GSLIB 0.51
Reference Manual

Design and Development by A. Lee (aka Hiryu)
gsPipe module partially based upon ‘gfxPipe’
by ‘vzzrzzn’ and ‘sjeep’

Reference Manual by A. Lee (aka Hiryu)

Version	Comments
0.50	Initial public release
0.51	Updated for gcc 3.2.2 & ps2lib 2.1 Fixed texture functions so don’t need to left-shift texture co-ords Added TextureDownload function

Table of Contents

Introduction

The aim of this library is to provide a simple and clean interface to enable the user to setup and draw to the PS2's graphics subsystem.

The library provides three distinct modules: -

gsDriver
Provides functionality for initialising and maintaining the drawing environment, such as setting up the display parameters, updating the display buffers, multi-buffer support and more.

gsPipe
Provides the functionality to draw to the screen, and to change the behaviour of the draw environment by controlling attributes like alpha-blending and scissor areas.

gsFont
Provides a bitmap font handler which allows formatted text to be 'printed' to the screen, and allows the appearance of the text to be customised by colour, bold highlighting, and underlining.

These three modules are documented comprehensively throughout the remainder of this document.

gsDriver Class Definition

The gsDriver class provides the functionality to initialise and maintain the drawing environment.

In addition to supporting the standard double-buffer display used by the PS2 graphics subsystem, the gsDriver class also provides a multi-buffer display enviroment where multiple draw buffers can be setup within the GS memory. This provides a couple of major advantages primarily when either drawing graphically intensive scenes, or when the majority of the vsync period within your code is taken up by other non-drawing activities. The use of multi-buffer displays allows the user code to draw or process a scene that may take longer than one vsync period to draw, without dropping frames, since the code can render ahead on previous less intensive scenes. The number of display buffers that you decide to use will be primarily down to the level of variation in processing time between scenes.

Initialisation Functions

The initialisation functions of the gsDriver class provide simple setup of the drawing and display environment

Constructor

gsDriver::gsDriver()

Description:

The default constructor initialises the draw environment for a 320x240 pixel, 32bit colour, double-buffer display, with Z-Buffer enabled.

These defaults can be changed by using the setDisplayMode() function.

setDisplayMode

```
void gsDriver::setDisplayMode(
                    unsigned int width,
                    unsigned int height,
                    unsigned int xpos,
                    unsigned int ypos,
                    unsigned int psm,
                    unsigned int num_bufs,
                    unsigned int TVmode,
                    unsigned int TVinterlace,
                    unsigned int zbuffer,
                    unsigned int zpsm
                    )
```

Description:

This function returns the readiness state of the drive and filing system.

'width'	should be set to the desired display width in pixels
'height'	should be set to the desired display height in pixels
'xpos'	should be set to the desired horizontal display position
'ypos'	should be set to the desired vertical display position
'psm'	should be set to one of the following values:

GS_PSMCT32
The display will be set to 32bit colour depth (8 bits alpha channel).

GS_PSMCT24
The display will be set to 24bit colour depth (no alpha channel)
.
GS_PSMCT16
The display will be set to 16bit colour depth (1 bit alpha channel)

'num_bufs'	should be set to the desired number of display buffers (default is 2)
'TVmode'	should be set to one of the following values:

GS_TV_AUTO
The display will be set to the default TV format for the console region.

GS_TV_PAL
The display will be set to the PAL TV format
.
GS_TV_NTSC
The display will be set to the NTSC TV format

'TVinterlace' should be set to one of the following values:

GS_TV_INTERLACE

GS _ENABLE
Allocates and enables the ZBuffer

GS_DISABLE
Disables the ZBuffer (and no memory is reserved)

'zbuffer' should be set to one of the following values:

‘zpsm’ should be set to one of the following values:

GS_PSMZ32
The zbuffer will be set to 32bit depth.

GS_PSMZ24
The zbuffer will be set to 24bit depth
.
GS_PSMZ16
The zbuffer will be set to 16bit depth

Informational Functions

The informational functions of the gsDriver class return various data about the current display environment, and also provided are a couple of useful utility functions

getFrameBufferBase

```
unsigned int gsDriver::getFrameBufferBase(
                    unsigned int fb_num
                    )
```

Description:

This function returns the base address of the specified frame buffer.

'fb_num'	should be set to the number (zero-based) of the frame buffer to retrieve the base address of.

The function will return the base address (relative to the start of GS memory) of the specified frame buffer.

getTextureBufferBase

```
unsigned int gsDriver::getTextureBufferBase()
```

Description:

This function returns the base address (relative to the start of GS memory) of the texture buffer.

getCurrentDisplayBuffer

unsigned int gsDriver::getCurrentDisplayBuffer()

Description:

This function returns the number (zero-based) of the frame buffer currently being displayed.

getCurrentDrawBuffer

unsigned int gsDriver::getCurrentDrawBuffer()

Description:

This function returns the number (zero-based) of the frame buffer currently being drawn to.

getBytesPerPixel

static unsigned int gsDriver::getBytesPerPixel(unsigned int psm)

Description:

This function returns the number of bytes per pixel, for the specified display mode.

'psm' should be set to one of the following values:

GS_PSMCT32
The display will be set to 32bit colour depth (8 bits alpha channel).

GS_PSMCT24
The display will be set to 24bit colour depth (no alpha channel)
.
GS_PSMCT16
The display will be set to 16bit colour depth (1 bit alpha channel)

getTexSizeFromInt

static gsTexSize gsDriver::getTexSizeFromInt(int texsize)

Description:

This function returns the gsTexSize (2^X) definition for the specified pixel width/height.
If the pixel width/height is not an exact power of 2, then the nearest texture-size above is returned.

'texsize' should be set to the desired width/height of the texture in pixels

VSync Functions

The VSync functions of the gsDriver class provide functionality for waiting until a vsync, or installing/removing a vsync callback function.

WaitForVSync

void gsDriver::WaitForVSync()

Description:

The function waits until the next vsync occurs, and then returns immediately

AddVSyncCallback

unsigned int gsDriver::AddVSyncCallback(void (*func_ptr)())

Description:

The function registers a vsync callback function, and enables vsync interrupt events (if not already enabled).

The function returns the unique ID of the registered vsync callback function.

<u>Note:</u> Interrupts are disabled whilst the callback function is being called. Callback functions must not use any non-interrupt-safe kernel mode functions, and must re-enable interrupts before returning.

RemoveVSyncCallback

void gsDriver::RemoveVSyncCallback(unsigned int RemoveID)

Description:

The function removes the specified registered vsync callback function.

'RemoveID' should be set to the unique ID returned by AddVSyncCallback

<u>Note:</u> DisableVSyncCallbacks() must be called before, or immediately after removing the last vsync callback.

EnableVSyncCallbacks

void gsDriver::EnableVSyncCallbacks()

Description:

This function enables vsync interrupt events so that vsync callbacks can be called on vsync.
This function is automatically called by AddVSyncCallback, but can be called after temporarily disabling callbacks using DisableVSyncCallbacks().

DisableVSyncCallbacks

void gsDriver::DisableVSyncCallbacks()

Description:

This function disables vsync interrupt events so that vsync callbacks are not called on vsync.
This function may be called to temporarily disable callbacks, which can then be re-enabled using EnableVSyncCallbacks().
This function must be called before, or immediately after, removing the last vsync callback using RemoveVSyncCallback().

Double Frame-buffer Functions

Only a single simple function is provided explicitly for standard double-buffers.

swapBuffers

void gsDriver::swapBuffers()

Description:

This function swaps the current draw and display buffers. This function will normally be called immediately following WaitForVSync() in a double-buffer setup.

Multi Frame-Buffer Functions

GsDriver provides a set of functions specifically for multi frame-buffer environments. These functions make it easy to correctly use multiple frame-buffers.

IsDrawBufferAvailable

bool gsDriver::isDrawBufferAvailable()

Description:

Returns TRUE if a new frame buffer is available for drawing This can be used when polling to see if a vsync callback function has made a new frame-buffer available for drawing.

IsDisplayBufferAvailable

bool gsDriver::isDisplayBufferAvailable()

Description:

Returns TRUE if a completed frame buffer is available for displaying This can by a vsync callback function to see if the mainline code has finished drawing to a frame-buffer, so that it is available for displaying.

setNextDrawBuffer

void gsDriver::setNextDrawBuffer()

Description:

Sets the draw environment to draw to the next free frame-buffer. If no frame-buffer is free for drawing to then the draw environment will not be changed (and if drawing continues then the current frame will be drawn over).

This function will normally be used by the mainline code, immediately after polling isDrawBufferAvailable().

DrawBufferComplete

void gsDriver::DrawBufferComplete()

Description:

This function is called to indicate that drawing to the current draw buffer has been completed, and that the buffer is now available for display.

This function will normally be called immediately after drawing has been completed, and immediately before polling isDrawBufferAvailable() to check if another frame buffer is available for drawing.

DisplayNextFrame

void gsDriver::DisplayNextFrame()

Description:

This function displays the next complete frame buffer, and makes the previously displayed frame available for drawing. If no complete drawn frame buffer is available, then the current frame is still displayed.

This will normally be called by a vsync callback function.

Additional Comments

The gsDriver class also contains a public gsPipe object which is used by the gsDriver class. This public gsPipe object may be used by the user code, without the need for the user code to instantiate its own gsPipe object.

The gsDriver’s public gsPipe object is names ‘drawPipe’ and this may be referenced as shown in the library examples.

gsPipe Class Definition

The gsPipe class provides functionality for drawing to the display initialised by the gsDriver class. Functionality provided by the class includes the drawing of various polygon and other shape types, as well as texture management, and the configuration of the drawing environment.

Initialisation Functions

Constructor

gsPipe::gsPipe(unsigned int size=0x20000)

Description:

The constructor for the gsPipe class creates a buffer or pipe that is used to hold the list of commands to be sent to the PS2's Graphics Subsystem.

The default size of the buffer is 128Kbytes, or 16K quad-word instructions.

'size'	May optionally contain the size of the pipe to create (in bytes).

Copy Constructor

GsPipe::gsPipe(const gsPipe& copyPipe)

Description:

The copy constructor will create a new gsPipe object, which is an exact copy of another, already existing, gsPipe object.

The pipe buffer will be the same size as that of the gsPipe object being copied, and the contents of the buffer, and the internal state of the gsPipe object, will also be duplicated.

This is primarily useful when performing the same base set of draw operations repeatedly, since you can create a single gsPipe object that contains all the common operations, then duplicate it, using either the copy constructor or the assignment operator, before adding the additional operations which may vary between uses.

Assignment operator

gsPipe& gsPipe::operator = (const gsPipe& copyPipe);

Description:

The assignment operation will create a new gsPipe object, which is an exact copy of another, already existing, gsPipe object.

The pipe buffer will be the same size as that of the gsPipe object being copied, and the contents of the buffer, and the internal state of the gsPipe object, will also be duplicated.

This is primarily useful when performing the same base set of draw operations repeatedly, since you can create a single gsPipe object that contains all the common operations, then duplicate it, using either the copy constructor or the assignment operator, before adding the additional operations which may vary between uses.

ReInit

void gsPipe::ReInit()

Description:

The ReInit function must be used to re-initialise the hardware to the state expected by the gsPipe object, after using a different gsPipe object.

getPipeSize

unsigned int gsPipe::getPipeSize(void)

Description:

This operation can be used to get the size of the pipe buffer. It is mostly used for checking that the pipe buffer has been allocated successfully after creating a new gsPipe object.

Flush functions

These functions ‘flush’ the drawing pipe (buffer) to the GS, and forces all drawing primitives currently within the pipe to be drawn to screen.

FlushCheck

void gsPipe::FlushCheck()

Description:

This function checks if the pipe is too full, so that the remaining free space in the pipe is below the minimum threshold, and if so then flushes the drawing pipe.

Flush

void gsPipe::Flush()

Description:

This function flushes the pipe, so that all settings are applied to the GS, and all prims in the pipe are drawn to screen.

This function will normally be called when finished drawing the current frame, but may also be called at other times.

FlushInt

void gsPipe::FlushInt()

Description:

This function is an interrupt-safe version of the Flush() function, so may be called from within an interrupt service routine.

Texture Management Functions

These functions provide operations for uploading textures, and setting the texture to be used when drawing textured primitives.

TextureUpload

```
void gsPipe::TextureUpload(
                unsigned int tbp,
                int tbw,
                int xofs,
                int yofs,
                int pxlfmt,
                char* tex,
                int wpxls,
                int hpxls
                )
```

Description:

This function uploads a texture to the texture buffer.

'tbp'	Texture buffer base address Usually the value returned by gsDriver::getTextureBufferBase()
'tbw'	Width (in pixels) of texture buffer
'xofs'	X offset, in texture buffer, to upload the texture to
'yofs'	Y offset, in texture buffer, to upload the texture to
'pxlfmt'	Should be set to one of the following:

GS_PSMCT32
The display will be set to 32bit colour depth (8 bits alpha channel).

GS_PSMCT24
The display will be set to 24bit colour depth (no alpha channel)
.
GS_PSMCT16
The display will be set to 16bit colour depth (1 bit alpha channel)

'tex'	Address (in EE mem) of the texture pixel-data
'wpxls'	Width (in pixels) of the texture to be uploaded
'hpxls'	Height (in pixels) of the texture to be uploaded

TextureDownload

```
void gsPipe::TextureDownload(
                unsigned int tbp,
                int tbw,
                int xofs,
                int yofs,
                int pxlfmt,
                char* tex,
                int wpxls,
                int hpxls
                )
```

Description:

This function downloads a texture from the texture buffer to EE memory.

'tbp'	Texture buffer base address Usually the value returned by gsDriver::getTextureBufferBase()
'tbw'	Width (in pixels) of texture buffer
'xofs'	X offset, in texture buffer, to upload the texture to
'yofs'	Y offset, in texture buffer, to upload the texture to
'pxlfmt'	Should be set to one of the following:

GS_PSMCT32
The display will be set to 32bit colour depth (8 bits alpha channel).

GS_PSMCT24
The display will be set to 24bit colour depth (no alpha channel)
.
GS_PSMCT16
The display will be set to 16bit colour depth (1 bit alpha channel)

'tex'	Address (in EE mem) of the texture pixel-data
'wpxls'	Width (in pixels) of the texture to be uploaded
'hpxls'	Height (in pixels) of the texture to be uploaded

TextureSet

```
void gsPipe::TextureSet(
                unsigned int tbp,
                int tbw,
                enum gsTexSize texwidth,
                enum gsTexSize texheight,
                u32 tpsm,
                u32 cbp,
                u32 csm,
                u32 cbw,
                u32 cpsm
                )
```

Description:

This function sets up a texture for use when drawing textured prims

'tbp'	Texture buffer base address Usually the value returned by gsDriver::getTextureBufferBase()
'tbw'	Width (in pixels) of texture buffer
'texwidth'	Width of texture in gsTexSize (powers of 2)
'texheight'	Height of texture in gsTexSize (powers of 2)
'tpsm'	Should be set to one of the following:

GS_PSMCT32
The texture is 32bit colour depth (8 bits alpha channel).

GS_PSMCT24
The texture is 24bit colour depth (no alpha channel)
.
GS_PSMCT16
The texture is 16bit colour depth (1 bit alpha channel)

GS_PSMC8
The texture is 8bit colour depth (CLUT used)

'cbp'	Base Address of CLUT Table (in GS mem) – 0 if CLUT not used
'csm'	0 if CLUT not used
'cbw'	Width of CLUT buffer – 0 if CLUT not used

‘cpsm’ Should be set to one of the following – or 0 if CLUT no used

GS_PSMCT32
The texture is 32bit colour depth (8 bits alpha channel).

GS_PSMCT24
The texture is 24bit colour depth (no alpha channel)
.
GS_PSMCT16
The texture is 16bit colour depth (1 bit alpha channel)

Draw Environment Functions

These functions change the behaviour of the drawing environment.

setZTestEnable

```
void gsPipe::setZTestEnable(
                    int enable
                    )
```

Description:

This function enables or disables ZTest. ZTest can only be enabled if the ZBuffer has been configured by the gsDriver.

‘enable’	Should be set to one of GS_ENABLE or GS_DISABLE

setAlphaEnable

```
void gsPipe::setAlphaEnable(
                    int enable
                    )
```

Description:

This function enables or disables Alpha Blending

‘enable’	Should be set to one of GS_ENABLE or GS_DISABLE

setScissorRect

```
void gsPipe::setScissorRect(
                    long x1,
                    long y1,
                    long x2,
                    long y2
                    )
```

Description:

This function sets the GS scissor area, that crops any prim drawing within the display.

'x1'	Should specify the left-hand horizontal position of the scissor rectangle (in pixels)
'y1'	Should specify the top edge vertical position of the scissor rectangle (in pixels)
'x2'	Should specify the right-hand horizontal position of the scissor rectangle (in pixels)
'y2'	Should specify the bottom-edge vertical position of the scissor rectangle (in pixels)

setOrigin

```
void gsPipe::setOrigin(
                    int x,
                    int y
                    )
```

Description:

This function sets the drawing co-ordinates origin of the screen. This is set to 1024,1024 as default. All the primitive drawing functions automatically use this offset when passing co-ordinates to the GS. This allows negative co-ordinates to be used to allow off-screen drawing (and thus clipping) even though the co-ordinate data passed to the GS is unsigned.

This setting should normally be left unchanged.

'x'	Should specify the horizontal drawing origin
'y'	Should specify the vertical drawing origin

setDither

```
void gsPipe::setDither(
                 unsigned long enable
                 )
```

Description:

Enables or disables the GS dithering mode. This should be disabled if the display is set to 24 or 32 bit colour.

'enable'	Should be set to GS_ENABLE or GS_DISABLE

.

setColClamp

```
void gsPipe::setColClamp(
                 unsigned long enable
                 )
```

Description:

Enables or disables colour clamping of the RGB value of pixels.

'enable'	Should be set to GS_ENABLE or GS_DISABLE

setPrModeCont

```
void gsPipe::setPrModeCont(
                 unsigned long state
                 )
```

Description:

Enables whether the PRIM attribute register fields are used from the PRIM register, when drawing each individual primitive, or whether the global settings are used from the PRMODE register.

'enable'	Should be set to GS_ENABLE to enable the settings in the PRIM register, or set to GS_DISABLE to force the global settings from the PRMODE register to be used

Point and Line Drawing Functions

These functions allow the drawing of simple point and line primitives.

Point

```
void gsPipe::Point(
                    int x,
                    int y,
                    unsigned z,
                    unsigned colour
                    )
```

Description:

This function draws a simple coloured pixel at the specified drawing co-ordinates. The pixel will be Z-Sorted if Z-Test is enabled.

'x' & **'y'**	Should be set to the co-ordinates of the pixel to be drawn
'z'	Should be set to the Z depth of the pixel to be drawn
'colour'	Should be set to the desired colour of the pixel.

Line

```
void gsPipe::Line(
                    int x1,
                    int y1,
                    int x2,
                    int y2,
                    unsigned z,
                    unsigned colour
                    )
```

Description:

This function draws a flat coloured line between the specified drawing co-ordinates. The line has a single Z value, so the whole line will be treated as having the same Z-Depth.

'x1' & **'y1'**	Should be set to the co-ordinates of one end of the line to be drawn
'x2' & **'y2'**	Should be set to the co-ordinates of the other end of the line to be drawn
'z'	Should be set the Z depth of the line to be drawn
'colour'	Should be set to the desired colour of the line.

Triangle Drawing Functions

These functions allow the drawing of various types of triangle primitives, including outline, flat coloured, gouraud shaded, and textured.

TriangleLine

```
void gsPipe::TriangleLine(
                int x1, int y1, unsigned z1, unsigned c1,
                int x2, int y2, unsigned z2, unsigned c2,
                int x3, int y3, unsigned z3, unsigned c3
                )
```

Description:

This function draws an outline triangle, with each side of the triangle being drawn in the specified colour. Each corner of the triangle may have a different Z depth.

The following parameter description applies to each of the three sets of parameters:

'x' & **'y'**	Should be set to the co-ordinates of one corner of the triangle
'z'	Should be set to the Z depth of the corner.
'c'	Should be set to the colour of the line starting at the corner.

TriangleFlat

```
void gsPipe::TriangleFlat(
                int x1, int y1, unsigned z1,
                int x2, int y2, unsigned z2,
                int x3, int y3, unsigned z3,
                unsigned colour
                )
```

Description:

This function draws a flat-coloured triangle.

The following parameter description applies to each of the three sets of parameters:

'x' & **'y'**	Should be set to the co-ordinates of one corner of the triangle
'z'	Should be set to the Z depth of the corner.
'colour'	Should be set to the overall colour of the triangle.

TriangleGouraud

```
void gsPipe::TriangleGouraud(
                int x1, int y1, unsigned z1, unsigned c1,
                int x2, int y2, unsigned z2, unsigned c2,
                int x3, int y3, unsigned z3, unsigned c3
                )
```

Description:

This function draws a gouraud-shaded triangle, with each corner of the triangle being drawn in the specified colour. The colour of the triangle is blended between the colours of each corner. Each corner of the triangle may have a different Z depth.

The following parameter description applies to each of the three sets of parameters:

'x' & **'y'**	Should be set to the co-ordinates of one corner of the triangle
'z'	Should be set to the Z depth of the corner.
'c'	Should be set to the colour of the corner.

TriangleTexture

```
void gsPipe::TriangleTexture(
                int x1, int y1, unsigned z1, unsigned u1, unsigned v1,
                int x2, int y2, unsigned z2, unsigned u2, unsigned v2,
                int x3, int y3, unsigned z3, unsigned u3, unsigned v3,
                unsigned colour
                )
```

Description:

This function draws a textured flat-tinted triangle. The texture is copied from the texture currently selected using TextureSet().

The following parameter description applies to each of the three sets of parameters:

'x' & **'y'**	Should be set to the co-ordinates of one corner of the triangle
'z'	Should be set to the Z depth of the corner.
'u' & **'v'**	Should be set to the co-ordinates of one corner of the texture (within the current selected texture)
'colour'	Should be set to the overall colour of the triangle (RGB values of 0x80 will leave the colour of the texture unchanged)

Triangle-Strip Drawing Functions

The following functions allow the drawing of triangle-strips. A triangle strip is (as the name suggests) a strip of triangles, where the first triangle is drawn from the first, second and third coordinates, and the second triangle is drawn from the second, third and fourth coordinates etc.

TriStripGouraud

```
void gsPipe::TriStripGouraud(
                    int x1, int y1, unsigned z1, unsigned c1,
                    int x2, int y2, unsigned z2, unsigned c2,
                    int x3, int y3, unsigned z3, unsigned c3,
                    int x4, int y4, unsigned z4, unsigned c4,
                    )
```

Description:

This function draws a gouraud-shaded rectangle, with each corner of the rectangle being drawn in the specified colour. The colour of the rectangle is blended between the colours of each corner.

The following parameter description applies to each of the four sets of parameters:

'x' & **'y'**	Should be set to the co-ordinates of one corner of the rectangle
'z'	Should be set to the Z depth of the corner.
'c'	Should be set to the colour of the corner.

TriStripGouraudTexture

```
void gsPipe::TriStripGouraudTexture(
                    int x1, int y1, unsigned z1, unsigned u1, unsigned v1, unsigned c1,
                    int x2, int y2, unsigned z2, unsigned u2, unsigned v2, unsigned c2,
                    int x3, int y3, unsigned z3, unsigned u3, unsigned v3, unsigned c3,
                    int x4, int y4, unsigned z4, unsigned u4, unsigned v4, unsigned c4,
                    )
```

Description:

This function draws a textured, gouraud-shaded triangle strip. The texture is copied from the texture currently selected using TextureSet().

The following parameter description applies to each of the sets of parameters:

'x' & **'y'**	Should be set to the co-ordinates of one corner of the triangle
'z'	Should be set to the overall Z depth of the corner
'u' & **'v'**	Should be set to the co-ordinates of one corner of the texture (within the current selected texture)
'c'	Should be set to the colour of the corner of the triangle (RGB values of 0x80 will leave the colour of the texture unchanged)

Rectangle Drawing Functions

These functions allow the drawing of various types of rectangle primitives, including outline, flat coloured, gouraud shaded, and textured.

RectLine

```
void gsPipe::RectLine(
                int x1, int y1,
                int x2, int y2,
                unsigned z,
                unsigned colour
                )
```

Description:

This function draws an outline rectangle, with all sides of the rectangle being drawn in the same colour.

'x1' & **'y1'**	Should be set to the co-ordinates of the top-left corner of the rectangle
'x2' & **'y2'**	Should be set to the co-ordinates of the bottom-right corner of the rectangle
'z'	Should be set to the Z depth of the rectangle.
'colour'	Should be set to the colour of rectangle.

RectFlat

```
void gsPipe::RectFlat(
                int x1, int y1,
                int x2, int y2,
                unsigned z,
                unsigned colour
                )
```

Description:

This function draws a flat-coloured rectangle.

'x1' & **'y1'**	Should be set to the co-ordinates of the top-left corner of the rectangle
'x2' & **'y2'**	Should be set to the co-ordinates of the bottom-right corner of the rectangle
'z'	Should be set to the Z depth of the rectangle.
'colour'	Should be set to the colour of rectangle.

RectTexture

```
void gsPipe::RectTexture(
                    int x1, int y1,
                    u32 u1, u32 v1,
                    int x2, int y2,
                    u32 u2, u32 v2,
                    unsigned z,
                    unsigned colour
                    )
```

Description:

This function draws a textured rectangle (or sprite).

‘x1’ & **‘y1’**	Should be set to the co-ordinates of the top-left corner of the rectangle
‘u1’ & **‘v1’**	Should be set to the co-ordinates of the top-left corner of the texture
‘x2’ & **‘y2’**	Should be set to the co-ordinates of the bottom-right corner of the rectangle
‘u2’ & **‘v2’**	Should be set to the co-ordinates of the bottom-right corner of the texture
‘z’	Should be set to the Z depth of the rectangle.
‘colour’	Should be set to the colour of rectangle (RGB values of 0x80 will leave the colour of the texture unchanged)

RectGouraud

```
void gsPipe::RectGouraud(
                    int x1, int y1, unsigned c1,
                    int x2, int y2, unsigned c2,
                    unsigned z
                    )
```

Description:

This function draws a gouraud shaded rectangle. The colour of the rectangle will be blended between the colours of the opposing corners.

‘x1’ & **‘y1’**	Should be set to the co-ordinates of the top-left corner of the rectangle
‘c1’	Should be set to the colour of the top-left corner of the rectangle
‘x2’ & **‘y2’**	Should be set to the co-ordinates of the bottom-right corner of the rectangle
‘c2’	Should be set to the colour of the bottom-right corner of the rectangle
‘z’	Should be set to the Z depth of the rectangle.

gsFont Class Definition

The gsFont class provides functionality for 'printing' text to the display in the form of bitmap fonts.

The gsFont class provides various text formatting options, including colour, underling, bold highlighting, and text alignment.

The bitmap font format used by the gsFont class is proprietary, but relatively simple. Tools for creating the font file format are provided in the tools section of the library.

The gsFont class must have a gsPipe object assigned to it, before it can be used to display text. The gsPipe object may be one used for other drawing operations, or may be one especially created for the purpose.

Initialisation Functions

The initialisation functions provide methods to construct a gsFontDraw object, and initialise it for use.

Constructor

```
gsFont::gsFont(
                    gsPipe* fontPipe = (gsPipe*)NULL
                    )
```

Description:

The constructor creates an object of the gsFont class. The optional parameter may be used to assign a gsPipe object to the gsPipe object.

'fontPipe' May optionally contain a pointer to a gsPipe object.

assignPipe

```
void gsFont::assignPipe(
                    gsPipe* fontPipe
                    )
```

Description:

This function assigns a gsPipe object to be used by the gsFont object when uploading for printing fonts.

'fontPipe' Should contain a pointer to a gsPipe object.

uploadFont

```
void gsFont::uploadFont(
                    gsFontTex* pFontTex,
                    unsigned int TBbase,
                    int TBwidth,
                    int TBxpos,
                    int Tbypos
                    )
```

Description:

This function uploads the bitmap section of a bitmap font file to the texture buffer, and makes a local copy of the other data contained within the bitmap font file.

'pFontTex'	Should contain a pointer to a bitmap font file (loaded into EE mem)
'TBbase'	Should contain the base address of the GS texture memory (usually the value returned by gsDriver::getTextureBufferBase())
'TBwidth'	Should contain the width of the texture buffer (in pixels)
'TBxpos'	Should contain the horizontal position (within the texture buffer) to upload the font bitmap data to.
'TBypos'	Should contain the vertical position (within the texture buffer) to upload the font bitmap data to.

Font Printing Functions

The following function provides the functionality for ‘printing’ text to the screen.

Print

```
void gsFont::Print(
                int x,
                int Xend,
                int y,
                int z,
                unsigned long colour,
                gsFontAlign alignment,
                const char* string
                )
```

Description:

This function prints the specified text string to the screen at the specified co-ordinates and using the specified formatting.

‘x’	Should be the horizontal position of the left-hand side of the text area.
‘Xend’	Should be the horizontal position of the right-hand side of the text area.
‘y’	Should be the vertical position of the top-edge of the text area.
‘z’	Should be the Z-Depth of the text.
‘colour’	Should be the colour to display the text in (RGB values of 0x80 leave the colours of the font bitmap unchanged)
‘alignment’	Should be one of the following values:

GSFONT_ALIGN_LEFT
The text will be left aligned to the position specified by ‘x’. Any strings that would continue past the position specified by ‘Xend’ are wrapped onto subsequent lines.

GSFONT_ALIGN_CENTRE
The text will be centred between the positions specified by ‘x’ and ‘Xend’. Any strings which would be longer than the distance between ‘x’ and ‘Xend’ are wrapped onto subsequent lines.
.
GSFONT_ALIGN_RIGHT
The text will be right-aligned to the position specified by ‘Xend’. Any strings that are longer than the distance between ‘x’ and ‘Xend’ are wrapped onto subsequent lines.

‘string’	Should contain a pointer to a null-terminated character string. The string may contain additional formatting characters, which are detailed below.

String formatting characters:

‘\n’	Standard ASCII new-line. Any text after a new-line will continue on the next line.
‘\a’	Toggle underlining on/off
‘\b’	Toggle bold highlighting on/off

All other control-characters are currently not supported. (‘tab’ support may be added in future)

File Formats

Below are the details of the proprietary file formats used by the library. Where relevant details are given on how to create these files, or tools are provided to allow the creation of files.

Bitmap Font File

The bitmap font file format used by the gsFont class is detailed below. Tools are provided to aid in the creation of these files, in the tools section of the library.

Offset	Size	Description
0x0000	4 characters	File format header, containing the ASCII characters ‘BFNT’
0x0004	Unsigned 32bit	Width of the bitmap font texture (in pixels)
0x0008	Unsigned 32bit	Height of the bitmap font texture (in pixels)
0x000C	Unsigned 32bit	Pixel Format of the bitmap font texture (using GS_PSM definitions)
0x0010	Unsigned 32bit	Number of characters horizontally in grid (usually 16)
0x0014	Unsigned 32bit	Number of characters vertically in grid (usually 16)
0x0018	Unsigned 32bit	Width of one character block within the grid (in pixels)
0x001C	Unsigned 32bit	Height of one character block within the grid (in pixels)
0x0020	256 bytes	Width of each of the 256 characters (in pixels)
0x0120	Variable	Pixel data of the bitmap font texture (of size and pixel format specified)