtriangle_texture routine draws a texture mapped triangle on the screen using texture as the image.

texture must have a width of 256, and has a maximum height of 256. This is the only limitation of the rendering system. You can place multiple pictures on a single texture block and show them by assigning different texture coordinates.

ttri is an array of 3 vertices. Each vertex has the following structure:

```
typedef struct
  {
   short x, y;              /* Coordinate on the screen */
   long sx, sy;             /* Coordinate on the texture */
   short col;               /* Shade/color  value */
  } tpoint;
```

For this routine, col is not used.

Before using this routine, you must initialize the triangle rendering engine using winit_triangle_renderer.

Parameters: ttri - Pointer to 3 tpoints which hold the vertex information.

texture - Image containing the texture (max 256x256)

Return Value: None

See Also: winit_triangle_renderer, wdeinit_triangle_renderer, wtriangle_flat_shaded_texture, wtriangle_gouraud, wtriangle_gouraud_shaded_texture, wtriangle_solid, wtriangle_translucent_gouraud, wtriangle_translucent_texture

Examples: 3D_CAM

wtriangle_translucent_gouraud

Function: Draws a translucent gouraud shaded triangle.

Declaration: void wtriangle_translucent_gouraud(tpoint *gtri, block shadetable)

Remarks: The wtriangle_translucent_gouraud routine draws a translucent Gouraud shaded triangle on the screen using shadetable for the lighting table. shadetable is an array of unsigned characters. It can be defined as follows:

```
unsigned char shadetable[256][256];
```

When drawing the triangle, a pixel is taken from the shaded triangle, and a pixel is taken from the current drawing buffer, then a new color is looked up from shadetable[backgroundcolor][trianglecolor] and displayed on the screen. The table is used to create a single color given two color. Usually this is done by taking a percentage of the first color and adding it to a percentage of the second color.

gtri is an array of 3 vertices. Each vertex has the following structure:

```
typedef struct
  {
   short x, y;              /* Coordinate on the screen */
   long sx, sy;             /* Coordinate on the texture */
   short col;               /* Shade/color  value */
  } tpoint;
```

For this routine, sx and sy are not used.

The colors within the Gouraud shaded triangle are in the range 0 to 255. The triangle will use all the colors between the three col values of the vertices.

Before using this routine, you must initialize the triangle rendering engine using winit_triangle_renderer.

Parameters: gtri - Pointer to 3 tpoints which hold the vertex information.

shadetable - Array of unsigned characters to use for remapping

Return Value: None

See Also: winit_triangle_renderer, wdeinit_triangle_renderer, wtriangle_flat_shaded_texture, wtriangle_gouraud, wtriangle_gouraud_shaded_texture, wtriangle_solid, wtriangle_texture, wtriangle_translucent_texture

Examples: 3D_CAM

wtriangle_translucent_texture

Function: Draws a translucent texture mapped triangle.

Declaration: void wtriangle_translucent_texture (tpoint *ttri, block texture,
block shadetable)

Remarks: The wtriangle_translucent_texture routine draws a translucent texture mapped triangle on the screen using texture as the image, and shadetable for the lighting table. shadetable is an array of unsigned characters. It can be defined as follows:

unsigned char shadetable[256][256];

When drawing the triangle, a pixel is taken from the image texture, and a pixel is taken from the current drawing buffer, then a new color is looked up from shadetable[backgroundcolor][texturecolor] and displayed on the screen. This allows you to remap the texture to a different palette as it is drawn. The table is used to create a single color given two color. Usually this is done by taking a percentage of the first color and adding it to a percentage of the second color. Besides translucent textures, you can also create tinted textures by adding to the red, green, or blue values of the color.

texture must have a width of 256, and has a maximum height of 256. This is the only limitation of the rendering system. You can place multiple pictures on a single texture block and show them by assigning different texture coordinates.

ttri is an array of 3 vertices. Each vertex has the following structure:

```
typedef struct
  {
   short x, y;              /* Coordinate on the screen */
   long sx, sy;             /* Coordinate on the texture */
   short col;               /* Shade/color  value */
  } tpoint;
```

For this routine, col is not used.

Before using this routine, you must initialize the triangle rendering engine using winit_triangle_renderer.

Parameters: ttri - Pointer to 3 tpoints which hold the vertex information.

texture - Image containing the texture (max 256x256)

shadetable - Array of unsigned characters to use for remapping

Return Value: None

See Also: winit_triangle_renderer, wdeinit_triangle_renderer, wtriangle_flat_shaded_texture, wtriangle_gouraud, wtriangle_gouraud_shaded_texture, wtriangle_solid, wtriangle_texture, wtriangle_translucent_gouraud

Examples: 3D_CAM

wupdate_imagebytes

Function: Updates status bytes for an array of images.

Declaration: void wupdate_imagebytes (block *image_array, short start,
short end)

Remarks: This routine updates a value from the last allocated byte of memory for each image in the array of images (from start index to end index). The value indicates the following information:

If the status is 1, the image contains some pixels of color index 0. This means that an XRAY putblock would be effective to make the image see-through for parallax scrolling and other such effects. If the status is 0, a normal putblock would be the fastest way of blasting it to the screen.

Parameters: image_array - Array of images in memory.

start - Index of first image to update

end - Index of last image to update

Return Value: none

See Also: wset_imagebyte, wget_imagebyte, wget_imagebit

Examples: None

wvertres

Function: Draws a previously stored image on the screen to fit within a given boundary but only stretches it vertically.

Declaration: void wvertres (short x1, short y1 short y2, block image)

Remarks: This is an extremely fast shrink/expand routine for blocks. It is similar to wresize but doesn't change the width of the block. Use this at all times if you are only resizing the height, since it is much faster.

Parameters: x1 - Horizontal coordinate of image.

y1 - Upper Y coordinate of image.

y2 - Lower Y coordinate of image.

image - Pointer to image which will be resized.

Return Value: None

See Also: wresize

Examples: 21

wwaitfortick

Function: Sits waiting for next timer tick, returns when tick occurs. Should return if timer is not running and function is called by accident. Makes a good replacement for DosSleep.

Declaration: void wwaitfortick()

Remarks: See Function desciption, you need the IBM Hi-Res Timer installed.

Parameters:None

Return Value: None

See Also: wresetticks wchecktimerticks

Examples: NONE

wwarp

Function: Displays a block on the screen, using different heights for each column displayed.

Declaration: void wwarp (short x1, short x2, short *top, short *bot, block image,
int mode)

Remarks: Instead of drawing the block straight across in a rectangular manner, this function resizes the block vertically for every column displayed. You can have each column stretched by different amounts allowing for some very interesting effects. X1 and X2 are the two x coordinates the block will be resized between. Top and bot are 320 integer arrays which hold the y values for every x coordinate. Image is the block to warp. Mode is the same as the XRAY or NORMAL modes used in wputblock.

Use wsline to set up the top and bot arrays if you are using straight lines.

Example:

```
wsline (0, 199, 319, 0, &top);                  // sets up line for top
wsline (0, 199, 319, 199, &bot);    // sets up bottom line
// Imagine drawing these two lines on the screen...

wwarp (0, 319, &top, &bot, warp_block);
```

Top should have all points above the corresponding point in bot. The block is resized between these points on the screen.

You do not need to use wsline to make straight lines for warping. Other possibilities include zig-zags, sine waves, etc.

Parameters: x1 - Horizontal coordinate of left edge.

x2 - Horizontal coordinate of right edge.

top - Array of y coordinates for upper edge.

bot - Array of y coordinates for lower edge.

image - Pointer to image which will be resized.

mode - Either XRAY or NORMAL display mode

Return Value: None

See Also: wsline

Examples: 32

wwipe

Function: Draws a line but uses colors from another virtual screen.

Declaration: void wwipe (short x1, short y1, short x2, short y2, block image)

Remarks: This special effect allows you to draw a line on the screen, but instead of using one color,each pixel drawn is the color of the pixel at the same location on the block screen. You can create impressive wipes and make your programs look very professional. You will want to make one or more 'for' statements that copy the whole screen over since this only does it one line at a time. The lines don't have to be horizontal or vertical, so you can create complex patterns with the lines.

Parameters: x1 - Horizontal coordinate of first endpoint.

y1 - Vertical coordinate of first endpoint.

x2 - Horizontal coordinate of second endpoint.

y2 - Vertical coordinate of second endpoint.

image - Screen to use as the wipe source.

Return Value: None

Examples: 24

wxorbox

Function: Draws a filled rectangle using exclusive-or mode.

Declaration: void wxorbox (short x, short y, short x2, short y2,
unsigned char col)

Remarks: This routine performs an XOR on the pixels within the box using the color given. It is useful for drawing outlines or showing button presses. If you draw a box on the same place twice in a row using the same color, the box will erase itself due to the results of the XOR mode.

Parameters:

x	- Horizontal coordinate of first endpoint.
y	- Vertical coordinate of first endpoint.
x2	- Horizontal coordinate of second endpoint.
y2	- Vertical coordinate of second endpoint.
col	- Color index used to draw the xor box.

Return Value: None

Examples: 36

WJOY_WC.LIB

Most games will need some method of detecting and reading from the joystick port(s) of the computer. These routines provide full access to 1 or 2 joysticks which are connected to the system. Routines for detection, calibration, and reading the button and positional values are all found in this library.

wcalibratejoystick

Function: Finds out the coordinate range of the joystick.

Declaration: short wcalibratejoystick (joystick *joy)

Remarks: Before using the joystick, it is best to calibrate it. Calibration simply finds out the range of values the joystick can return, and this is done by moving the stick to the extreme direction in each axis.

This routine can be used in two ways:
1. Tell the user to move the joystick to the upper left and press a fire button. Call wcalibratejoystick. Tell the user to move the joystick to the lower right and press a fire button. Call wcalibratejoystick again.

2. Tell the user to swirl the joystick a few times. This method only requires one wcalibratejoystick call.

If you want to calibrate the joystick in a setup program, you can save all the calibrated values in a configuration file, and load them back in for your main program. This way the user only has to calibrate the joystick once.

Here is the data structure you must save:

```
typedef struct {
   short x, y;
   short cenx, ceny;
   short port, buttons;
   short xrange, yrange;
   short scale;
} joystick;
```

Once the joystick has been calibrated, you must set the scale to the maximum absolute value you wish to receive. For example, if you set the scale to be 2000, the value returned by wreadjoystick will be between -2000 and +2000.

Parameters: joy - Pointer to the joystick information data structure.

Return Value: 1 if the joystick was calibrated 0 if the joystick was not calibrated.

See Also: wcheckjoystick, winitjoystick, wreadjoystick

Examples: 50

wcheckjoystick

Function: Determines which joysticks are connected.

Declaration: short wcheckjoystick (void)

Remarks: This routine checks for the presence of each joystick. It returns an integer with 2 bits set. If bit 1 is set, joystick 1 has been found. If bit 2 is set, joystick 2 has been found.

Parameters: None

Return Value: 0 = no joysticks found
1 = joystick 1 found
2 = joystick 2 found
3 = both joysticks found

See Also: wcalibratejoystick, winitjoystick, wreadjoystick

Examples: 50

winitjoystick

Function: Initializes a joystick.

Declaration: void winitjoystick (joystick *joy, short joynum)

Remarks: Initializes joystick number joynum using the joystick structure joy. This assumes the joystick is centered when you call the routine. joynum can be either 0 (for joystick 1) or 1 (for joystick 2).

You should calibrate the joystick after this routine.

Parameters: joy - Pointer to the joystick information structure.

joynum - Indicates joystick number to initialize.

Return Value: None

See Also: wcalibratejoystick, wcheckjoystick, wreadjoystick

Examples: 50

wreadjoystick

Function: Reads the information from a joystick.

Declaration: short wreadjoystick (joystick *joy)

Remarks: This routine reads the coordinates of the stick and button state and stores them in the joystick structure.

joy.x is the x coordinate
joy.y is the y coordinate
joy.buttons is the button status.

The buttons flag uses bits to determine which buttons are pressed, similar to the mouse button routines.

Parameters: joy - Pointer to the joystick information structure.

Return Value: 1 if successful, 0 if not successful

See Also: wcalibratejoystick, wcheckjoystick, winitjoystick

Examples: 50

WGT3D_WC.LIB

WGT 5.1 features a simple, but effective library of routines for rotating a set of points in all 3 dimensions. When used in combination with the polygon and triangle routines in the main library, this system can produce excellent wireframe and solid polygon-based animations.

winit3d

Function: Initializes the WGT 3D system.

Declaration: void winit3d (void)

Remarks: This command must be called in your program once, before the wrotatepoints command is used. It creates a sine and cosine lookup table that is used for 3D rotations. WGT's 3D rotation library is meant for simple applications only. The main limitation is that you cannot change the camera's location within the 3D coordinate system. You can move 3D objects around by changing their center point however.

Parameters: None

Return Value: None

See Also: wrotatepoints, wsetrotation

Examples: 54, 55

wrotatepoints

Function: Rotates a set of three dimensional points.

Declaration: void wrotatepoints (point3d *orig_pointlist,
point3d *rotated_pointlist, short maxpoint)

Remarks: The wrotatepoints command will take the array of 3D points called orig_pointlist, with maxpoint points, and rotate them around their axis. It stores the rotated points in the rotated_pointlist array. The rotation amounts are set with the command wsetrotation.

The point3d structure is defined as:

```
typedef struct
{
  short x, y, z;
  short sx, sy;
} point3d;
```

sx, and sy are used for storing colors and texture bitmap offsets when using wtexturedpoly and wgouraudpoly.

In addition, the following variables are used in WGT's 3D system:

```
int sx, sy, sz;                                  /* Scale Factors */
short move_x, move_y, move_z;     /* Translational Offsets
                                  - Added to each point after  rotation */
short origin_x, origin_y, origin_z;    /* Point of origin.  All
points will                       be rotated around this point */
```

Parameters: pointlist - Pointer to array of original point3d structure.

rotated_pointlist - Pointer to array of rotated points.

maxpoint - Number of points in the original buffer.

Return Value: None

See Also: winit3d, wsetrotation

Examples: 54, 55

wsetrotation

Function: Sets the rotational amount in each axis that all points will be rotated by.

Declaration: void wsetrotation (short rotate_x, short rotate_y, short rotate_z)

Remarks: The wrotatepoints routine rotates all the points given to it by the amount set with this command. rotate_x, rotate_y and rotate_z can range between 0 and 360.

For example, to rotate the points by 30 degrees in the x axis:
wsetrotation (30,0,0);
wrotatepoints (&stpoint, &finpoint, 4);

Parameters: rotate_x - Degrees to rotate in the x-axis.

rotate_y - Degrees to rotate in the y-axis.

rotate_z - Degrees to rotate in the z-axis.

Return Value: None

See Also: winit3d, wrotatepoints

Examples: 54, 55

WFILE_WC.LIB (FULL SCREEN OS/2 ONLY)

A graphical file selector is the easiest way to select files from large directory lists. This library contains a single routine which creates this selector. The file selector is capable of being moved to a different position on the screen by the user, and it provides full access to all available floppy disks and hard drives on the system. File listings are sorted alphabetically, with the drives and subdirectories at the top of each list. Programs are required to load the font file "LITTLE.WFN" in order to properly display this selector. Larger fonts will not work properly with the display. Refer to the example programs to see how this works.

wfilesel

Function: Activates the WGT file selector routine.

Declaration: char *wfilesel (char *mask, char *title, short x, short y, block image)

Remarks: This routine displays a graphical file selector, and allows the user to scroll up and down through the file listing, change drives and directories, and pick a file, by using the mouse. This is the same file selector used in the Sprite Editor and Map Maker.

mask contains a file extension mask, such as "* " or "spr". It will automatically add a "*." to the file mask. It controls what kind of files will be listed in the file selector when first activated. The user can change this from within the selector if needed. It must be 3 characters long or the routine will always return NULL.

title contains the text which is displayed at the top of the selector. This is used to inform what file operation will be performed on the chosen file.

(x,y) is the coordinate where the selector will be drawn.
It must fit completely on the screen. The user can move the file selector around, if you give a background screen, by clicking and holding on the title bar of the selector.

image is a full screen block which will be used to restore the background when moving the file selector. If image is NULL, the selector will be fixed at the (x,y) coordinates you pass, and it will not erase itself.

The file selector returns a pointer to a string containing the name of the file selected. If no file was chosen, it returns NULL.

Example usage:

```
char *filename;
...
filename = wfilesel("spr", "Load a sprite file", 10, 10, back);
printf ("filename was: %s", filename);
free (filename);            /* Free the filename once we're finished with it */
```

Parameters: mask - File extension to list as default.

title - Title for file selector.

x - Horizontal coordinate of selector on screen.

y - Vertical coordinate of selector on screen.

image - Pointer to buffer for background preservation.

Return Value: Pointer to the string which contains the filename.

Examples: 47

WSCR_WC.LIB

The most popular games in the past few years have all followed the same trend. Graphics are created using a set of repeated "tiles" to simulate a background. The tiles are placed on a map which is larger than the visual screen and this map is scrolled to show the various regions requested.

In the WGT system, the background tiles are created with the WGT Sprite Editor and may be any size up to 64*64 pixels. The tiles do not need to be square, although map design is simpler with square tiles. The WGT Map Maker is then used to create the map itself. On each map, up to 2000 sprites ("objects") may be placed in their starting positions. Each tile may be assigned a number indicating some property or value that the program may interpret. Maps may be up to 320*200 tiles in size. This results in a maximum pixel resolution of 20480*12800; a size much larger than any available video mode supports at the current time (thus the scrolling).

Maps may be superimposed on each other to produce an effect known as "parallax". Each map may be scrolled independently of the other maps which are loaded. Up to 50 maps may be loaded into memory at any given time.

Full source code for this library is provided so that you may customize the library limitations to suit your program's needs. We are not responsible for the use of this code once it has been altered in any way.

is_in_window

Function: Determines if the coordinate given is inside the scrolling window.

Declaration: short is_in_window (short currentwindow, short x, short y,
short range)

Remarks: This command will see if the coordinate (x,y) is within the scrolling window currentwindow. It is commonly used for playing sound effects triggered by a certain action. Since it would be annoying for sound to play because of an action happening somewhere else on the map, is_in_window comes in handy for this. range is a distance (in pixels, always >0) to extend the window's dimensions. A range of 0 would return 1 if the coordinate was exactly within the window. A value of 50 would increase the area to check by 50 pixels on each side.
This is used in case the point is close to the window and the window may scroll over it in the next few frames.

Parameters: currentwindow - Scrolling window containing the map.

x - World X coordinate to check.

y - World Y coordinate to check.

range - Number of pixels of to expand the test window by.

Return Value: 0 if the coordinate is outside the window (plus the range) 1 if the coordinate is inside the window (plus the range)

Examples: 62, 63

soverlap

Function: Detects collisions between two scrollsprites.

Declaration: short soverlap (short s1, scrollsprite *wobjects1,
block *sprites1, short s2,
scrollsprite *wobjects2, block *sprites2)

Remarks: This command checks for collisions between two scrollsprites. It uses a quick rectangular region check which sees if the bounding rectangles of the sprites overlap. This means the collision is not pixel accurate, but in most computer games, exact collisions are not required. If a collision is found, it will return 1, otherwise it returns 0.

If one of the scrollsprites is turned off (ie. scrollsprite[i].on == 0) then no collision is found.

soverlap is usually used in a loop where ranges of sprites are checked for collisions between each other.

Parameters: s1 - Scrollsprite number of sprite 1

wobjects1 - Pointer to the array of scrollsprites for sprite 1

sprites1 - Pointer to the images used for sprite 1

s2 - Scrollsprite number of sprite 2

wobjects2 - Pointer to the array of scrollsprites for sprite 2

sprites2 - Pointer to the images used for sprite 2

Return Value: 0 if the sprites are not overlapping
1 if the sprite are overlapping

See Also: wshowobjects

Examples: 58, 59

wcopymap

Function: Makes a scrolling window share a map with another window.

Declaration: void wcopymap (short sourcewindow, short destwindow)

Remarks: This command allows you to have two scrolling windows on the screen at once which use the same map.
This is perfect for two player games. Any changes made to the map with wputworldblock update both windows at the same time. Note that you can have as many windows sharing a map as you want. This is also useful for creating a reduced view map which scrolls along with a larger one.

Parameters: sourcewindow - Window containing loaded map.

destwindow - Window to assign duplicate map.

Return Value: None

Examples: 59, 67

wendscroll

Function: Releases memory used for a scrolling window.

Declaration: void wendscroll (short currentwindow)

Remarks: wendscroll deallocates the memory used to store a scrolling window's buffers. It must be called to shut down the scrolling system.

Parameters: currentwindow - Scrolling window to close.

See Also: winitscroll

Examples: 56, 57, 58, 59, 62, 63, 65, 67

wfreemap

Function: Releases memory used to store a map file.

Declaration: void wfreemap (wgtmap mapname)

Remarks: When loading a new map file, you must first free the memory used by the previous map before you use the same pointer.

Parameters: mapname - Pointer to the map in memory.

Return Value: None

See Also: wloadmap

Examples: 56, 57, 58, 59, 62, 63, 65, 67

wgetworldblock

Function: Returns the tile number at the given location in a map.

Declaration: unsigned short wgetworldblock (short currentwindow, short posx,
short posy)

Remarks: wgetworldblock is used for detecting collisions between sprites and different parts of the map. posy and posy are world coordinates, so you may pass it the coordinates of a scrollsprite and add a pixel offset if needed.

It is essential for determining when the scrollsprites should react with their surrounding map.

Parameters: currentwindow - Scrolling window of containing the map.

posx - X coordinate of tile. (in world coordinates)

posy - Y coordinate of tile. (in world coordinates)

Return Value: Tile number at the location given.

See Also: wgetworldpixel, wputworldblock

Examples: 58, 59, 62, 63

wgetworldpixel

Function: Returns the color of a pixel within a map.

Declaration: unsigned char wgetworldpixel (short currentwindow, short x,
short y)

Remarks: wgetworldpixel will return the color of the pixel at (x,y) within the map. x and y are world coordinates. This command can be used for pixel-precise collision detection if you "color code" your tiles.

Parameters: currentwindow - Scrolling window containing the map.

x - World X coordinate of pixel.

y - World Y coordinate of pixel.

Return Value: Color of the pixel at the location given.

See Also: wgetworldblock

Examples: None

winitscroll

Function: Initializes one of the four scrolling windows.

Declaration: void winitscroll (short currentwindow, short mode, short link,
short windowwidth, short windowheight,
block *tileref)

Remarks: The WGT scrolling library has the ability to control 50 scrolling windows at a time. This command will initialize one of these windows. currentwindow is the window to initialize from 0 to 49. mode is either NORMAL (0) or PARALLAX (1). NORMAL windows are used as a base window meaning other PARALLAX windows may be shown overtop it. PARALLAX windows are like the xray mode in wputblock where color 0 is considered to be see-through.

link is used for PARALLAX windows only. To use a PARALLAX window, you must also have a NORMAL window which is shown underneath. link is the scrolling window number of this NORMAL window. Also note that you may have more than one NORMAL scrolling window.

windowwidth is the width (in tiles) of the scrolling window. windowheight is the height (in tiles) of the scrolling window. Both of these variables must be at least 1. The maximum value for these variables depends on the size of the tiles being used. When using a PARALLAX window, you must make sure the NORMAL window and the PARALLAX window have the same window dimensions in pixels. If you are using different sizes of tiles for each window, it may be impossible to achieve this. To calculate the window dimensions in pixel, multiply the size of the tiles by the size of the window.

Example of a correct scrolling window setup:
NORMAL window: tile size is 16x16, window size is 20x12 tiles
window width = 320 , window height = 192 (in pixels)
PARALLAX window: tile size is 40x32, window size is 8x6 tiles
window width = 320 , window height = 192 (in pixels)

tileref is a pointer to the block array containing the tiles. winitscroll looks at the first tile in this array which isn't NULL to obtain the dimensions of the tiles.

Parameters: currentwindow - Scrolling window to initialize (0-49).

mode - Window mode : NORMAL or PARALLAX.
link - Number of related NORMAL window, only used if the mode is PARALLAX.

wwidth - Width of window (in tiles).

wheight - Height of window (in tiles).

tileref - Pointer to the array of tile images.

Return Value: None

See Also: wloadmap, wscrollwindow, wshowwindow

Examples: 56, 57, 58, 59, 62, 63, 65, 67

wloadmap

Function: Loads a tiled map into a scrolling window.

Declaration: wgtmap wloadmap (short currentwindow, char *filename,
short *tiletypes, scrollsprite *wobjects)

Remarks: Maps that are created with the WGT Map Maker can be loaded into memory with this command. currentwindow can be a number from 0 to 49, which defines which scrolling window to load the map into. Each scrolling window may have different maps and use different tiles. filename is the name of the WGT map file (.WMP extension) to load.

tiletypes is a pointer to an array of 256 short ints which contains a number for each tile. These numbers can be used to group tiles into certain categories, such as solid or hollow.

wobjects is a scrollsprite structure to load the positions and sprite numbers of the objects in the map. wobjects must be large enough to hold all the objects. For example if the last object number used in a map is 678, you must define wobjects as:

```
scrollsprite wobjects[679];
```

Global Variables Set:
mapwidth[currentwindow] contains the width of the map in tiles.
mapheight[currentwindow] contains the height of the map in tiles.

Parameters: currentwindow - Scrolling window to load the map into (0-49).

filename - Filename of the map to load.

tiletypes - Pointer to an array of 256 short integers
used for storing tiletypes.

wobject - Pointer to an array of scrollsprites.

Return Value: A pointer to the WGT map in conventional memory. NULL is returned map could not loaded due to not enough memory, or the file could not be found.

See Also: wfreemap

Examples: 56, 57, 58, 59, 62, 63, 65, 67

wputworldblock

Function: Changes the tile number at the given location on a map.

Declaration: void wputworldblock (short currentwindow, short posx,
short posy, unsigned short tilenum)

Remarks: wputworldblock is used for modifying the map while displaying it in a window. posx and posy are the map coordinates of the tile, and tilenum is the new tile number.

Parameters: currentwindow - Scrolling window of containing the map.

posx - X coordinate of tile.

posy - Y coordinate of tile.

tilenum - New tile number

Return Value: None

See Also: wgetworldblock

Examples: 58, 59, 62, 63

wsavemap

Function: Saves a WGT map stored in memory into a file.

Declaration: void wsavemap (short currentwindow, char *filename,
wgtmap savemap, short *tiletypes,
scrollsprite *wobjects, short numobj)

Remarks: This command will save the current map in the given scrolling window, along with all of the scrollsprite positions and tiletypes. currentwindow is the scrolling window containing the map to save. filename is the map file to create. savemap is the wgtmap pointer which points to the map data. tiletypes is an array of 256 short integers for the tile types.

wobjects is an array of scrollsprites holding the information about the sprites in the map. numobj contains the size of the wobjects array.

Parameters: currentwindow - Scrolling window containing the map to save.

filename - Filename of the map file to save.

savemap - Pointer to the map in memory.

tiletypes - Pointer to the array of tile types. (256 short integers)

wobjects - Pointer to the array of scrollsprites.

numobj - Size of the wobjects array.

Return Value: None

See Also: wfreemap, wloadmap
Examples: 56

wscreen_coordx

Function: Returns the absolute X screen coordinate of a world coordinate.

Declaration: short wscreen_coordx (short currentwindow, short xcoord)

Remarks: This command is used to find out if a sprite is on the screen and where it appears. It can be used for sound effects panning or to find out the scrolling speed of a window.

If returned coordinate is between 0 and the window width, it is within the window.

Parameters: currentwindow - Window to base coordinate system.

xcoord - World coordinate on map.

Return Value: Screen coordinate of the object.

See Also: wscreen_coordy

Examples: 67

wscreen_coordy

Function: Returns the absolute Y screen coordinate of a world coordinate.

Declaration: short wscreen_coordy (short currentwindow, short ycoord)

Remarks: This command is used to find out if a sprite is on the screen and where it appears. It can be used for sound effects panning or to find out the scrolling speed of a window.

Parameters: currentwindow - Window to base coordinate system.

ycoord - World coordinate on map.

Return Value: Screen coordinate of the object.

See Also: wscreen_coordx

Examples: 67

wscrollwindow

Function: Scrolls a window by a distance vertically and horizontally.

Declaration: void wscrollwindow (short currentwindow, short windowspeedx,
short windowspeedy)

Remarks: wscrollwindow scrolls the given window by a number of pixels in any direction. It must be called continuously to update the scrolling window, even if the scrolling distances are 0.

windowspeedx is the number of pixels to move in the x direction. A negative value for windowspeedx will scroll to the left.

windowspeedy is the number of pixels to move in the y direction. A negative value for windowspeedy will scroll upwards.

When using parallax scrolling with many layers, the NORMAL layer must be the first window to be scrolled/drawn. Each parallax layer will be added to the image from back to front when you use wscrollwindow to draw it. Since each layer is constructed separately, it easy to change the order of the parallax layers, and draw sprites between layers.

Parameters: currentwindow - Scrolling window of the map to scroll (0-49).

windowspeedx - Number of pixels to scroll in X direction.

windowspeedy - Number of pixels to scroll in Y direction.

Return Value: None

See Also: winitscroll, wshowwindow

Examples: 56, 57, 58, 59, 62, 63, 65, 67

wshowobjects

Function: Displays a range of scrollsprites on the scrolling window.

Declaration: void wshowobjects (short currentwindow, short start, short end,
block *image_array, scrollsprite *wobjects)

Remarks: wshowobjects provides an easy way to display your scrollsprites over the scrolling background. start and end refer to the starting and ending indexes of the scrollsprite.

array wobjects. All sprites within this range will be drawn from the lowest index to the highest. Lower indexes will are therefore drawn behind higher indexes. You can also change the order the sprites are drawn by splitting your sprites into ranges, and draw them with a number of calls to wshowobjects. Sprites may be placed behind a PARALLAX layer by showing them before the layer is drawn with the wscrollwindow command.

Parameters: currentwindow - Scrolling window containing the map.

start - Number of first scrollsprite to show.

end - Number of last scrollsprite to show.

image_array - Pointer to the array of images for sprites.

wobjects - Pointer to the array of scrollsprites.

Return Value: None

See Also: soverlap, wscrollwindow

Examples: 56, 58, 59, 62, 63, 67

wshowwindow

Function: Sets the location of the world within a scrolling window.

Declaration: void wshowwindow (short currentwindow, short posx, short posy)

Remarks: wshowwindow is used to set where the window will be located within the map. It is called once before wscrollwindow to set the initial viewing coordinates, and may be used while

scrolling to instantly change the viewing coordinates. posy and posy are the coordinates, in tile measurements, of the new viewing coordinates.

Parameters: currentwindow - Scrolling window to set viewpoint.

posx - Initial X coordinate in map. (in map coords)

posy - Initial Y coordinate in map. (in map coords)

Return Value: None

See Also: winitscroll

Examples: 56, 57, 58, 59, 62, 63, 65, 67

WMENU_WC.LIB

User interfaces can be so confusing at times, and so helpful in others. This library provides a set of routines for providing dropdown menus (such as used in a windowing system). For examples of this library in action, take a look at the Map Maker which is included with this release of WGT. The program uses the dropdown menu system as the core decision maker of the program logic flow.

Full source code for this library is provided so that you may customize the menu limitations to suit your program's needs. We are not responsible for the use of this code once it has been altered in any way.

checkmenu

Function: Polls the mouse until the user clicks a button, and returns which menu choice was selected.

Declaration: short checkmenu (void)

Remarks: checkmenu loops until the user selects a choice from the menu bar or clicks the mouse button outside of the menu bar. If a selection is made, it returns the number of the menu choice.

The return value is a combination of the menu bar title and the option within that bar. For each menu bar, multiply by 10, and add the choice within the menu.

For example, if you selected the menu below, it would return the number 11.
dropdown[1].choice[1]=" Save ";

Mathematically:
result = menu*10 + choice
= 1*10 + 1
= 11

If the mouse is clicked outside the menu, it returns -1.

Parameters: None

Return Value: The number of the menu choice selected, or -1 if the mouse was clicked elsewhere.

See Also: initdropdowns, removemenubar, showmenubar

Examples: 40

initdropdowns

Function: Initializes the drop down menu system.

Declaration: void initdropdowns (void)

Remarks: You must call this after you have defined the menu bar and drop down menus. If you change the text in the menus, such as toggling choices, you may need to call this again to get the correct sizes for the drop down menus.

Parameters: None

Return Value: None

See Also: checkmenu, removemenubar, showmenubar

Examples: 40

removemenubar

Function: Turns off the menu bar at the top of the screen.

Declaration: void removemenubar (void)

Remarks: This command restores the screen underneath the menu bar and shuts down the menu system.

Parameters: None

Return Value: None

See Also: checkmenu, initdropdowns, showmenubar

Examples: 40

showmenubar

Function:	Displays the menu bar at the top of the screen.
Declaration:	void showmenubar (void)
Remarks:	This will activate the menu you have defined, and display the menu bar at the top of the screen.
Parameters:	None
Return Value:	None
See Also:	checkmenu, initdropdowns, removemenubar
Examples:	40

WSPR_WC.LIB

Some of the simplest and most entertaining games are created with the use of animated characters over a non-changing background. This library of routines provides you with the ability to animate images created with the WGT Sprite Editor. The full range of motion, animation and object collision is provided through these routines.

This library is NOT compatible with the WSCR_WC.LIB file. This library can not be used to produce animations over a tiled background created with our other utilities. Limitations to the number of "sprites" onscreen and the specialization of the routines themselves has prompted us to provide full source code for this library. Feel free to alter the code and recompile the library file to meet your own personal needs.

We are not responsible for the use of this code once it has been altered in any way.

animate

Function: Sets the animation sequence for a sprite.

Declaration: void animate (short spritenum, char *animation_sequence)

Remarks: animate sets up animation for a sprite. Spritenum is the sprite number to animate. The animation_sequence string takes the following form:

"(arrnumber,delay)(arrnumber2,delay2)...(arrnumberN,delayN)R"

Between each set of brackets is the information required to animate the sprite. The first number is the sprite array number, and the second is how long that sprite number is displayed. You then make another set and use different sprite numbers and delays. If you place a CAPITAL R at the end of the string, the animation will repeat itself when it gets to the end.
If you leave the R out, the animation will occur once, and stop. You can have up to 40 sets in the animation sequence.

For example:
To animate sprite number one between two sprites, and repeat:

```
Animate(1,"(1,0)(2,1)R");
```

After calling this procedure, you must turn the animation on with animon.

Parameters: spritenum - Sprite number to animate.

animation_sequence - Pointer to the animation control string.

Return Value: None

See Also: animoff, animon

Examples: 22, 23, 25, 61

animoff

Function: Freezes animation for the sprite.

Declaration: void animoff (short spritenum)

Remarks: animoff will temporarily turn off the animation sequence for the given sprite. animation will resume from the point it stopped when animon is called.

Parameters: spritenum - Sprite numver to freeze animation.

Return Value: None

See Also: animate, animon

Examples: 25

animon

Function: Turns animation on for a sprite.

Declaration: void animon (short spritenum)

Remarks: Animation sequences are set with the animate command. To activate the sequence, the sprite's animation must be turned on. Each time draw_sprites is called, the animation will advance by one frame, or delay unit.

Parameters: spritenum - Sprite number to turn on animation.

Return Value: None

See Also: animate, animoff

Examples: 22, 23, 25, 61

deinitialize_sprites

Function: Deinitializes the sprite system.

Declaration: void deinitialize_sprites (void)

Remarks: Before quitting your program, or calling initialize_sprites again, you must first shut down the sprite system with deinitialize_sprites. This will free the memory used by the sprites on the screen.

Parameters: None

Return Value: None

See Also: initialize_sprites

Examples: 22, 23, 25, 60, 61

draw_sprites

Function: Draws the sprites and updates movement and animation counters.

Declaration: void draw_sprites (int movement_multiplier)

Remarks: draw_sprites will first update the animation and movement of all the sprites, and then draw them in the correct location on the screen. movement_multiplier is the number of times to perform the movement and animation for each sprite. This can used so the sprites move at a constant speed regardless of how fast the computer is at drawing them.

Parameters: movement_multiplier - Number of times to move and animate the sprites

Return Value: None

See Also: erase_sprites

Examples: 22, 23, 25, 60, 61

erase_sprites

Function: Erases the sprites from the screen.

Declaration: void erase_sprites (void)

Remarks: erase_sprites will remove the sprites from the virtual screen spritescreen, in preparation for drawing them in a new location. They will not disappear from the visual screen. To remove a sprite from the visual screen requires you to call erase_sprites, turn the sprite off using spriteoff, and call draw_sprites which updates the screen.

Parameters: None

Return Value: None

See Also: draw_sprites

Examples: 22, 23, 25, 60, 61

initialize_sprites

Function: Initializes the sprite library.

Declaration: void initialize_sprites (block *sprite_blocks)

Remarks: initialize_sprites sets up some internal variables, and makes two virtual screens called spritescreen and backgroundscreen.

It assumes you have loaded sprites into an array called sprite_blocks.

You must call initialize_sprites after you load in the sprites using wloadsprites. If you need to call thismore than once in a program, you must use the deinitialize_sprites command first.

Parameters: sprite_blocks - Array of images used for all sprites

Return Value: None

See Also: deinitialize_sprites

Examples: 22, 23, 25, 60, 61

movex

Function: Sets the horizontal movement of a sprite.

Declaration: void movex (short spritenum, char *movement_sequence)

Remarks: movex works similar to animate only there are 3 numbers in each set. You can have up to 15 sets in the movement sequence.

movex (3, "(1,50,1)(-1,50,0)R");

The first number in the set is added to the current x coordinate of the sprite. Therefore if the sprite is at 100,50 on the screen, and you make the number a 5, the sprite will move to 105,50. This number can be anything, so you can have the sprite move right (positive numbers), left (negative numbers) or nowhere (zero).

The second number in the set is the number of times to move the sprite before going on to the next set. It must always be greater then 0.

The third number in the set is the delay for each time the sprite moves. Try using 0 at first and increase it to slow the sprites down.

If the set was (1,50,1), then the sprite would move to the right one pixel, 50 times, with a delay of 1 for each move. Then the next set would be activated. If you have a CAPITAL R at the end of the string, the movement will repeat.

For example, to make sprite 2 move back and forth on the screen continuously:

movex(2,"(1,200,0)(-1,200,0)R");

To make a sprite move from the left side to the right, and jump back to the left again:

movex(2,"(1,319,0)(-319,1,0)R");

Parameters: spritenum - Sprite number to set movement.

movement_sequence - Movement sequence string.

Return Value: None

See Also: movexoff, movexon

Examples: 22, 25, 61