LibPSn00b Library Reference

Published date: < to be filled >

Meido-Tek Computer Entertainment Philippines
Non existent address
Non existent city, FU, 6669
Philippines

- This particular page is intended to be a spoof of the official library documents -

Table of Contents

About this Document

About LibPSn00b and the PSn00bSDK project

LibPSn00b is a free and open source implementation of Sony's Runtime Libraries used for developing software for the original PlayStation that also tries to be more efficient than Sony's original libraries with the goal of making the library functions written mostly in MIPS assembly.

LibPSn00b follows most of the original syntax of Sony's libraries for familiarity reasons and to make it easier to port existing homebrew originally written for Sony's SDK over to LibPSn00b with minimal effort, provided the functions and other things it depends on have already been implemented. LibPSn00b is obviously part of the PSn00bSDK project.

The PSn00bSDK project aims to develop a 100% free and open source SDK for the original PlayStation that is as capable as Sony's official SDK in terms of hardware features to open up the possibility of developing homebrew titles that are on par of officially released titles 100% legally.

The PSn00bSDK Project is in no affiliation with Sony Computer Entertainment Inc.

Typographic Conventions

Certain typographic conventions are used throughout this document to clarify the meaning of the text. The following conventions apply in this document:

Convention	Meaning
italic	Function arguments and structure members.
`monospace`	Program code. Usually contained inside of a code block.
bold	Types, structure and function names.
blue	Hyperlink.

Original Code

Functions, structures and macro definitions that are originally not part of the official libraries fall under Original Code and must be indicated as such in both this document and the library headers.

For contributors; original code should never use existing names of the official libraries to avoid compatibility issues when porting existing homebrew originally written for Sony's SDKs.

Credits & Acknowledgments

Library Developers:

Lameguy64

Library Documentation:

Lameguy64

Reference Materials:

Nocash's PlayStation specs document: http://problemkaputt.de/psx-spx.htm

tails92's PSXSDK library source: https://github.com/simias/psxsdk

Chapter 1: Graphics Library

Table of Contents

Overview

The psxgpu library provides functions, structures and macros to initialize, configure and draw graphic primitives using the PlayStation's GPU. It is an essential component to any PlayStation project unless its something that does not show anything on the display but such a project would be pretty ridiculous.

This library does not provide functions for 3D processing, only display and graphics primitives. 3D functions are provided in psxgte instead.

Display and Drawing Environments

The display environment refers to the video mode and display area of the frame buffer to be displayed on the television screen that is set to the GPU. These parameters are usually defined using DISPENV and applied to the GPU using PutDispEnv(). The video standard used can be changed using SetVideoMode().

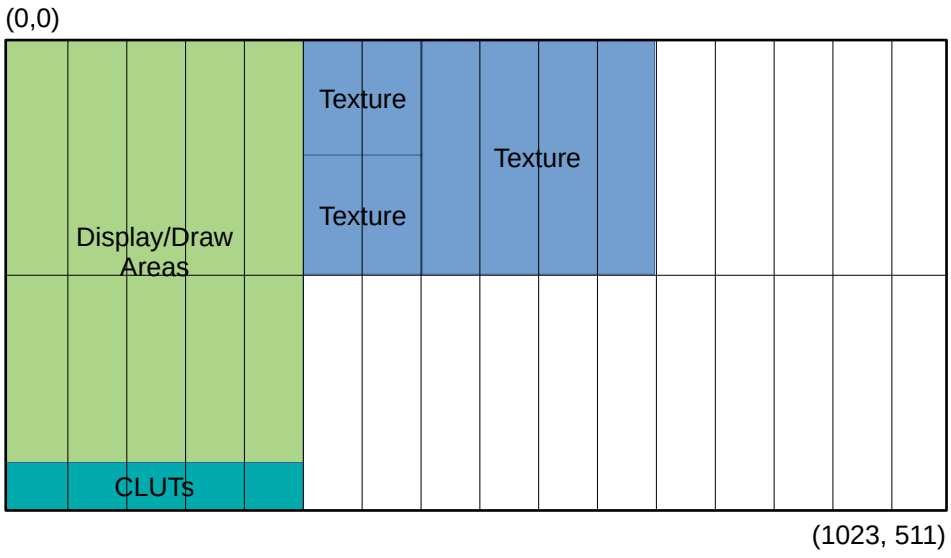
The drawing environment refers to the area of the frame buffer where drawing takes place and the base texture page and texture window coordinates. These parameters are typically defined using DRAWENV and applied to the GPU using PutDrawEnv(). Parameters such as current texture page and drawing area can be altered mid-drawing using appropriate primitives such as DR_TPAGE and DR_AREA.

While the GPU is capable of displaying 24-bit color it can only draw graphics in 16-bit color limiting the usefulness of 24-bit color mode to static images and FMV video sequences.

Video Memory

The PlayStation has 1 megabyte of video memory used for frame buffers, textures and color look-up tables (CLUTs). The video memory is not connected to the memory bus so the only way to transfer data to and from it is through the GPU using LoadImage() and StoreImage() respectively.

The GPU addresses the video memory as a 1024x512 16-bit frame buffer so display, draw and image areas are specified in 2D coordinates. The frame buffer is divided into 64x256 pixel pages but this is most relevant on textures.



Sample layout of display buffers, textures and Clutch in VRAM

Textures & Color Lookup Tables (CLUT)

Textures are bitmap images to be used with sprites and textured polygons. The GPU supports 16-bit texture images as well as color indexed 8-bit and 4-bit textures. Since the VRAM is always treated as a 16-bit 1024x512 frame buffer regardless of the color depth f the texture 8-bit and 4-bit textures will appear at half width and quarter width respectively. (eg. 64x64 texture will be 32x64 in 8-bit and 16x64 in 4-bit)

While the size of the texture does not necessarily have to be in powers of two the width must be at least a multiple of 2 for 16-bit and 8-bit textures and 4 for 4-bit textures to ensure consistent texturing without any graphical corruption around the edges of a sprite or textured polygon.

The pixel format of 16-bit textures, CLUTs and the pixels the GPU draws is RGB555 which can support up to 32,768 unique colors. The 16th bit is used as a semi-transparency mask bit for textures and as a write protection mask to the GPU which can be used as a basic 1-bit stencil mask.

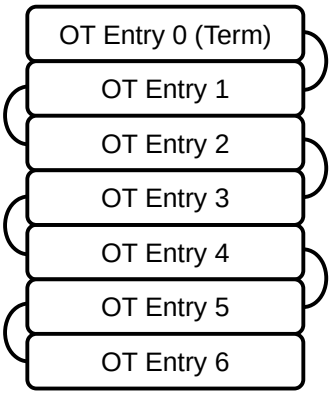
The GPU has a 2KB texture cache which can be taken advantage of by having textures of a specific size which varies depending on the color depth of the texture. For 4-bit textures it is 64x64, for 8-bit textures it is 32x64 and for 16-bit textures it is 32x32. Taking advantage of the texture cache can help improve drawing performance.

The maximum texture resolution sprite and polygon primitives can support is 256x256 on all texture color depths. Larger textures can be drawn with multiple primitives that draws each portion of the texture image.

Color indexed texture formats have the addition of a color lookup table or CLUT which is essentially the color palette of color indexed textures. It appears on the VRAM as a 16-bit 256x1 or 16x1 texture and is positioned on the VRAM much like a texture image.

Ordering Tables

An ordering table is an array of 32-bit integers defined with pointers that point from one element to another forming a chain of pointers. The chain must end with a terminator value of 0x00FFFFFF but ClearOTagR() sets that automatically.



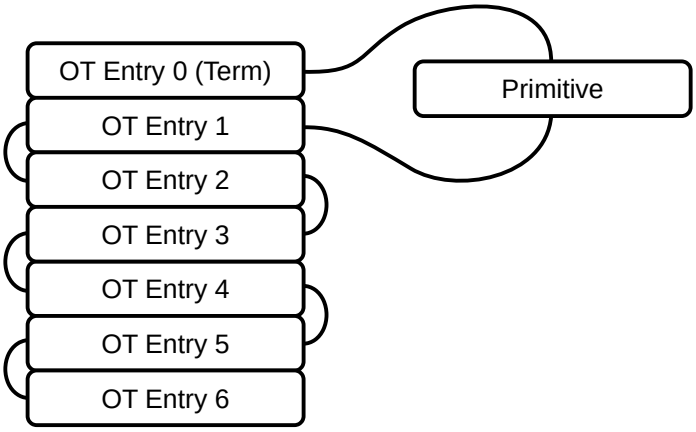
A reverse ordering table chain

A reverse ordering table which is defined using the ClearOTagR() function begins at the last element of the array and ends at the first element. This style of ordering table is most commonly used for 3D graphics as it allows for simplified depth sorting of polygons as primitives sorted to higher elements are drawn first and lower elements are drawn last. This is unofficially known as

blit sort but there doesn't seem to be much information about this sorting algorithm so the exact term is unknown. The table can also be used to control layering of sprite and tile elements in 2D games.

Primitives to be drawn are usually 'sorted' into an element of an ordering table by linking the primitive to the chain using AddPrim(). This pretty much explains why many graphics functions that draw graphics primitives or objects often reference sorting rather than drawing.

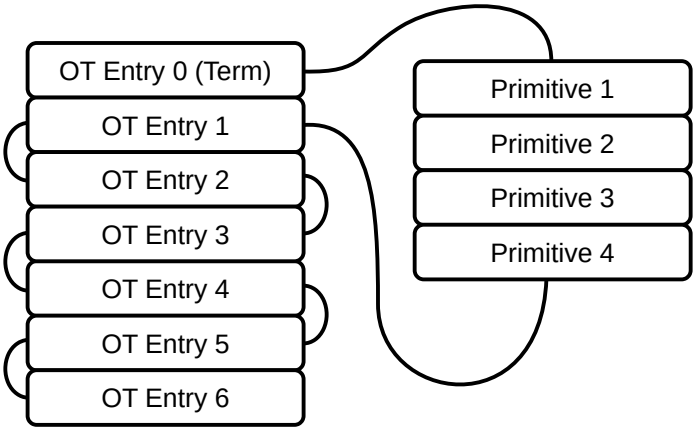
The primitives themselves are usually stored in a separate buffer, usually an array of char elements. Preferably one should additionally define a pointer variable that initially points to the beginning of a primitive buffer and is incremented based on the size of a primitive that had been sorted to use as a reference as to where the next primitive should be stored to and is reset to the start of a buffer on every frame. Unlike the ordering table it does not need to be cleared on every frame.



Ordering table chain with primitive sorted (0)

Multiple primitives may be sorted to a single ordering table element and does not overwrite any previously sorted primitives in the element. There's a common misconception where you can only sort a single primitive on each element and the number of primitives you can sort is limited by the size of your ordering table. Both of these are entirely false as the ordering table mechanism chains previously sorted primitives together and can sort as many primitives to a single element as you'd like.

Remember that primitives sorted first are always processed last as shown in the figure below.



Multiple primitives sorted to the same table entry (0)

Double Buffering

Ideally you'd want to set up your display/drawing areas, ordering tables and primitive buffers in a doubled buffered configuration to achieve seamless animations and better graphics performance.

For the display/draw areas you must reserve two areas of the frame buffer, one for displaying and one for drawing to prevent flicker in the event of a slowdown. This can easily be achieved witha pair of DISPENV and DRAWENV definitions with one pointing to an area to be displayed and the other pointing to an area to be drawn. Either definitions must not overlap each other's areas.

This scheme is only applicable to 240 or 256 line video modes as there's not enough space to contain two high resolution display/draw areas in the frame buffer. In interlaced high resolution modes however, graphics are only drawn to lines that are not yet shown on the display to hide flickering issues in a single buffered display/draw configuration but a consistent 60 FPS for NTSC and 50 FPS for PAL must be achieved to avoid any flicker (well, apart from the flicker of interlacing).

For the ordering tables and packet buffers you must define two pairs of each buffer. This is so that while the GPU is busy processing an ordering table full of primitives the time it takes to wait for the GPU to finish processing can be used to process the next frame but the ordering table the GPU is currently processing cannot be touched while the GPU is busy processing it.

The following page describes a code example is a typical implementation of double buffered display/draw areas, ordering tables and packet buffers.

```
#define OTLEN     128                     // Length of ordering table

DISPENV disp[2];                          // Display environments
DRAWENV draw[2];                          // Draw environments

int ot[2][OTLEN];  // Ordering tables
char pribuff[2][32768]                    // Primitive buffers
char *nextpri;                            // Pointer for next primitive
int db=0;          // Double buffer index

void init() {

    // Define the display/drawing environments
    SetDefDispEnv( &disp[0], 0, 0, 320, 240 );
    SetDefDrawEnv( &draw[0], 0, 240, 320, 240 );

    SetDefDispEnv( &disp[1], 0, 240, 320, 240 );
    SetDefDrawEnv( &draw[1], 0, 0, 320, 240 );

    // Clear the ordering tables
    ClearOTagR( ot[0], OTLEN );
    ClearOTagR( ot[1], OTLEN );
    nextpri = pribuff[0];

}

int main() {

    init();

    while( 1 ) {

        /* sort primitives */

        DrawSync();                       // Wait for GPU and vertical retrace
        VSync();

        PutDispEnv( &disp[db] );          // Apply display/draw environments
        PutDrawEnv( &draw[db] );

        SetDispMask(1);                   // Enable display

        DrawOTag( ot[db]+OTLEN-1 );       // Draw ordering table

        db = !db;// Alternate buffers

        ClearOTagR( ot[db], OTLEN );      // Clear next ordering table
        nextpri = pribuff[db];            // Reset primitive address

    }

    return 0;

}
```

Framebuffer double buffering sample cod

Structures

DISPENV

Display environment

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*12/22/2018*

Structure

```
typedef struct {
    RECT disp;              Frame buffer area to display and display resolution.
    RECT screen;            Picture display position and size on the display.
    unsigned char isinter;  Interlace flag. 0: no interlace (progressive); 1: interlace
    unsigned char isrgb24;  24-bit color mode flag. 0: 16-bit mode; 1: 24-bit color mode
    unsigned char reverse;  'Reverse mode' flag (sets bit 7 to video mode parameters).
                            Doesn't do much when non-zero (varies on GPU version?).
    unsigned char pad;      Padding.
} DISPENV;
```

Explanation

Specifies display parameters and video mode as well as the area of the frame buffer to display on the screen.

disp specifies both the frame buffer coordinates to display and the display resolution. Valid horizontal resolutions are 256, 320, 384, 512 or 640 while vertical resolutions are 240 or 480 for NTSC and 256 or 512 for PAL. Sizes that are not valid are snapped to a valid size.

screen specifies the position and size of the display on the TV screen. *screen.x* and *screen.y* control the X and Y position of the picture while *screen.w* and *screen.h* control the size of the picture. If either *screen.w* or *screen.h* are set to zero a default image size is used. Changing the picture size does not resize the image shown on the screen but rather controls how much pixels are shown on the screen and may be used to set non standard display resolutions.

If you intend to use a PAL video mode that is 256 or 512 lines tall you need to manually set *screen.h* to 272 so all 256 lines will be displayed and set *screen.y* to 10 to center the picture vertically.

Work in progress

screen may behave differently compared to the implementation in the official SDK.

See also

DRAWENV

Drawing environment

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*12/21/2018*

Structure

typedef struct {

RECT *clip;*	Drawing area. Must be within (0, 0) – (1023, 511).
short *ofs[2];*	Drawing offset. ofs[0] and ofs[1] are X and Y respectively.
RECT *tw;*	Texture window. Species offset and size where texture wrapping starts (can be set mid-drawing using DR_TPAGE).
unsigned short *tpage;*	Initial texture page value (can be set mid-drawing using DR_TPAGE).
unsigned char *dtd;*	Dithering enable. 0: off; 1: on (Simply merges an appropriate bit to *tpage*)
unsigned char *dfe;*	Drawing on displayed area enable. 0: No drawing on displayed area (required in interlaced mode). 1: Draw on currently displayed area. (Simply merges an appropriate bit to *tpage*)
unsigned char *r0,g0,b0;*	Drawing area clear color.
DR_ENV *dr_env;*	Packet buffer used by PutDrawEnv() when applying the drawing environment.

} DRAWENV;

Explanation

Used to specify various GPU drawing parameters such as draw area, GPU offset, initial texture window, initial texture page and more.

Work in progress

tw has no effect to the drawing environment.

See also

RECT

Defines a rectangular area

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*12/21/2018*

Structure

```
typedef struct {
    short x,y;          Top left coordinates of the rectangular area.
    short w,h;          Width and height of the rectangular area.
} RECT;
```

Explanation

Used by many functions and structures to define a rectangular area. Coordinates may not exceed (0, 0) – (1024, 512) unless you want to corrupt the frame buffer.

See also

SPRT

Any-size textured sprite primitive

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*05/23/2019*

Structure

typedef struct {

unsigned int *tag;*	Pointer tag to next primitive packet.
unsigned char *r0,g0,b0;code;*	RGB color of sprite + packet code.
short *x0,y0;*	Position of sprite.
unsigned char *u0,v0;*	Sprite texture coordinates within texture page. u*0* must be a multiple of 2.
unsigned short *clut;*	Sprite texture CLUT ID.
unsigned short *w,h;*	Sprite size (w must be a multiple of 2).

} SPRT;

Explanation

Draws a textured sprite primitive of any defined size. Faster than POLY_FT4.

If you use a sprite size greater than 256x256 the texture will repeat. Size can be up to 1023 x 511 pixels.

Because the SPRT primitive has no *tpage* parameter so a DR_TPAGE primitive must be sorted to the ordering table after the SPRT primitive to set desired *tpage* value for the sprite primitive.

SPRT_8, SPRT_16

Fixed size 8 x 8 or 16 x 16 textured sprite.

Library	Header	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*05/23/2019*

Structure

```
typedef struct {
    unsigned int tag;                   Pointer tag to next primitive packet.
    unsigned char r0,g0,b0,code;        RGB color of sprite + primitive code.
    short x0,y0;                        Position of sprite (top-left coordinates).
    unsigned char u0,v0;                Sprite texture coordinates within texture page. u0 must
                                        be a multiple of 2.
    unsigned short clut;                Sprite texture CLUT ID.
} SPRT_8;

typedef struct {
    unsigned int tag;                   Pointer tag to next primitive packet.
    unsigned char r0,g0,b0,code;        RGB color of sprite + primitive code.
    short x0,y0;                        Position of sprite (top-left coordinates)
    unsigned char u0,v0;                Sprite texture coordinates within texture page. u0 must
                                        be a multiple of 2.
    unsigned short clut;                Sprite texture CLUT ID.
} SPRT_16;
```

Explanation

Draws a fixed size 8 x 8 or 16 x 16 pixel textured sprite. Supposedly faster than **SPRT**.

Much like **SPRT** it has no *tpage* parameter so a DR_TPAGE primitive must be sorted to the ordering table after the SPRT primitive to set desired *tpage* value for the sprite primitive.

TIM_IMAGE

Texture Image file parameters

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*02/02/2019*

Structure

```
typedef struct {
    unsigned int *mode;*        Image mode (bit 0-3: color depth, bit 4: CLUT flag).
    RECT *crect;*               Pointer to CLUT rectangle coordinates.
    unsigned int *caddr;*       Pointer to CLUT data (or NULL if no CLUT).
    RECT *prect;*               Pointer to pixel data rectangle coordinates.
    unsigned int *paddr;*       Pointer to pixel data.
} TIM_IMAGE;
```

Explanation

Defines the pointers to VRAM coordinates and pixel data of a TIM image file. Values are to be set by GetTimInfo().

The pointers point directly to relevant parts of the TIM image file.

See also

TILE

Any size flat colored sprite primitive

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*05/23/2019*

Structure

```
typedef struct {
    unsigned int tag;                   Pointer tag to next primitive packet.
    unsigned char r0,g0,b0,code;        RGB color of tile + packet code
    short x0,y0;                        Position of tile (top-left coordinate)
    short w,h;                          Size of tile in pixels.
} TILE;
```

Explanation

Draws a flat colored sprite of specified size.

TILE_1, TILE_8, TILE_16

Fixed size 1 x 1, 8 x 8 and 16 x 16 colored sprites.

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*05/23/2019*

Structure

```
typedef struct {
    unsigned int tag;               Pointer tag to next primitive packet.
    unsigned char r0,g0,b0,code;    RGB color of tile + packet code.
    short x0,y0;                    Position of tile (top-left coordinates).
} TILE_1;

typedef struct {
    unsigned int tag;               Pointer tag to next primitive packet.
    unsigned char r0,g0,b0,code;    RGB color of tile + packet code.
    short x0,y0;                    Position of tile (top-left coordinates)
} TILE_8;

typedef struct {
    unsigned int tag;               Pointer tag to next primitive packet.
    unsigned char r0,g0,b0,code;    RGB color of tile + packet code.
    short x0,y0;                    Position of tile (top-left coordinates)
} TILE_16;
```

Explanation

Draws a fixed size 1 x 1, 8 x 8 or 16 x 16 flat colored sprite.

Functions

ClearOTagR

Initializes an array to an empty ordering table (reverse order)

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*12/21/2018*

Syntax

void ClearOTagR(
unsigned int **ot*, — Pointer to an array to initialize into a linked list.
int *n***)** — Number of array elements.

Explanation

Initializes an array specified by *ot to a linked list to use as an ordering table. An ordering table consists of an array of pointers that point from one entry to the next which primitives may be registered to.

This function uses DMA to clear the ordering table. It prepares a reverse order list which starts at the last entry of the array and ends at the first. This is ideal for 3D graphics as higher table entries are drawn first and lower entries are drawn last. Primitives registered to the same entry first are always drawn last.

To execute and array initialized by ClearOTag(), execute DrawOTag(ot+n-1) (draw from last entry of array) since the ordering table is in reverse order.

See Also

DrawOTag

Executes an ordering table

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*12/21/2018*

Syntax

void DrawOTag(
unsigned int **ot***)** Pointer to an ordering table to execute.

Explanation

Executes primitives linked into an ordering table array specified by **ot*.

When executing an ordering table initialized by ClearOTagR(), you must specify the last entry in the array.

DrawOTag() uses DMA to query primitives to the GPU and may be non-blocking during DMA page gaps. Use DrawSync() to ensure execution of the ordering table has completed.

See also

DrawSync

Waits until all GPU drawing or transfers have completed

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*12/21/2018*

Syntax

void DrawSync()

Explanation

Waits until the GPU has finished processing drawing commands or VRAM transfers.

Work in progress

This function currently does not take an argument nor return the number of primitives left as it would in the official SDK. There's no timeout when the GPU freezes when it runs into a bad primitive.

GetTimInfo

Get image information of a TIM image file

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*Yes*	*0.1b*	*02/02/2019*

Syntax

int GetTimInfo(	
unsigned int **tim*,	Pointer to a TIM image file.
TIM_IMAGE **timimg***);**	Pointer to a **TIM_IMAGE** structure.

Explanation

Sets the members of the specified **TIM_IMAGE** structure to the relevant image parameters inside a TIM image file specified by **tim*.

Return value

0: success, 1: invalid file ID, 2: unsupported TIM version

GetVideoMode

Gets the current video standard mode

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*2/2/2019*

Syntax

int GetVideoMode()

Explanation

Returns the current video standard mode.

Differences

Unlike the official libraries, this function returns the video mode standard currently set (ie. If this function is called on a PAL machine while in a PAL display mode, it returns 1).

Returns

MODE_NTSC: NTSC, MODE_PAL: PAL

See also

LoadImage

Upload image data to VRAM.

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*12/21/2018*

Syntax

void LoadImage(
RECT **rect*, Pointer to a RECT specifying VRAM destination coordinates.
int *n*) Pointer to source image data.

Explanation

Uploads image data from the source address *data to VRAM. The image size and destination offset in VRAM is specified by *rect using a RECT object.

LoadImage() uses DMA to upload data to VRAM at high speed and may be non-blocking. Use DrawSync() to ensure DMA transfer has completed. Using DrawSync() when uploading multiple images at once is not necessary as LoadImage() will wait for a previous transfer to complete.

If you want to upload a texture image on every frame in a real time sequence it is best to perform the upload after DrawSync() and Vsync().

See also

ResetGraph

Resets the graphics subsystem.

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*12/28/2018*

Syntax

void ResetGraph(
int *mode***)** Reset mode.

Explanation

Resets the GPU and interrupt subsystem according to *mode*.

When this function is first called it calls BIOS functions SetDefaultExitFromException(), ChangeClearPad(), ChangeClearRCnt() and _96_remove() and installs its own event handler for RCntCNT3 used for VSync.

Mode	Operation
0	Resets the GPU completely including the video mode and turns off the display.
1	Cancels any DMA transfer and resets the command buffer.
3	Resets the command buffer.

Work in progress

There's currently no way of removing the event installed by ResetGraph() which may be necessary for loaders. The way the VSync interrupt handler is installed is also not the same as the one in the official library (ResetGraph() in official libraries does not install an event handler).

SetVideoMode

Sets the video standard mode

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*2/2/2019*

Syntax

void SetVideoMode(
int *mode***)** Video standard mode to set.

Explanation

Sets the video signaling standard by *mode* (MODE_NTSC for NTSC or MODE_PAL for PAL).

VSync

Waits for vertical retrace.

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*12/21/2018*

Syntax

void VSync()

Explanation

Waits until a vertical retrace occurs. In an interlaced video mode it additionally waits if the GPU is currently displaying odd or even rows of the frame buffer which alternates on every call of Vsync.

If a callback function is set, the callback function is executed first before this function completes.

Work in progress notes

This function currently does not take an argument for retrieving VSync counter values and waiting until a number of VSyncs have passed as it would on the official SDK.

See also

VSyncCallback()

VSyncCallback

Sets a specified function to be executed on every vertical blanking period.

Library	Header	Original Code	Introduced	Documentation Date
liblibpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*2/2/2019*

Syntax

void VsyncCallback(
void *(*func)()***);** Pointer to a callback function

Explanation

Sets a callback function specified by **func* to be executed on every vertical blanking period. Setting 0 will disable a previously specified callback function.

Because the callback function is executed during a critical section on every vertical blank, it is necessary to finish any processing quickly. Or an interrupt deadlock will occur (exception handler finishes but a vblank has passed so the system enters exception handling again).

It is recommended to define any variable manipulated by a callback function as **volatile** to make sure that any code reading the variable outside the callback handler will always read the variable for changes.

See also

VSync()

Macros

addPrim

Register a primitive to an ordering table

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*2/2/2019*

Syntax

addPrim(
ot, Pointer to an ordering table element.
*pri***);** Pointer to a primitive to register.

Explanation

Registers a primitive specified by *pri* to an ordering table or another primitive specified by *ot*. The pointers of both the specified ordering table element and primitive are manipulated in such a way that the primitive is linked to the ordering table chain.

There's a common misconception where you can only sort one primitive per ordering table element at a time otherwise it would corrupt the ordering table. This is not the case in reality as adding a primitive to an ordering table element that has a primitive already registered to it will get append to the chain. This can be done safely any number of times.

This misconception probably came from a misunderstanding of this statement in the official library documents:

"A primitive may be added to a primitive list only once in the same frame. Attempting to add it multiple times in the same frame results in a corrupted list."

It likely meant that a primitive that had already been added to an ordering table can't be added to the ordering table again, only new primitives that have not been previously registered can be safely added.

setVector

Set elements of a vector

Library	Header File	Original Code	Introduced	Documentation Date
libpsxgpu.a	*psxgpu.h*	*No*	*0.1b*	*2/2/2019*

Syntax

setVector(
**p,* Pointer to a vector.
*x, y, z***);** Coordinate values.

Explanation

Sets the (x, y, z) elements of a VECTOR or SVECTOR structure defined in psxgte.h.