PlayStation Support Library

The PlayStation support library is not a core BRender library. It provides useful support functionality on a platform specific basis.

Initialising the Support Library

This should be performed after BRender library initialisation.

PSGfxBegin()

Description: Initialise the PlayStation support library.

Declaration: **`br_pixelmap * PSGfxBegin(brps_draw_environment* draw_env, brps_display_environment* disp_env, void* primitive, br_uint_32 primitive_size, br_uint_32 flags)`**

Arguments: **`brps_draw_environment * draw_env`**

A pointer to an array of 1 or 2 drawing environments.

`brps_display_environment * disp_env`

A pointer to an array of 1 or 2 display environments.

`void * primitive`

A pointer to an allocated area of memory which is used as a temporary workspace for the support library. Support library functions which require a temporary workspace are documented accordingly.

`br_uint_32 primitive_size`

Size of support library primitive workspace in bytes.

`br_uint_32 flags`

Used to determine single or double buffering. Use `PS_GFX_SINGLE_BUFFER` or `PS_GFX_DOUBLE_BUFFER`.

Result: **`br_pixelmap *`**

A pixelmap which describes the PlayStation frame buffer. Texture pages in the frame buffer may be sub-allocated from this pixelmap.

Remarks: The drawing and display environments are used to redefine the current environments when double buffering is taking place. Attributes of the current environments may be altered during display list traversal, although this does not alter the environment definition which is used when the environments are switched.

Example:

```
/* Setup double buffered environment */
brps_draw_environment draw[2];
brps_display_environment [2];
br_pixelmap *screen;
void *primitive;
```

```
...
BrBegin();
...
/* Define double buffered drawing environments */
BrPSDrawEnvironmentDefine(draw, 0, 0, 320, 240);
BrPSDrawEnvironmentDefine(draw + 1, 320, 0, 320, 240);

/* Define double buffered display environments */
BrPSDisplayEnvironmentDefine(display, 320, 0, 320, 240);
BrPSDisplayEnvironmentDefine(display + 1, 0, 0, 320, 240);

screen = PSGfxBegin(draw, display, primitive, 0,
PS_GFX_DOUBLE_BUFFER);
...
```

Support Library Functions

Once the support library has been initialised, a number of functions are available.

PSGfxDoubleBufferIndexRead()

Description: Get the current double buffer index.

Declaration: **br_uint_8 PSGfxDoubleBufferIndexRead(void)**

Result: **br_uint_8**

The buffer index (0 or 1) used for indexing double buffered data.

Remarks: When using a double buffered system, data such as rendering primitives must be arranged in a two element array. The double buffer index returned by this function is used to index into the data array.

Example:

```
br_uint_8 db_index;
brps_prim_poly_f3 poly_f3[2];
...
db_index = PSGfxDoubleBufferIndexRead();
BrPSPrimPolyF3Set(&poly_f3[db_index]);
...
```

PSGfxDoubleBufferCallbackSet()

Description: Set the call-back function which is invoked by the double buffering operation.

Declaration: **ps_gfx_double_buffer_cbfn * PSGfxDoubleBufferCallbackSet(
ps_gfx_double_buffer_cbfn *new_cbfn)**

Arguments: **`ps_gfx_double_buffer_cbfn * new_cbfn`**

A pointer to a new call-back function. `NULL` will indicate that no call-back is used.

Preconditions: Between **`BrBegin()`** & **`BrEnd()`**. Between **`PSGfxBegin()`** & **`PsGfxEnd()`**.

Result: **`ps_gfx_double_buffer_cbfn *`**

Returns a pointer to the old call-back function.

Remarks: The callback occurs after the environments have been switched, but before the display list traversal begins. This allows users to perform some display operations before primitive rendering takes place.

See Also: **`ps_gfx_double_buffer_cbfn`**.

PSGfxDrawSyncCallbackSet()

Description: Set the call-back function which is invoked upon termination of display list traversal.

Declaration: **`ps_gfx_draw_sync_cbfn * PSGfxDrawSyncCallbackSet(ps_gfx_draw_sync_cbfn *new_cbfn)`**

Arguments: **`ps_gfx_draw_sync_cbfn * new_cbfn`**

A pointer to a new call-back function. `NULL` will indicate that no call-back is used.

Preconditions: Between **`BrBegin()`** & **`BrEnd()`**. Between **`PSGfxBegin()`** & **`PsGfxEnd()`**.

Result: **`ps_gfx_draw_sync_cbfn *`**

Returns a pointer to the old call-back function.

Remarks: The callback occurs when the end of the display list is encountered whilst rendering. Subsequent draw sync call-backs are masked whilst inside the call-back function, so minimal processing should be performed.

See Also: **`ps_gfx_draw_sync_cbfn`**.

PSGfxVSyncCallbackSet()

Description: Set the call-back function which is invoked upon vertical synchronization.

Declaration: **`ps_gfx_vsync_cbfn * PSGfxVSyncCallbackSet( ps_gfx_vsync_cbfn *new_cbfn)`**

Arguments: **`ps_gfx_vsync_cbfn * new_cbfn`**

A pointer to a new call-back function. `NULL` will indicate that no call-back is used.

Preconditions: Between **`BrBegin()`** & **`BrEnd()`**. Between **`PSGfxBegin()`** & **`PsGfxEnd()`**.

Result: **`ps_gfx_vsync_cbfn *`**

Returns a pointer to the old call-back function.

Remarks: The callback occurs at the start of the vertical synchronisation period. Subsequent vsync call-backs are masked whilst inside the call-back function, so minimal processing should be performed.

See Also: **`ps_gfx_vsync_cbfn`**.

Terminating the Support Library

The support library should be terminated correctly before theallocated workspace assigned to the support library is de-allocated.

PSGfxEnd()

Description: Terminate the support library.

Declaration: **`void PSGfxEnd(void)`**

Effects: Terminates the support library and releases any internal resources.

See Also: **`PSGfxBegin()`**.

ps_gfx_double_buffer_cbfn

The Call-Back Function

This type defines a call-back function which can be specified by using the function **`PSGfxDoubleBufferCallbackSet()`**. It is called when double buffered environments are being swapped. It enables an application to perform extra computations after the environments have been switched, but before display list traversal takes place.

The ***`typedef`***

(See *`psio.h`* for precise declaration)

`void ps_gfx_double_buffer_cbfn(br_uint_8 db_index)`***Double Buffer call-back***

Related Functions

For details of how to specify that a double buffer call-back function should be called during double buffer switching, see **`PSGfxDoubleBufferCallbackSet()`**.

Specification

CBFnDoubleBuffer()

Description: An application defined call-back function that is called during double buffer switching. The pass through equivalent for this call-back is to do nothing.

Declaration: **`void BR_CALLBACK CBfnDoubleBuffer(br_uint_8 db_index)`**

Arguments: **`br_uint_8 db_index`**

Current double buffer index.

Preconditions: Double buffer switching is in progress. The application call-back function has been set using **`PSGfxDoubleBufferCallbackSet()`**.

Effects: Behaviour is up to the application.

Remarks: Any other BRender functions may be called from within this call-back with the following restrictions:

- Don’t call the double buffer switching function.

ps_gfx_draw_sync_cbfn

The Call-Back Function

This type defines a call-back function which can be specified by using the function **`PSGfxDrawSyncCallbackSet()`**. It is called when the current drawing queue is empty i.e. upon encountering the end of the display list. It enables an application to perform extra computations after the current queue of rendering primitives has been rendered by the GPU.

The ***`typedef`***

(See *`psio.h`* for precise declaration)
`void ps_gfx_draw_sync_cbfn(br_uint_8 db_index)` ***Drawing synchronization call-back***

Related Functions

For details of how to specify that a drawing synchronization call-back function should be called when drawing is completed, see **`PSGfxDrawSyncCallbackSet()`**.

Specification

CBFnDrawSync()

Description: An application defined call-back function that is called when drawing is completed. The pass through equivalent for this call-back is to do nothing.

Declaration: **`void BR_CALLBACK CBfnDrawSync(br_uint_8 db_index)`**

Arguments: **`br_uint_8 db_index`**

Current double buffer index.

Preconditions: Drawing is complete. The application call-back function has been set using **`PSGfxDrawSyncCallbackSet()`**.

Effects: Behaviour is up to the application. Further drawing synchronization call-backs are inhibited whist inside the call-back function, so any operations should be kept to a minimum.

Remarks: Any other BRender functions may be called from within this call-back.

ps_gfx_vsync_cbfn

The Call-Back Function

This type defines a call-back function which can be specified by using the function **`PSGfxVSyncCallbackSet()`**. It is called when vertical synchronization is taking place. It enables an application to perform extra computations during the vertical retrace period at regular intervals.

The ***`typedef`***

(See *`psio.h`* for precise declaration)
`void ps_gfx_vsync_cbfn(br_uint_8 db_index)` ***Vertical synchronization call-back***

Related Functions

For details of how to specify that a vertical synchronization call-back function should be called at the start of vertical synchronization, see **`PSGfxVSyncCallbackSet()`**.

Specification

CBFnVSync()

Description: An application defined call-back function that is called at the start of vertical synchronization. The pass through equivalent for this call-back is to do nothing.

Declaration: **`void BR_CALLBACK CBfnVSync(br_uint_8 db_index)`**

Arguments: **`br_uint_8 db_index`**

Current double buffer index.

Preconditions: The application call-back function has been set using **`PSGfxVSyncCallbackSet()`**.

Effects: Behaviour is up to the application. Further vertical synchronization call-backs are inhibited whist inside the call-back function, so any operations should be kept to a minimum.

Remarks: Any other BRender functions may be called from within this call-back.

Scene Rendering

The PlayStation has dedicated hardware which is designed for primitive sorting hidden surface schemes. It does not perform z buffering and the implementation of a z buffer scheme is not practical for reasons of performance. Consult the Technical Reference Manual for details of the z sort scheme used by BRender.

Additional Functionality

The following functions are an extension of the functionality offered by the BRender z sort scheme.

BrZsDefaultOrderTableGet()

Description: Return a pointer to the BRender z sort default order table.

Declaration: **`br_order_table * BrZsDefaultOrderTableGet(void)`**

Result: **`br_order_table *`**

A pointer to the default order table. This order table is used when no user order table is specified.

BrZsDefaultOrderTableClear()

Description: Clear the default BRender z sort order table.

Declaration: **`void BrZsDefaultOrderTableClear(void)`**

Effects: Clear the default order table.

BrZsCurrentOrderTableGet()

Description: Return a pointer to the current order table used for primitive insertion.

Declaration: **`br_order_table *BrZsDefaultOrderTableClear(void)`**

Result: **`br_order_table *`**

A pointer to the current order table used for primitive insertion.

Preconditions: Model or Scene rendering is being performed. A **`br_renderbounds_cbfn`**, **`br_primitive_cbfn`** or **`br_model_custom_cbfn`** call-back has been invoked.

Remarks: This function is useful in call-back functions to obtain the current order table for user insertion and manipulation. The order table being used is defined by the current actor or by inheritance across a hierarchy.

BrZsSceneFogSet()

Description: Set an arbitrary environment fog colour and distance.

Declaration: **`br_vector2 *BrZsSceneFogSet(br_uint_32 colour, br_vector2 *bounds)`**

Preconditions: Model or Scene rendering is not being performed. This function may not be called from a call-back.

Arguments: **`br_uint_32 colour`**

Arbitrary colour for environment fog. Local fog colours may also be used. See **`br_material`**.

`br_vector2 * bounds`

First and second ordinates. The first ordinate is the near distance from the camera along the negative z axis. This specifies the distance where the fog is 0%. The second ordinate is the far distance from the camera. This specifies the distance where the fog is 100%. These values should be greater than zero.

Result: **`br_vector2 *`**

The bounds are returned for convenience.

br_vector2s

The Structure

This is the two ordinate vector structure, typically used for 2D calculations on the PlayStation. Macros are provided to allow it be used as though it were an integral type.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

br_int_16	**v[2]**	***Ordinates (0=x, 1=y)***

Members

br_int_16 v[2]

First and second ordinate. Conventionally, the first ordinate is the x-axis component and the second, the y axis component.

Arithmetic

Use the **br_vector2** arithmetic macros supplied.

See **br_vector2**.

Copy/Assign

Use the **br_vector2** assignment and copying macros supplied.

See **br_vector2**.

Referencing & Lifetime

This structure may be freely referenced, though take care if there is potential to supply the same vector as more than one argument to the same function.

Initialisation

Use the **br_vector2** macros supplied for initialisation.

See **br_vector2**.

br_vector3s

The Structure

This is the three ordinate vector structure, typically used for 3D calculations on the PlayStation. Macros are provided to allow it be used as though it were an integral type.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

br_int_16	**v[3]**	***Ordinates (0=x, 1=y, 2=z)***

Members

br_int_16 v[3]

First, second and third ordinate. Conventionally, the first ordinate is the x-axis component, the second, the y axis component, and the third, the z axis component. Remember that BRender has a right handed co-ordinate system and so, with the x axis positive to the right, and the y axis positive upwards, the z axis is therefore positive toward you (typically, the z axis points out of the screen).

Arithmetic

Use the **br_vector3** arithmetic macros supplied.

See **br_vector3**.

Copy/Assign

Use the **br_vector3** assignment and copying macros supplied.

See **br_vector3**.

Referencing & Lifetime

This structure may be freely referenced, though take care if there is potential to supply the same vector as more than one argument to the same function.

Initialisation

Use the **br_vector3** macros supplied for initialisation.

See **br_vector3**.

br_vector2b

The Structure

This is the two ordinate vector structure, typically used for texture map coordinate declarations on the PlayStation. Macros are provided to allow it be used as though it were an integral type.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

br_uint_8	**v[2]**	***Ordinates (0=u, 1=v)***

Members

br_uint_8 v[2]

First and second ordinate. Conventionally, the first ordinate is the u component, and the second, the v component. The ordinates specify an offset within one specific texture page in the frame buffer.

Arithmetic

Use the **br_vector2** arithmetic macros supplied.

See **br_vector2**.

Copy/Assign

Use the **br_vector2** assignment and copying macros supplied.

See **br_vector2**.

Referencing & Lifetime

This structure may be freely referenced, though take care if there is potential to supply the same vector as more than one argument to the same function.

Initialisation

Use the **br_vector2** macros supplied for initialisation.

See **br_vector2**.

br_vector3b

The Structure

This is the three ordinate vector structure, typically used for RGB colour declarations on the PlayStation. Macros are provided to allow it be used as though it were an integral type.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

br_uint_8	**v[3]**	***Ordinates (0=r, 1=g, 2=b)***

Members

br_uint_8 v[3]

First, second and third ordinate. Conventionally, the first ordinate is the red (r) component, the second, the green (g) component, and the third the blue (b) component. The ordinates specify an RGB triple which define a point in the RGB colour space.

Arithmetic

Use the **br_vector3** arithmetic macros supplied.

See **br_vector3**.

Copy/Assign

Use the **br_vector3** assignment and copying macros supplied.

See **br_vector3**.

Referencing & Lifetime

This structure may be freely referenced, though take care if there is potential to supply the same vector as more than one argument to the same function.

Initialisation

Use the **br_vector3** macros supplied for initialisation.

See **br_vector3**.

br_vector4b

The Structure

This is the four ordinate vector structure, typically used for RGB colour declarations for rendering primitives on the PlayStation. Macros are provided to allow it be used as though it were an integral type.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

br_uint_8	**v[4]**	***Ordinates (0=r, 1=g, 2=b, 3=code)***

Members

br_uint_8 v[4]

First, second, third and fourth ordinate. Conventionally, the first ordinate is the red (r) component, the second, the green (g) component, the third the blue (b) component, and the fourth, the rendering primitive code component. The ordinates specify an RGB triple which define a point in the RGB colour space and the type of primitive to be rendered by the GPU.

Arithmetic

Use the **br_vector4** arithmetic macros supplied.

See **br_vector4**.

Copy/Assign

Use the **br_vector4** assignment and copying macros supplied.

See **br_vector4**.

Referencing & Lifetime

This structure may be freely referenced, though take care if there is potential to supply the same vector as more than one argument to the same function.

Initialisation

Use the **br_vector4** macros supplied for initialisation.

See **br_vector4**.

brps_prim_tag

The Structure

This structure is a header description for a GPU rendering primitive on the PlayStation.

The **typedef**

(See *ps.h* for precise declaration and ordering)

```
br_uint_32              addr:24
br_uint_32              len:8
```

Members

br_uint_32 addr:24

This is the low 24 bits of a 32 bit pointer to a rendering primitive in a display list. An order table is a list of primitives linked by **brps_prim_tag** members.

br_uint_32 len:8

This is the length of a rendering primitive which contains the **brps_prim_tag** structure in 32 bit units. Primitives may be merged to form larger primitive packets which enhances the performance of the display list traversal. The maximum size of a primitive packet in a display list is 16 units.

Copy/Assign

Use copy by structure assignment freely, although pointer references must be resolved to avoid the creation of circular display lists as this will result in a GPU timeout.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Macros for Standard Operations

`BrPSPrimAddrSet(p,address)`

Assign address to the addr member of the rendering primitive or **brps_prim_tag** structure.

`BrPSPrimLenSet(p,length)`

Assign length to the len member of the rendering primitive or **brps_prim_tag** structure. The length is expressed in 32 bit units.

Referencing & Lifetime

This structure may be freely referenced. A rendering primitive header with length zero may be used as a blank entry in an order table. A simple description of an order table is an array of **`brps_prim_tag`** with length zero.

Macros for Standard Operations

`BrPSPrimAddrGet(p)`

Return the low 24 bits of the address of the next primitive pointed by the **`brps_prim_tag`** header in a display list.

`BrPSPrimLenGet()`

Return the length in 32 bit units of the rendering primitive containing the **`brps_prim_tag`** structure. This includes the size of the **`brps_prim_tag`** structure.

`BrPSPrimNext(p)`

Return the next primitive in a display list.

`BrPSPrimIsEnd(p)`

Return a binary condition flag indicating if the primitive is at the end of a display list.

`BrPSPrimAdd(p, primitive)`

Assign the address of primitive to the addr member of p.

`BrPSPrimTerminate(p)`

Mark this primitive as the end of a display list.

`BrPSPrimCat(p, primitive)`

Join the display list headed by primitive to the display list with the tail p.

`BrPSPrimMerge(p, primitive)`

Merge primitive with p to form a single primitive packet. The maximum length of a rendering primitive is 16 units. A length greater than this will cause a GPU error.

Macros for General Primitive Operations

`BrPSPrimTPageSet(p, type, abr, x, y)`

Set the tpage member of the primitive pointed to by p.

`type`

0: 4 bit indexed

1: 8 bit indexed

2: 16 bit direct colour

`abr` ***(semi-translucency rate)***

0:	0.5 back x 0.5 front
1:	1.0 back x 1.0 front
2:	0.5 back x 1.0 front
3:	-1.0 back x 1.0 front

`x,y`

Offset of texture page within frame buffer. X is limited to 64 pixel boundaries, y is limited to 256 pixel boundries.

`BrPSPrimClutSet(p, x, y)`

Set the clut member of the primitive pointed to by p.

`x,y`

Offset of clut within frame buffer. X is limited to 64 pixel boundaries.

`BrPSPrimSemiTransSet(p, abe)`

Set the semi-translucency flag of the primitive pointed to by p.

`abe` ***(semi-translucency flag)***

0: Opaque, 1: Semi-translucent.

This macro must be used after the rendering primitive has been initialised.

Example:

```
brps_prim_poly_f3 poly_f3;

...

BrPSPrimPolyF3Set(&poly_f3);

BrPSPrimSemiTransSet(&poly_f3, 1);
```

`BrPSPrimShadeTexSet(p, tge)`

Set the shade flag of the primitive pointed to by p. If shading is disabled, the brightness values of the rendering primitive are ignored and only the texture colour value is used.

`tge`

0: Shaded, 1: Not shaded

This macro must be used after the rendering primitive has been initialised.

Example:

```
brps_prim_poly_f3 poly_gt3;
...
BrPSPrimPolyF3Set(&poly_gt3);
BrPSPrimShadeTexSet(&poly_gt3, 1);
```

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macros for Standard Operations

`BrPSPrimAddrSet(p, address)`

Set the addr member of a rendering primitive or **`brps_prim_tag`** structure.

`BrPSPrimLenSet(p, length)`

Set the len member of a rendering primitive or **`brps_prim_tag`** structure.

brps_rectangle

The Structure

This structure is a description of a rectangular region of the frame buffer on the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

```
br_uint_16              x
br_uint_16              y
br_uint_16              w
br_uint_16              h
```

Related Macros

See BrPSDrawEnvironmentDefine(), BrPSDisplayEnvironmentDefine().

Related Structures

See **brps_draw_environment**, **brps_display_environment**.

Members

br_int_16 x,y

These members indicate an offset which is the top left of a rectangular region within the frame buffer in pixels. Neither negative values or those exceeding the size of the frame buffer (1024x512) may be specified.

br_int_16 w,h

These members indicate the width and height of the rectangular region in pixels. Neither negative values or those exceeding the size of the frame buffer (1024x512) may be specified.

Copy/Assign

Use copy by structure assignment freely.

Referencing & Lifetime

This structure may be freely referenced.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`brps_rectangle`

`BrPSRectangleSet(r,x,y,w,h)`

Assign members of **`brps_rectangle`** structure pointed to by r.

brps_prim_poly_f3

The Structure

This structure describes a flat shaded triangle rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

```
brps_prim_tag          tag
br_vector4b            colour0                Ordinates (0=r, 1=g, 2=b, 3=code)
br_vector2s            v0
br_vector2s            v1
br_vector2s            v2
```

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_poly_f4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

brps_draw_environmentBrightness of colour components of flat shaded polygon and primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set().

br_vector2s v0,v1,v2

Vertex coordinates of triangle.

See `BrPSPrimVertex0Set()`, `BrPSPrimVertex1Set()`, `BrPSPrimVertex2Set()`.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimPolyF3Set(p)`

Set code identifier and header length members.

brps_prim_poly_f4

The Structure

This structure describes a flat shaded quadrilateral rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

```
brps_prim_tag          tag
br_vector4b            colour0              Ordinates (0=r, 1=g, 2=b, 3=code)
br_vector2s            v0
br_vector2s            v1
br_vector2s            v2
br_vector2s            v3
```

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_poly_f3**, **brps_prim_tile**, **brps_prim_tile_1**, **brps_prim_tile_8**, **brps_prim_tile_16**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of flat shaded polygon and primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set().

br_vector2s v0,v1,v2,v3

`brps_prim_poly_f4`

Vertex coordinates of quadrilateral.

See `BrPSPrimVertex0Set()`, `BrPSPrimVertex1Set()`, `BrPSPrimVertex2Set()`, `BrPSPrimVertex3Set()`.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimPolyF4Set(p)`

Set code identifier and header length members.

brps_prim_poly_g3

The Structure

This structure describes a gouraud shaded triangle rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	
br_vector4b	**colour1**	***Ordinates (0=r, 1=g, 2=b)***
br_vector2s	**v1**	
br_vector4b	**colour2**	***Ordinates (0=r, 1=g, 2=b)***
br_vector2s	**v2**	

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_poly_g4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0,colour1,colour2

Brightness of colour components of vertices for gouraud shading. The fourth ordinate of **colour0** contains the primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set(), BrPSPrimColour1Set(), BrPSPrimColour2Set().

br_vector2s v0,v1,v2

Vertex coordinates of triangle.

See BrPSPrimVertex0Set(), BrPSPrimVertex1Set(), BrPSPrimVertex2Set().

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

BrPSPrimPolyG3Set(p)

Set code identifier and header length members.

brps_prim_poly_g4

The Structure

This structure describes a gouraud shaded quadrilateral rendering primitive for the PlayStation.

The ***typedef***

(See `ps.h` for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	
br_vector4b	**colour1**	***Ordinates (0=r, 1=g, 2=b)***
br_vector2s	**v1**	
br_vector4b	**colour2**	***Ordinates (0=r, 1=g, 2=b)***
br_vector2s	**v2**	
br_vector4b	**colour3**	***Ordinates (0=r, 1=g, 2=b)***
br_vector2s	**v3**	

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_poly_g3**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0,colour1,colour2,colour3

Brightness of colour components of vertices for gouraud shading. The fourth ordinate of **colour0** contains the primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set(), BrPSPrimColour1Set(), BrPSPrimColour2Set(), BrPSPrimColour3Set().

br_vector2s v0,v1,v2,v3

Vertex coordinates of quadrilateral.

See BrPSPrimVertex0Set(), BrPSPrimVertex1Set(), BrPSPrimVertex2Set(), BrPSPrimVertex3Set().

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

BrPSPrimPolyG4Set(p)

Set code identifier and header length members.

brps_prim_poly_ft3

The Structure

This structure describes a flat shaded texture mapped triangle rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	
br_vector2b	**map0**	***Ordinates (0=u, 1=v)***
br_uint_16	**clut**	
br_vector2s	**v1**	
br_vector2b	**map1**	***Ordinates (0=u, 1=v)***
br_uint_16	**tpage**	
br_vector2s	**v2**	
br_vector2b	**map2**	***Ordinates (0=u, 1=v)***

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_poly_ft4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of flat shaded triangle. The fourth ordinate of **`colour0`** contains the primitive code identifier.

See **`br_vector4b`**, `BrPSPrimColour0Set()`.

`br_vector2s v0,v1,v2`

Vertex coordinates of triangle.

See `BrPSPrimVertex0Set()`, `BrPSPrimVertex1Set()`, `BrPSPrimVertex2Set()`.

`br_vector2b map0,map1,map2`

Texture map coordinates of triangle within texture page.

See **`br_vector2b`**, `BrPSPrimMap3Set()`.

`br_uint_16 clut`

Clut identifier. This identifier determines the offset of the clut within the frame buffer for indexed texture pages.

See `BrPSPrimClutSet()`.

`br_uint_16 tpage`

Texture page identifier. This identifier determines the pixel depth of the texture page, the offset of the texture page within the frame buffer and the translucency rate.

See `BrPSPrimTPageSet()`.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimPolyFT3Set(p)`

Set code identifier and header length members.

brps_prim_poly_ft4

The Structure

This structure describes a flat shaded texture mapped quadrilateral rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

```
brps_prim_tag        tag
br_vector4b          colour0          Ordinates (0=r, 1=g, 2=b, 3=code)
br_vector2s          v0
br_vector2b          map0             Ordinates (0=u, 1=v)

br_uint_16           clut
br_vector2s          v1
br_vector2b          map1             Ordinates (0=u, 1=v)

br_uint_16           tpage
br_vector2s          v2
br_vector2b          map2             Ordinates (0=u, 1=v)

br_vevtor2s          v3
br_vector2b          map3             Ordinates (0=u, 1=v)
```

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_poly_ft3**, **brps_prim_sprite**, **brps_prim_sprite_8**, **brps_prim_sprite_16**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of flat shaded triangle. The fourth ordinate of **colour0** contains the primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set().

br_vector2s v0,v1,v2,v3

Vertex coordinates of quadrilateral.

See BrPSPrimVertex0Set(), BrPSPrimVertex1Set(), BrPSPrimVertex2Set(), BrPSPrimVertex3Set().

br_vector2b map0,map1,map2,map3

Texture map coordinates of quadrilateral within texture page.

See **br_vector2b**, BrPSPrimMap4Set().

br_uint_16 clut

Clut identifier. This identifier determines the offset of the clut within the frame buffer for indexed texture pages.

See BrPSPrimClutSet().

br_uint_16 tpage

Texture page identifier. This identifier determines the pixel depth of the texture page, the offset of the texture page within the frame buffer and the translucency rate.

See BrPSPrimTPageSet().

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

BrPSPrimPolyFT4Set(p)

Set code identifier and header length members.

brps_prim_poly_gt3

The Structure

This structure describes a gouraud shaded texture mapped triangle rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	
br_vector2b	**map0**	***Ordinates (0=u, 1=v)***
br_uint_16	**clut**	
br_vector4b	**colour1**	***Ordinates (0=r, 1=g, 2=b)***
br_vector2s	**v1**	
br_vector2b	**map1**	***Ordinates (0=u, 1=v)***
br_uint_16	**tpage**	
br_vector4b	**colour2**	***Ordinates (0=r, 1=g, 2=b)***
br_vector2s	**v2**	
br_vector2b	**map2**	***Ordinates (0=u, 1=v)***

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_poly_gt4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0,colour1,colour2

Brightness of colour components of vertices for gouraud shading. The fourth ordinate of **`colour0`** contains the primitive code identifier.

See **`br_vector4b`**, `BrPSPrimColour0Set()`, `BrPSPrimColour1Set()`, `BrPSPrimColour2Set()`.

`br_vector2s v0,v1,v2`

Vertex coordinates of triangle.

See `BrPSPrimVertex0Set()`, `BrPSPrimVertex1Set()`, `BrPSPrimVertex2Set()`.

`br_vector2b map0,map1,map2`

Texture map coordinates of triangle within texture page.

See **`br_vector2b`**, `BrPSPrimMap3Set()`.

`br_uint_16 clut`

Clut identifier. This identifier determines the offset of the clut within the frame buffer for indexed texture pages.

See `BrPSPrimClutSet()`.

`br_uint_16 tpage`

Texture page identifier. This identifier determines the pixel depth of the texture page, the offset of the texture page within the frame buffer and the translucency rate.

See `BrPSPrimTPageSet()`.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimPolyGT3Set(p)`

Set code identifier and header length members.

brps_prim_poly_gt4

The Structure

This structure describes a gouraud shaded texture mapped quadrilateral rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	
br_vector2b	**map0**	***Ordinates (0=u, 1=v)***
br_uint_16	**clut**	
br_vector4b	**colour1**	***Ordinates (0=r, 1=g, 2=b)***
br_vector2s	**v1**	
br_vector2b	**map1**	***Ordinates (0=u, 1=v)***
br_uint_16	**tpage**	
br_vector4b	**colour2**	***Ordinates (0=r, 1=g, 2=b)***
br_vector2s	**v2**	
br_vector2b	**map2**	***Ordinates (0=u, 1=v)***
br_vector4b	**colour3**	***Ordinates (0=r, 1=g, 2=b)***
br_vector2s	**v2**	
br_vector2b	**map2**	***Ordinates (0=u, 1=v)***

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_poly_gt3**.

Members

brps_prim_tag tag

brps_prim_poly_gt4

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0,colour1,colour2

Brightness of colour components of vertices for gouraud shading. The fourth ordinate of **colour0** contains the primitive code identifier.

See **br_vector4b**, **BrPSPrimColour0Set()**, BrPSPrimColour1Set(), BrPSPrimColour2Set(), BrPSPrimColour3Set().

br_vector2s v0,v1,v2

Vertex coordinates of quadrilateral.

See BrPSPrimVertex0Set(), BrPSPrimVertex1Set(), BrPSPrimVertex2Set(), **BrPSPrimVertex3Set()**.

br_vector2b map0,map1,map2

Texture map coordinates of quadrilateral within texture page.

See **br_vector2b**, BrPSPrimMap4Set().

br_uint_16 clut

Clut identifier. This identifier determines the offset of the clut within the frame buffer for indexed texture pages.

See BrPSPrimClutSet().

br_uint_16 tpage

Texture page identifier. This identifier determines the pixel depth of the texture page, the offset of the texture page within the frame buffer and the translucency rate.

See BrPSPrimTPageSet().

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimPolyGT4Set(p)`

Set code identifier and header length members.

brps_prim_line_f2

The Structure

This structure describes a flat shaded non-connecting line rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	
br_vector2s	**v1**	

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_line_f3**. **brps_prim_line_f4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of flat shaded non-connected line and primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set().

br_vector2s v0,v1

Vertex coordinates of line. A line is drawn connecting (v0.v[0], v0.v[1]) to (v1.v[0], v1.v[1]).

See BrPSPrimVertex0Set(), BrPSPrimVertex1Set().

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimLineF2Set(p)`

Set code identifier and header length members.

brps_prim_line_f3

The Structure

This structure describes a flat shaded two segment connecting line rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

```
brps_prim_tag          tag
br_vector4b            colour0               Ordinates (0=r, 1=g, 2=b, 3=code)
br_vector2s            v0
br_vector2s            v1
br_vector2s            v2
```

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_line_f2**. **brps_prim_line_f4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of flat shaded two segment connected line and primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set().

br_vector2s v0,v1,v2

Vertex coordinates of line. A line is drawn connecting (v0.v[0], v0.v[1]) to (v1.v[0], v1.v[1]) to (v2.v[0], v2.v[1]).

See `BrPSPrimVertex0Set()`, `BrPSPrimVertex1Set()`, `BrPSPrimVertex2Set()`.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimLineF3Set(p)`

Set code identifier and header length members.

brps_prim_line_f4

The Structure

This structure describes a flat shaded three segment connecting line rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

```
brps_prim_tag           tag
br_vector4b             colour0                 Ordinates (0=r, 1=g, 2=b, 3=code)
br_vector2s             v0
br_vector2s             v1
br_vector2s             v2
br_vector2s             v3
```

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_line_f2**. **brps_prim_line_f3**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of flat shaded three segment connected line and primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set().

br_vector2s v0,v1,v2,v3

Vertex coordinates of line. A line is drawn connecting (v0.v[0], v0.v[1]) to (v1.v[0], v1.v[1]) to (v2.v[0], v2.v[1]) to (v3.v[0], v3.v[1]).

See `BrPSPrimVertex0Set()`, `BrPSPrimVertex1Set()`, `BrPSPrimVertex2Set()`, `BrPSPrimVertex3Set()`.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimLineF4Set(p)`

Set code identifier and header length members.

brps_prim_line_g2

The Structure

This structure describes a gouraud shaded non-connecting line rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	
br_vector4b	**colour1**	***Ordinates (0=r, 1=g, 2=b)***
br_vector2s	**v1**	

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_line_g3**. **brps_prim_line_g4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0,colour1

Brightness of colour components of vertices for gouraud shaded non-connected line and primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set(), BrPSPrimColour1Set().

br_vector2s v0,v1

Vertex coordinates of line. A line is drawn connecting (v0.v[0], v0.v[1]) to (v1.v[0], v1.v[1]).

See **`BrPSPrimVertex0Set()`**, `BrPSPrimVertex1Set()`.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimLineG2Set(p)`

Set code identifier and header length members.

brps_prim_line_g3

The Structure

This structure describes a gouraud shaded two segment connecting line rendering primitive for the PlayStation.

The ***typedef***

(See `ps.h` for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	
br_vector4b	**colour1**	***Ordinates (0=r, 1=g, 2=b)***
br_vector2s	**v1**	
br_vector4b	**colour2**	***Ordinates (0=r, 1=g, 2=b)***
br_vector2s	**v2**	

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See `BrPSPrimAdd()`, `BrPSPrimCat()`, `BrPSPrimNext()`, `BrPSPrimTerminate()`.

Related Structures

See **brps_prim_tag**, **brps_prim_line_g2**. **brps_prim_line_g4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0,colour1,colour2

Brightness of colour components of vertices of gouraud shaded two segment connected line and primitive code identifier.

See **br_vector4b**, `BrPSPrimColour0Set()`, `BrPSPrimColour1Set()`, `BrPSPrimColour2Set()`.

br_vector2s v0,v1,v2

Vertex coordinates of line. A line is drawn connecting (v0.v[0], v0.v[1]) to (v1.v[0], v1.v[1]) to (v2.v[0], v2.v[1]).

See `BrPSPrimVertex0Set()`, `BrPSPrimVertex1Set()`, `BrPSPrimVertex2Set()`.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimLineG3Set(p)`

Set code identifier and header length members.

brps_prim_line_g4

The Structure

This structure describes a flat shaded three segment connecting line rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

```
brps_prim_tag           tag
br_vector4b             colour0                 Ordinates (0=r, 1=g, 2=b, 3=code)
br_vector2s             v0
br_vector2s             v1
br_vector2s             v2
br_vector2s             v3
```

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_line_g2**. **brps_prim_line_g3**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of flat shaded three segment connected line and primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set().

br_vector2s v0,v1,v2,v3

Vertex coordinates of line. A line is drawn connecting (v0.v[0], v0.v[1]) to (v1.v[0], v1.v[1]) to (v2.v[0], v2.v[1]) to (v3.v[0], v3.v[1]).

See `BrPSPrimVertex0Set()`, `BrPSPrimVertex1Set()`, `BrPSPrimVertex2Set()`, `BrPSPrimVertex3Set()`.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimLineF4Set(p)`

Set code identifier and header length members.

brps_prim_sprite

The Structure

This structure describes an arbitrary width sprite rendering primitive for the PlayStation.

The ***typedef***

(See `ps.h` for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	
br_vector2b	**map0**	***Ordinates (0=u, 1=v)***
br_uint_16	**clut**	***Only used for 4/8 bit index texture pages***
br_int_16	**w**	
br_int_16	**h**	

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See `BrPSPrimAdd()`, `BrPSPrimCat()`, `BrPSPrimNext()`, `BrPSPrimTerminate()`.

Related Structures

See **brps_prim_tag**, **brps_prim_sprite_8**, **brps_prim_sprite_16**, **brps_prim_poly_ft4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of sprite and primitive code identifier.

See **br_vector4b**, `BrPSPrimColour0Set()`.

br_vector2s v0

Position of sprite.

See `BrPSPrimVertex0Set()`.

br_vector2b map0

Position of sprite texture within the current texture page. The first ordinate (u) must be an even value. The **`brps_prim_sprite`** structure has no texture page member so the current texture page (the texture page last specified by a rendering primitive) is used. To select a texture page, merge the **`brps_prim_sprite`** structure with a **`brps_prim_draw_mode`** structure. Alternatively use a **`brps_prim_poly_ft4`** primitive in place of the sprite, however a sprite is rendered faster than a polygon.

See `BrPSPrimMap0Set()`.

See also **`brps_prim_draw_mode`**, `BrPSPrimMerge()`.

br_uint_16 clut

Position of the clut to use with the current texture page within the frame buffer. This clut is only used when the current texture page is specified as 4/8 bit indexed.

See `BrPSPrimClutSet()`, `BrPSPrimTPageSet()`.

br_int_16 w,h

These specify the width and height of the sprite primitive. The width must be an even number.

See `BrPSPrimWHSet()`, `BrPSPrimMapWHSet()`.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimSpriteSet(p)`

Set code identifier and header length members.

brps_prim_sprite_8

The Structure

This structure describes an 8x8 pixel sprite rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	
br_vector2b	**map0**	***Ordinates (0=u, 1=v)***
br_uint_16	**clut**	***Only used for 4/8 bit index texture pages***

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_sprite**, **brps_prim_sprite_16**, **brps_prim_poly_ft4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of sprite and primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set().

br_vector2s v0

Position of sprite.

See BrPSPrimVertex0Set().

br_vector2b map0

Position of sprite texture within the current texture page. The first ordinate (u) must be an even value. The **brps_prim_sprite** structure has no texture page member so the current texture page (the texture page last specified by a rendering primitive) is used. To select a texture page, merge the **brps_prim_sprite** structure with a **brps_prim_draw_mode** structure. Alternatively use a **brps_prim_poly_ft4** primitive in place of the sprite, however a sprite is rendered faster than a polygon.

See BrPSPrimMap0Set().

See also **brps_prim_draw_mode**, BrPSPrimMerge().

br_uint_16 clut

Position of the clut to use with the current texture page within the frame buffer. This clut is only used when the current texture page is specified as 4/8 bit indexed.

See BrPSPrimClutSet(), BrPSPrimTPageSet().

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

BrPSPrimSprite8Set(p)

Set code identifier and header length members.

brps_prim_sprite_16

The Structure

This structure describes an 16x16 pixel sprite rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	
br_vector2b	**map0**	***Ordinates (0=u, 1=v)***
br_uint_16	**clut**	***Only used for 4/8 bit index texture pages***

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See `BrPSPrimAdd()`, `BrPSPrimCat()`, `BrPSPrimNext()`, `BrPSPrimTerminate()`.

Related Structures

See **brps_prim_tag**, **brps_prim_sprite**, **brps_prim_sprite_8**, **brps_prim_poly_ft4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of sprite and primitive code identifier.

See **br_vector4b**, `BrPSPrimColour0Set()`.

br_vector2s v0

Position of sprite.

See `BrPSPrimVertex0Set()`.

br_vector2b map0

Position of sprite texture within the current texture page. The first ordinate (u) must be an even value. The **brps_prim_sprite** structure has no texture page member so the current texture page (the texture page last specified by a rendering primitive) is used. To select a texture page, merge the **brps_prim_sprite** structure with a **brps_prim_draw_mode** structure. Alternatively use a **brps_prim_poly_ft4** primitive in place of the sprite, however a sprite is rendered faster than a polygon.

See BrPSPrimMap0Set().

See also **brps_prim_draw_mode**, BrPSPrimMerge().

br_uint_16 clut

Position of the clut to use with the current texture page within the frame buffer. This clut is only used when the current texture page is specified as 4/8 bit indexed.

See BrPSPrimClutSet(), BrPSPrimTPageSet().

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

```
BrPSPrimSprite16Set(p)
```

Set code identifier and header length members.

brps_prim_tile

The Structure

This structure describes an arbitrary width flat colour tile rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	
br_int_16	**w**	
br_int_16	**h**	

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_tile_1**, **brps_prim_tile_8**, **brps_prim_tile_16**, **brps_prim_poly_f4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of tile and primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set().

br_vector2s v0

Position of sprite.

See `BrPSPrimVertex0Set()`.

`br_int_16 w,h`

These specify the width and height of the sprite primitive. The width must be an even number.

See `BrPSPrimWHSet()`.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimTileSet(p)`

Set code identifier and header length members.

brps_prim_tile_1

The Structure

This structure describes a 1x1 pixel flat colour tile rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_tile**, **brps_prim_tile_8**, **brps_prim_tile_16**, **brps_prim_poly_f4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of tile and primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set().

br_vector2s v0

Position of sprite.

See BrPSPrimVertex0Set().

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimTile1Set(p)`

Set code identifier and header length members.

brps_prim_tile_8

The Structure

This structure describes a 8x8 pixel flat colour tile rendering primitive for the PlayStation.

The ***typedef***

(See `ps.h` for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See `BrPSPrimAdd()`, `BrPSPrimCat()`, `BrPSPrimNext()`, `BrPSPrimTerminate()`.

Related Structures

See **brps_prim_tag**, **brps_prim_tile**, **brps_prim_tile_1**, **brps_prim_tile_16**, **brps_prim_poly_f4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of tile and primitive code identifier.

See **br_vector4b**, `BrPSPrimColour0Set()`.

br_vector2s v0

Position of sprite.

See `BrPSPrimVertex0Set()`.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimTile8Set(p)`

Set code identifier and header length members.

brps_prim_tile_16

The Structure

This structure describes a 16x16 pixel flat colour tile rendering primitive for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_vector4b	**colour0**	***Ordinates (0=r, 1=g, 2=b, 3=code)***
br_vector2s	**v0**	

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_tile**, **brps_prim_tile_1**, **brps_prim_tile_8**, **brps_prim_poly_f4**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_vector4b colour0

Brightness of colour components of tile and primitive code identifier.

See **br_vector4b**, BrPSPrimColour0Set().

br_vector2s v0

Position of sprite.

See BrPSPrimVertex0Set().

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation or by member initialisation.

Macro for Standard Initialisation

`BrPSPrimTile16Set(p)`

Set code identifier and header length members.

brps_prim_draw_mode

The Structure

This structure describes a rendering control primitive for the PlayStation. It is used to alter attributes of the current drawing environment (state of GPU).

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_uint_32	**code[2]**	***Reserved***

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_texture_window**, **brps_draw_environment**, **brps_prim_environment**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_uint_32 code[2]

Primitive data in bit fields.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation.

Function for Standard Initialisation

BrPSPrimDrawModeSet()

Description: Define a GPU draw mode rendering control primitive

Declaration: **`void BrPSPrimDrawModeSet(brps_prim_draw_mode* dm, br_int_32 display, br_int_32 dither, br_int_32 tpage, brps_rectangle* rect)`**

Arguments: **`brps_prim_draw_mode * dm`**

A pointer to the destination draw mode primitive.

`br_int_32 display`

Flag to control drawing to display area. The current drawing area need not always be the current display area to enable double buffering. This flag enables GPU drawing to the current display area. 0: Off 1: On.

`br_int_32 dither`

Flag to control dithering by GPU. This flag enables dithering of GPU rendering primitives.

0: Off, 1: On.

`br_int_32 tpage`

Set new current texture page. This texture page will be used until a rendering primitive with a tpage member is encountered. Thus sprites following a **`brps_prim_draw_mode`** primitive will use the new texture page, but the texture page will be reset if a **`brps_prim_poly_ft3`** or similar is used.

`brps_rectangle * rect`

Set new texture window. This window demarks a rectangular region within the current texture page. All subsequent texture co-ordinates will be offset from this region rather than from the origin of the texture page. Texture co-ordinates will be wrapped at the limites of the texture window rather than at the limits of the texture window.

Remarks: It is useful to merge this primitive with sprites to allow mulitple texture pages to be used.

Example:

```
struct {
     brps_prim_draw_mode dm;
     brps_prim_sprite_8 sprite8;
} dm_sprite8;
brps_rectangle texture_window;
```

```
br_int_32 tpage;

...
BrPSPrimSprite8Set(&dm_sprite8.sprite8);
BrPSPrimMap0Set(&dm_sprite8.sprite8, 0, 0);

/* Set display ON, dither OFF, texture page, texture window */
BrPSRectangleSet(&texture_window, 32, 32, 8, 8);
BrPSPrimDrawModeSet(&dm_sprite8.dm, 1, 0, tpage,
   &texture_window);

/* Merge primitives to form single primitive packet
   Sprite texture begins at (32,32) within texture page
   although
   texture coordinates are (0,0) due to texture window offset
   */
BrPSPrimMerge(&dm_sprite8.dm, &dm_sprite8.sprite8);
...
```

brps_prim_texture_window

The Structure

This structure describes a rendering control primitive for the PlayStation. It is used to alter the current texture window attribute of the current drawing environment (state of GPU).

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_uint_32	**code[2]**	***Reserved***

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_prim_draw_mode**, **brps_draw_environment**, **brps_prim_environment**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_uint_32 code[2]

Primitive data in bit fields.

Copy/Assign

Use copy by structure assignment freely.

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation.

Function for Standard Initialisation

BrPSPrimTextureWindowSet()

Description: Define a GPU texture window rendering control primitive

Declaration: **`void BrPSPrimTextureWindowSet(brps_prim_draw_mode* dm, brps_rectangle* rect)`**

Arguments: **`brps_prim_draw_mode * dm`**

A pointer to the destination draw mode primitive.

`brps_rectangle * rect`

Set new texture window. This window demarks a rectangular region within the current texture page. All subsequent texture co-ordinates will be offset from this region rather than from the origin of the texture page. Texture co-ordinates will be wrapped at the limites of the texture window rather than at the limits of the texture window.

See Also: **`brps_prim_draw_mode`**

brps_prim_draw_area

The Structure

This structure describes a rendering control primitive for the PlayStation. It is used to alter the current drawing area attribute of the current drawing environment (state of GPU). It can be used in an order table to change the drawing area while rendering.

The **`typedef`**

(See *`ps.h`* for precise declaration and ordering)

`brps_prim_tag`	**`tag`**	
`br_uint_32`	**`code[2]`**	***Reserved***

Related Functions

Order Tables

See **`br_order_table`**.

Related Macros

PlayStation Rendering Primitives

See `BrPSPrimAdd()`, `BrPSPrimCat()`, `BrPSPrimNext()`, `BrPSPrimTerminate()`.

Related Structures

See **`brps_prim_tag`**, **`brps_draw_environment`**, **`brps_prim_environment`**.

Members

`brps_prim_tag tag`

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

`br_uint_32 code[2]`

Primitive data in bit fields.

Copy/Assign

Use copy by structure assignment freely.

brps_prim_draw_area

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation.

Function for Standard Initialisation

BrPSPrimDrawAreaSet()

Description: Define a GPU drawing area rendering control primitive

Declaration: **`void BrPSPrimDrawAreaSet(brps_prim_draw_area* da, brps_rectangle* rect)`**

Arguments: **`brps_prim_draw_area * da`**

A pointer to the destination draw area primitive.

`brps_rectangle * rect`

Set new drawing area. This window demarks a rectangular region within the frame buffer to which rendering is restricted.

See Also: **`brps_draw_environment`**

brps_prim_draw_offset

The Structure

This structure describes a rendering control primitive for the PlayStation. It is used to alter the current drawing offset attribute of the current drawing environment (state of GPU). It can be used in an order table to change the drawing offset while rendering.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_uint_32	**code[2]**	***Reserved***

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_draw_environment**, **brps_prim_environment.**

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_uint_32 code[2]

Primitive data in bit fields.

Copy/Assign

Use copy by structure assignment freely.

`brps_prim_draw_offset`

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation.

Function for Standard Initialisation

BrPSPrimDrawOffsetSet()

Description: Define a GPU drawing area rendering control primitive

Declaration: **`void BrPSPrimDrawOffsetSet(brps_prim_draw_offset* do, br_vector2s* offset)`**

Arguments: **`brps_prim_draw_offset * do`**

A pointer to the destination draw offset primitive.

`br_vector2s * offset`

First and second ordinates. Set new drawing offset. This offset (x,y) demarks an offset within the current drawing area for rendering. The offset and address after addition of the offset are wrapped at (-1024, -1024) - (1023, 1023).

See Also: **`brps_draw_environment`**

brps_prim_environment

The Structure

This structure describes a rendering control primitive for the PlayStation. It is used to alter the entire current drawing environment (state of GPU). It can be used in an order table to change the drawing environment while rendering.

The **typedef**

(See *ps.h* for precise declaration and ordering)

brps_prim_tag	**tag**	
br_uint_32	**code[15]**	***Reserved***

Related Functions

Order Tables

See **br_order_table**.

Related Macros

PlayStation Rendering Primitives

See BrPSPrimAdd(), BrPSPrimCat(), BrPSPrimNext(), BrPSPrimTerminate().

Related Structures

See **brps_prim_tag**, **brps_draw_environment**.

Members

brps_prim_tag tag

Rendering primitive header. Contains low 24 bits of a pointer to the next primitive in the display list and length of rendering primitive in 32 bit units.

br_uint_32 code[15]

Primitive data in bit fields.

Copy/Assign

Use copy by structure assignment freely.

`brps_prim_environment`

Access & Maintenance

Do not modify members while display list traversal is being performed.

Initialisation

This structure is initialised by rendering primitive initialisation.

Function for Standard Initialisation

BrPSPrimDrawEnvironmentSet()

Description: Define a GPU drawing area rendering control primitive

Declaration: **`void BrPSPrimDrawOffsetSet(brps_prim_environment *e, brps_draw_environment* de)`**

Arguments: **`brps_prim_environment * e`**

A pointer to the destination environment primitive.

`brps_draw_environment * de`

New drawing environment to set (GPU state).

See Also: **`brps_draw_environment`**

brps_draw_environment

The Structure

This structure describes a drawing environment (GPU state) for the PlayStation. Attributes of the current drawing environment (state) can be altered whilst display list traversal being performed. Typical modifications are changing texture page or texture window.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_rectangle	**clip**	
br_vector2s	**offset**	
brps_rectangle	**texture_window**	
br_uint_16	**tpage**	
br_uint_8	**dither**	
br_uint_8	**display_draw**	
br_uint_8	**clear_screen**	
br_uint_8	**r,g,b**	
brps_prim_environment	**dr_env**	***Reserved***

Related Structures

See **brps_prim_environment**, **brps_display_environment**.

Members

brps_rectangle clip

Define a rectangular clip region within the frame buffer.

br_vector2s offset

Define a drawing offset (origin) within the clip region.

brps_rectangle texture_window

Define a rectangular clip region within the current texture page. Rendering primitive texture coordinates are offset from this window.

br_uint_16 tpage

Current texture page to use until a rendering primitive with a tpage member is encountered.

br_uint_8 dither

Rendering dither flag. 0: Off, 1: On.

br_uint_8 display_draw

Flag to control drawing to display area. The current drawing area is not necessarily the current display area to allow for double buffering. 0: Off, 1: On.

br_uint_8 clear_screen

Screen auto-clear flag. When the drawing environment is set, automatically clear the drawing area.

0: Off, 1: On.

br_uint_8 r,g,b

Colour for screen auto-clear.

Copy/Assign

Use copy by structure assignment freely. Do not modify **dr_env** member while display list traversal is being performed.

Initialisation

This structure is initialised by member-wise initialisation.

Macros for Standard Initialisation

BrPSDrawEnvironmentRGBSet(p, r, g, b)

Set screen auto-clear colour components of draw environment.

See **br_vector2s**, **brps_rectangle**.

See also BrPSRectangleSet().

BrPSDrawEnvironmentDefine()

Description: Define drawing environment from structure members

Declaration: **brps_draw_environment* BrPSDrawEnvironmentSet(brps_draw_environment* d, br_int_32 x, br_int_32 y, br_int_32 w, br_int_32 h)**

Arguments: **brps_draw_environment * d**

A pointer to drawing environment to be defined.

br_int_32 x,y

Upper left co-ordinates of drawing area.

br_int_32 w,h

Width and height of drawing area.

See Also: **brps_prim_environment**

BrPSDrawEnvironmentSet()

Description: Set drawing environment as current draw environment (reset GPU state).

Declaration: **`void BrPSDrawEnvironmentSet(brps_draw_environment* d)`**

Arguments: **`brps_draw_environment * d`**

A pointer to drawing environment to be set.

BrPSDrawEnvironmentGet()

Description: Get the current drawing environment (get GPU state).

Declaration: **`void BrPSDrawEnvironmentGet(brps_draw_environment* d)`**

Arguments: **`brps_draw_environment * d`**

A pointer to a buffer for the current drawing environment.

brps_display_environment

The Structure

This structure describes a display environment (GPU state) for the PlayStation.

The ***typedef***

(See *ps.h* for precise declaration and ordering)

brps_rectangle	**display**	
brps_rectangle	**screen**	
br_uint_8	**interlace**	
br_uint_8	**rgb24**	
br_uint_8	**type**	
br_uint_8	**_pad**	***Reserved***

Related Structures

See **brps_draw_environment**.

Members

brps_rectangle display

Display area within the frame buffer.

brps_rectangle screen

Output screen display area. The screen area is calculated without regard to thevalue of the display member, using the standard monitor screen upper-left point (0,0) and lower-right point (256,240).

br_uint_8 interlace

Interlace mode flag. 0: Non-interlaced, 1: Interlaced.

br_uint_8 rgb24

Display 24 bit mode flag. 0: 16 bit, 1: 24 bit.

br_uint_8 type

Display type flag. 0: PAL, 1: NTSC.

Copy/Assign

Use copy by structure assignment freely. Do not modify **dr_env** member while display list traversal is being performed.

Initialisation

This structure is initialised by member-wise initialisation.

Macros for Standard Initialisation

See **brps_rectangle**.

See also BrPSRectangleSet().

BrPSDisplayEnvironmentDefine()

Description: Define display environment from structure members

Declaration: **brps_display_environment* BrPSDrawEnvironmentSet(brps_display_environment* d, br_int_32 x, br_int_32 y, br_int_32 w, br_int_32 h)**

Arguments: **brps_display_environment * d**

A pointer to display environment to be defined.

br_int_32 x,y

Upper-left co-ordinates of display area.

br_int_32 w,h

Width and height of display area.

See Also: **brps_draw_environment**

BrPSDisplayEnvironmentSet()

Description: Set display environment as current display environment (GPU state).

Declaration: **void BrPSDisplayEnvironmentSet(brps_display_environment* d)**

Arguments: **brps_display_environment * d**

A pointer to display environment to be set.

BrPSDisplayEnvironmentGet()

Description: Get the current drawing environment (GPU state).

Declaration: **void BrPSDisplayEnvironmentGet(brps_display_environment* d)**

Arguments: **brps_display_environment * d**

A pointer to a buffer for the current display environment.

br_material

The Structure

Consult the Technical Reference Manual for full details of the **br_material** structure.

The ***typedef***

(See *material.h* for precise declaration and ordering)

```
br_uint_32          flags
br_colour           fog_colour          /* local fog colour */
br_uint_8           divide_level        /* level of recursion for polygon sub-division */
br_scalar           divide_hither       /* threshold for polygon sub-division */
br_uint_8           alpha_blend_rate
```

Members

br_uint_32 flags

This member determines how faces using the material are rendered, in terms of other members and aspects of the scene.

Flag Symbol	**Behaviour**
BR_MATF_LIGHT	The material is lit – affected by lights in the scene
BR_MATF_PRELIT	The material is pre-lit – colours are taken directly from models' vertex structures (see **br_vertex**). Any lights are ignored.
BR_MATF_SMOOTH	Any lighting is applied using Gouraud shading. Lighting levels are linearly interpolated between vertices. Otherwise, the same lighting level is used across the face
BR_MATF_DITHER	Effectively applies a filter to the screen to soften transitions between pixels.
BR_MATF_ALWAYS_VISIBLE	Faces using the material will always be visible, and so back-face culling need not be performed for such faces
BR_MATF_TWO_SIDED	The material has two sides, and lighting calculations are performed for both of them
BR_MATF_SEMI_TRANS	Perform semi-transparent rendering. Pixels marked with the high bit set (bit 15) will be rendered as semi-transparent. For 4 and 8 bit indexed texture pages, the palette entries must be marked, and for 16 bit direct colour the individual pixels must be marked. Semi-transparent rendering is slow as a screen read is needed per pixel.
BR_MATF_FOG_ENVIRONMENT	Fog material to an arbitrary global environment fogging colour. Textured materials may also be fogged to an arbitrary colour.
BR_MATF_FOG_LOCAL	Fog material to an arbitrary local fogging colour. Textured materials may also be fogged to an arbitrary colour.
BR_MATF_DEPTH_CUE	Fog material to black (rgb colour 0,0,0). This is faster than using environment or local fogging.

BR_MATF_DONT_SHADE	Perform textured rendering using just the colour value of the texels in the texture page. The brightness values of the polygons are ignored.
BR_MATF_SUB_DIVIDE	Perform polygon sub-division to reduce texture map distortion due to linear texture mapping. The polygon with this material is recursively sub-divided to minimise the perspective error when texture mapping.
BR_MATF_TILE	Allow a texture map to be tiled across a polygon more than once. If this flag is not specified and texture co-ordinates indicate a tiled texture, the texture mapping will be incorrect.

br_colour fog_colour

An arbitrary colour for fogging a material, including textured materials. Each material may have a different local fog value.

br_uint_8 divide_level

Recursion level for polygon sub-division. Polygons are divided into four smaller polygons at each recursive level. Although a deep level of recursion is desirered to reduce linear texture mapping distortions, this will consume a large amount of memory which is not available on the PlayStation.

br_scalar divide_hither

Threshold distance for polygon sub-division. Distance in front of view volume from camera along negative z axis. The value should be greater than zero.

br_uint_8 alpha_blend_rate

Semi-transparency rate for rendering.

Semi-Transparency Rate	Behaviour
0	0.5 back x 0.5 front
1	1.0 back x 1.0 front
2	0.5 back x 1.0 front
3	-1.0 back x 1.0 front

br_matrix3t

The Structure

A three column, four row, scalar array, used as a 3D affine matrix for general purpose 3D transformations (translation, rotation) on the PlayStation (low precision). Functions are provided to allow it to be used as though it were an integral type.

The ***typedef***

(See *matrix.h* for precise declaration and ordering)

br_int_16	**m[3][3]**	***Three rows of three columns***
br_vector3	**t**	***Fourth row (vector which has translational effect).***

Related Functions

Scene Modelling
See **BrActorToActorMatrix3t()**.

Related Structures

See **br_matrix34**.

Members

br_int_16 m[3][3]

Each element of the matrix can be freely abd individually accessed. The elements are in 3:12 signed fixed point format.

br_vector3 t

Fourth row of matrix. Vector which has translation effect.

Arithmetic

BrMatrix3tMul()

Description: Multiply two matrixes together and place the result in a third matrix.

Declaration: **void BrMatrix3tMul(br_matrix3t* A, const br_matrix3t * B, const br_matrix3t* C)**

Arguments: **`br_matrix3t * A`**

A pointer to the destination matrix (must be different from both sources).

`const br_matrix3t * B`

Pointer to the left hand source matrix.

`const br_matrix3t * C`

Pointer to the right hand source matrix.

See Also: **`BrMatrix34Mul()`**, **`BrMatrix3tPre()`**, **`BrMatrix3tPost()`**.

BrMatrix3tInverse()

Description: Compute the inverse of the supplied 3D affine matrix.

Declaration: **`br_scalar BrMatrix3tInverse(br_matrix3t* A, const br_matrix3t* B)`**

Arguments: **`br_matrix3t * A`**

A pointer to the destination matrix (must be different from source).

`const br_matrix3t * B`

A pointer to the source matrix.

Result: **`br_scalar`**

If the inverse exists, the determinant of the source matrix is returned. If there is no inverse, scalar zero is returned.

Remarks: Remember that while an inverse may be obtained using double precision arithmetic, this does not necessarily mean that it can using the **`br_scalar`** type. Only fixed point libraries are supplied for the PlayStation due to performance considerations.

See Also: **`BrMatrix34Inverse()`**, **`BrMAtrix3tLPInverse()`**.

BrMatrix3tLPInverse()

Description: Compute the inverse of the supplied length preserving transformation matrix. The resulting matrix is undefined for non-length oreserving matrixes.

Declaration: **`br_scalar BrMatrix3tLPInverse(br_matrix3t* A, const br_matrix3t* B)`**

Arguments: **`br_matrix3t * A`**

A pointer to the destination matrix (must be different from source).

`const br_matrix3t * B`

A pointer to the source matrix.

See Also: **`BrMatrix3tInverse()`**.

BrMatrix3tApply()

Description: Applies a transformation to a 3D point which may have non-unity homogenous co-ordinates.

Declaration: **`void BrMatrix3tApply(br_vector3* A, const br_vector4* B, const br_matrix3t* C)`**

Arguments: **`br_vector3 * A`**

A pointer to the destination vector (must be different from the source, and not part of the transformation), to hold the transformed point.

`const br_vector4 * B`

A pointer to the source vector, holding the point to be transformed.

`const br_matrix3t * c`

A pointer to the transformation matrix to be applied.

See Also: **`BrMatrix34Apply()`**.

BrMatrix3tApplyP()

Description: Applies a transformation to a 3D point.

Declaration: **`void BrMatrix3tApplyP(br_vector3* A, const br_vector3* B, const br_matrix3t* C)`**

Arguments: **`br_vector3 * A`**

A pointer to the destination vector (must be different from the source, and not part of the transformation), to hold the transformed point.

`const br_vector3 * B`

A pointer to the source vector, holding the point to be transformed.

`const br_matrix3t * c`

A pointer to the transformation matrix to be applied.

See Also: **`BrMatrix34ApplyP()`**, **`BrMAtrix34Apply()`**.

BrMatrix3tApplyV()

Description: Applies a transformation to a 3D vector, i.e. as for a point but without translation components (a vector has no location).

Declaration: **`void BrMatrix3tApplyP(br_vector3* A, const br_vector3* B, const br_matrix3t* C)`**

Arguments: **`br_vector3 * A`**

A pointer to the destination vector (must be different from the source, and not part of the transformation), to hold the transformed vector.

`const br_vector3 * B`

A pointer to the source vector, holding the vector to be transformed.

`const br_matrix3t * c`

A pointer to the transformation matrix to be applied.

See Also: **`BrMatrix34ApplyV()`**, **`BrMAtrix34Apply()`**.

BrMatrix3tTApply()

Description: Applies a transformation to a transposed 3D point which may have non-unity homogenous co-ordinates.

Declaration: **`void BrMatrix3tTApply(br_vector4* A, const br_vector4* B, const br_matrix3t* C)`**

Arguments: **`br_vector4 * A`**

A pointer to the destination vector (must be different from the source, and not part of the transformation), to hold the transformed point.

`const br_vector4 * B`

A pointer to the source vector, holding the point to be transformed.

`const br_matrix3t * c`

A pointer to the transformation matrix to be applied transposed.

See Also: **`BrMatrix34TApply()`**, **`BrMAtrix34Apply()`**.

BrMatrix3tTApplyP()

Description: Applies a transposed transform to a 3D point.

Declaration: **`void BrMatrix3tTApplyP(br_vector3* A, const br_vector3* B, const br_matrix3t* C)`**

Arguments: **`br_vector3 * A`**

A pointer to the destination vector (must be different from the source, and not part of the transformation), to hold the transformed point.

`const br_vector3 * B`

A pointer to the source vector, holding the point to be transformed.

`const br_matrix3t * c`

A pointer to the transform matrix to be applied transposed - the translation elements are presumed zero or irrelevant.

See Also: **`BrMatrix34TApplyP()`**, **`BrMAtrix34Apply()`**.

BrMatrix3tTApplyV()

Description: Applies a transposed transform to a 3D vector, i.e. as for a point but without translation components (a vector has no location).

Declaration: **`void BrMatrix3tTApplyP(br_vector3* A, const br_vector3* B, const br_matrix3t* C)`**

Arguments: **`br_vector3 * A`**

A pointer to the destination vector (must be different from the source, and not part of the transformation), to hold the transformed vector.

`const br_vector3 * B`

A pointer to the source vector, holding the vector to be transformed.

`const br_matrix3t * c`

A pointer to the transform matrix to be applied transposed - the translation elements are presumed zero or irrelevant.

See Also: **`BrMatrix34TApplyV()`**.

BrMatrix3tPre()

Description: Pre-multiply one matrix by another.

Declaration: **`void BrMatrix3tPre(br_matrix3t* A, const br_matrix3t* b)`**

Arguments: **`br_matrix3t * A`**

A pointer to the subject matrix (may be same as B).

`const br_matrix3t * B`

A pointer to the pre-multiplying matrix.

See Also: **`BrMatrix34Pre()`**.

BrMatrix3tPreScale()

Description: Pre-multiply a matrix by a scaling transform matrix.

Declaration: **`void BrMatrix3tPreScale(br_matrix3t* mat, br_scalar sx, br_scalar sy, br_scalar sz)`**

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix.

br_scalar sx

Scaling component along the x axis.

br_scalar sy

Scaling component along the y axis.

br_scalar sz

Scaling component along the z axis.

See Also: **BrMatrix34PreScale()**, **BrMatrix3tScale()**, **BrMatrix3tPostScale()**.

BrMatrix3tPreShearX()

Description: Pre-multiply a matrix by an x invariant shearing transform matrix.

Declaration: **void BrMatrix3tPreShearX(br_matrix3t* mat, br_scalar sy, br_scalar sz)**

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix.

br_scalar sy

Shear factor by which the x co-ordinate is included in the transformed y co-ordinate.

br_scalar sz

Shear factor by which the x co-ordinate is included in the transformed z co-ordinate.

See Also: **BrMatrix34PreShearX()**, **BrMatrix3tShearX()**, **BrMatrix3tPostShearX()**.

BrMatrix3tPreShearY()

Description: Pre-multiply a matrix by a y invariant shearing transform matrix.

Declaration: **void BrMatrix3tPreShearY(br_matrix3t* mat, br_scalar sx, br_scalar sz)**

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix.

br_scalar sx

Shear factor by which the y co-ordinate is included in the transformed x co-ordinate.

br_scalar sz

Shear factor by which the y co-ordinate is included in the transformed z co-ordinate.

See Also: **BrMatrix34PreShearY()**, **BrMatrix3tShearY()**, **BrMatrix3tPostShearY()**.

BrMatrix3tPreShearZ()

Description: Pre-multiply a matrix by a z invariant shearing transform matrix.

Declaration: **void BrMatrix3tPreShearZ(br_matrix3t* mat, br_scalar sx, br_scalar sy)**

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix.

br_scalar sx

Shear factor by which the z co-ordinate is included in the transformed x co-ordinate.

br_scalar sy

Shear factor by which the z co-ordinate is included in the transformed y co-ordinate.

See Also: **BrMatrix34PreShearZ()**, **BrMatrix3tShearZ()**, **BrMatrix3tPostShearZ()**.

BrMatrix3tPreTranslate()

Description: Pre-multiply a matrix br a translation transform matrix.

Declaration: **void BrMatrix3tPreTranslate(br_matrix3t* mat, br_scalar dx, br_scalar dy, br_scalar dz)**

Arguments: **br_matrix3t * mat**

A pointer to the subject matrix.

br_scalar dx

The x axis component used to form the translation matrix.

br_scalar dy

The yaxis component used to form the translation matrix.

br_scalar dz

The z axis component used to form the translation matrix.

See Also: **BrMatrix34PostTranslate()**, **BrMatrix34Translate()**, **BrMatrix3tTranslate()**.

BrMatrix3tPreRotate()

Description: Pre-multiply a matrix by a vector specified axis, rotational transform matrix.

Declaration: **void BrMatrix3tPreRotate(br_matrix3t* mat, br_angle r, const br_vector3* axis)**

Arguments: **`br_matrix3t * mat`**

A pointer to the subject matrix.

`br_angle r`

The angle about the specified axis used to form the rotation matrix. A positive angle represents a clockwise rotation (with a vector pointing at you).

`const br_vector3*`

The arbitrary (normalised) axis vector about which the rotation occurs.

`br_scalar dz`

The z axis component used to form the translation matrix.

See Also: **`BrMatrix34PostRotate()`**, **`BrMatrix34Rotate()`**, **`BrMatrix3tPostRotate()`**, **`BrMatrix3tRotate()`**.

BrMatrix3tPreRotateX()

Description: Pre-multiply a matrix by an x axis rotational transform matrix.

Declaration: **`void BrMatrix3tPreRotateX(br_matrix3t* mat, br_angle rx)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the subject matrix.

`br_angle rx`

The angle about the x axis used to form the rotation matrix. A positive angle represents a clockwise rotation (looking toward the origin).

See Also: **`BrMatrix34PostRotateX()`**, **`BrMatrix34RotateX()`**, **`BrMatrix3tPostRotateX()`**, **`BrMatrix3tRotateX()`**.

BrMatrix3tPreRotateY()

Description: Pre-multiply a matrix by a y axis rotational transform matrix.

Declaration: **`void BrMatrix3tPreRotateY(br_matrix3t* mat, br_angle ry)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the subject matrix.

`br_angle ry`

The angle about the y axis used to form the rotation matrix. A positive angle represents a clockwise rotation (looking toward the origin).

See Also: **`BrMatrix34PostRotateY()`**, **`BrMatrix34RotateY()`**, **`BrMatrix3tPostRotateY()`**, **`BrMatrix3tRotateY()`**.

BrMatrix3tPreRotateZ()

Description: Pre-multiply a matrix by an zaxis rotational transform matrix.

Declaration: **`void BrMatrix3tPreRotateZ(br_matrix3t* mat, br_angle rz)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the subject matrix.

`br_angle rz`

The angle about the z axis used to form the rotation matrix. A positive angle represents a clockwise rotation (looking toward the origin).

See Also: **`BrMatrix34PostRotateZ()`**, **`BrMatrix34RotateZ()`**, **`BrMatrix3tPostRotateZ()`**, **`BrMatrix3tRotateZ()`**.

BrMatrix3tPreTransform()

Description: Pre-multiply a matrix by a generic transform.

Declaration: **`void BrMatrix3tPreTransform(br_matrix3t* mat, const br_transform* xform)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the subject matrix.

`const br_transform * xform`

The pre-multiplying generic transform.

Effects: The transform is first converted to a general 3x4 transform matrix using **`BrTransformToMatrix3t()`** and then applied as a pre-multiplying matrix using **`BrMatrix3tPre()`**.

See Also: **`BrMatrix34PostTransform()`**, **`BrMatrix3tPostTransform()`**.

BrMatrix3tPost()

Description: Post-multiply one matrix by another.

Declaration: **`void BrMatrix3tPost(br_matrix3t* A, const br_matrix3t* B)`**

Arguments: **`br_matrix3t * A`**

A pointer to the subject matrix (may be same as B).

`const br_matrix * B`

A pointer to the post-multiplying matrix.

See Also: **`BrMatrix34Pre()`**, **`BrMatrix34Mul()`**, **`BrMatrix3tPre()`**, **`BrMatrix3tMul()`**.

BrMatrix3tPostTranslate()

Description: Post-multiply one matrix by a a translation transform matrix.

Declaration: **`void BrMatrix3tPostTranslate(br_matrix3t* mat, br_scalar dx, br_scalar dy, br_scalar dz)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the subject matrix.

`br_scalar dx`

The x axis component used to form the translation matrix.

`br_scalar dy`

The y axis component used to form the translation matrix.

`br_scalar dz`

The z axis component used to form the translation matrix.

See Also: **`BrMatrix34PreTranslate()`**, **`BrMatrix34Translate()`**, **`BrMatrix3tPreTranslate()`**, **`BrMatrix3tTranslate()`**.

BrMatrix3tPostScale()

Description: Post-multiply a matrix by a scaling transform matrix.

Declaration: **`void BrMatrix3tPostScale(br_matrix3t* mat, br_scalar sx, br_scalar sy, br_scalar sz)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the destination matrix.

`br_scalar sx`

Scaling component along the x axis.

`br_scalar sy`

Scaling component along the y axis.

`br_scalar sz`

Scaling component along the z axis.

See Also: **`BrMatrix34PostScale()`**, **`BrMatrix3tScale()`**, **`BrMatrix3tPreScale()`**.

BrMatrix3tPostShearX()

Description: Post-multiply a matrix by an x invariant shearing transform matrix.

Declaration: **`void BrMatrix3tPostShearX(br_matrix3t* mat, br_scalar sy, br_scalar sz)`**

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix.

br_scalar sy

Shear factor by which the x co-ordinate is included in the transformed y co-ordinate.

br_scalar sz

Shear factor by which the x co-ordinate is included in the transformed z co-ordinate.

See Also: **BrMatrix34PostShearX()**, **BrMatrix3tShearX()**, **BrMatrix3tPreShearX()**.

BrMatrix3tPostShearY()

Description: Post-multiply a matrix by a y invariant shearing transform matrix.

Declaration: **void BrMatrix3tPostShearY(br_matrix3t* mat, br_scalar sx, br_scalar sz)**

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix.

br_scalar sx

Shear factor by which the y co-ordinate is included in the transformed x co-ordinate.

br_scalar sz

Shear factor by which the y co-ordinate is included in the transformed z co-ordinate.

See Also: **BrMatrix34PostShearY()**, **BrMatrix3tShearY()**, **BrMatrix3tPreShearY()**.

BrMatrix3tPostShearZ()

Description: Post-multiply a matrix by a z invariant shearing transform matrix.

Declaration: **void BrMatrix3tPostShearZ(br_matrix3t* mat, br_scalar sx, br_scalar sy)**

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix.

br_scalar sx

Shear factor by which the z co-ordinate is included in the transformed x co-ordinate.

br_scalar sy

Shear factor by which the z co-ordinate is included in the transformed y co-ordinate.

See Also: **BrMatrix34PostShearZ()**, **BrMatrix3tShearZ()**, **BrMatrix3tPreShearZ()**.

BrMatrix3tPostRotate()

Description: Post-multiply a matrix by a vector specified axis, rotational transform matrix.

Declaration: **`void BrMatrix3tPostRotate(br_matrix3t* mat, br_angle r, const br_vector3* axis)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the subject matrix.

`br_angle r`

The angle about the specified axis used to form the rotation matrix.

`const br_vector3 *`

The arbitrary (normalised) axis vector about which the rotation occurs.

See Also: **`BrMatrix34PreRotate()`**, **`BrMatrix34Rotate()`**, **`BrMatrix3tPreRotate()`**, **`BrMatrix3tRotate()`**.

BrMatrix3tPostRotateX()

Description: Post-multiply a matrix by an x axis rotational transform matrix.

Declaration: **`void BrMatrix3tPostRotateX(br_matrix3t* mat, br_angle rx)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the subject matrix.

`br_angle rx`

The angle about the x axis used to form the rotation matrix. A positive angle represents a clockwise rotation (looking toward the origin).

See Also: **`BrMatrix34PreRotateX()`**, **`BrMatrix34RotateX()`**, **`BrMatrix3tPreRotateX()`**, **`BrMatrix3tRotateX()`**.

BrMatrix3tPostRotateY()

Description: Post-multiply a matrix by a y axis rotational transform matrix.

Declaration: **`void BrMatrix3tPostRotateY(br_matrix3t* mat, br_angle ry)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the subject matrix.

`br_angle ry`

The angle about the y axis used to form the rotation matrix. A positive angle represents a clockwise rotation (looking toward the origin).

See Also: **`BrMatrix34PreRotateY()`**, **`BrMatrix34RotateY()`**, **`BrMatrix3tPreRotateY()`**, **`BrMatrix3tRotateY()`**.

BrMatrix3tPostRotateZ()

Description: Post-multiply a matrix by an zaxis rotational transform matrix.

Declaration: **`void BrMatrix3tPostRotateZ(br_matrix3t* mat, br_angle rz)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the subject matrix.

`br_angle rz`

The angle about the z axis used to form the rotation matrix. A positive angle represents a clockwise rotation (looking toward the origin).

See Also: **`BrMatrix34PretRotateZ()`**, **`BrMatrix34RotateZ()`**, **`BrMatrix3tPreRotateZ()`**, **`BrMatrix3tRotateZ()`**.

BrMatrix3tPostTransform()

Description: Post-multiply a matrix by a generic transform.

Declaration: **`void BrMatrix3tPostTransform(br_matrix3t* mat, const br_transform* xform)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the subject matrix.

`const br_transform * xform`

The pre-multiplying generic transform.

Effects: The transform is first converted to a general 3x4 transform matrix using **`BrTransformToMatrix3t()`** and then applied as a pre-multiplying matrix using **`BrMatrix3tPost()`**.

See Also: **`BrMatrix34PreTransform()`**, **`BrMatrix3tPreTransform()`**.

Conversion

Note that only matrixies can represent the full gamut of translation, shearing, reflection and scaling effects, some of these effects will be lost (or produce undefined behaviour) when converting into another transformation.

From Eulers, Quaternions and Transforms

See **`BrEulerToMatrix3t()`**, **`BrQuatToMatrix3t()`**, **`BrTransformToMatrix3t()`**.

Also see **`BrTransformToTransform()`**.

To Eulers, Quaternions and Transforms

See **`BrMatrix3tToEuler()`**, **`BrMatrix3tToQuat()`**, **`BrMatrix3tToTransform()`** as described below.

Also see **`BrTransformToTransform()`**.

BrMatrix3tToEuler()

Description: Convert a 3D affine matrix to a Euler angle set, that would have the same rotational effect.

Declaration: **`br_euler* BrMatrix3tToEuler(br_euler* euler, const br_matrix3t* mat)`**

Arguments: **`br_euler * euler`**

A pointer to the destination Euler angle set to receve the conversion. The Euler angle set's Euler order is used to determine each angle.

`const br_matrix3t * mat`

A pointer to the source matrix to convert from.

Result: **`br_euler *`**

Returns euler for convenience.

Remarks: Translation components of the matrix are lost in conversion.

BrMatrix3tToQuat()

Description: Convert a 3D affine matrix to a quaternion, that would have the same rotational effect.

Declaration: **`br_quat* BrMatrix3tToQuat(br_quat* q, const br_matrix3t* mat)`**

Arguments: **`br_quat * q`**

A pointer to the destination quaternion to receive the conversion.

`const br_matrix3t * mat`

A pointer to the source matrix to convert from.

Result: **`br_quat *`**

Returns q for convenience.

Remarks: Translation components of the matrix are lost in conversion.

BrMatrix3tToTransform()

Description: Convert a 3D affine matrix toto a specific transform, that would have a similar transformational effect.

Declaration: **`void BrMatrix3tToTransform(br_transform* xform, const br_matrix3t* mat)`**

Arguments: **`br_transform * xform`**

A pointer to the destination transform. The type member of the destination transform is retained and determines the method of conversion.

`const br_matrix3t * mat`

A pointer to the source matrix to be converted.

See Also: **`BrMatrix34ToTransform()`**.

Copy/Assign

Although copy by structure assignment currently works, use **`BrMatrix3tCopy()`** or **`BrMatrix3tCopy34()`** to ensure compatibility.

BrMatrix3tCopy()

Description: Copy a matrix.

Declaration: **`void BrMatrix3tCopy(br_matrix3t* A, const br_matrix3t* B)`**

Arguments: **`br_matrix * A`**

A pointer to the destination matrix (may be the same as source - though redundant).

`const br_matrix3t * B`

A pointer to the source matrix.

See Also: **`BrTransformToTransform()`**.

BrMatrix3tCopy34()

Description: Copy a 3x4 high precision matrix to a lower precision 3x4 matrix.

Declaration: **`void BrMatrix3tCopy34(br_matrix3t* A, const br_matrix34* B)`**

Arguments: **`br_matrix3t * A`**

A pointer to the destination matrix (lower precision).

`const br_matrix34 * B`

A pointer to the source 3x4 matrix (high precision).

See Also: **`BrMatrix34Copy3t()`**.

BrMatrix3tCopy4()

Description: Copy a 4x4 high precision matrix to a lower precision 3x4 matrix, discarding right-hand column.

Declaration: **`void BrMatrix3tCopy4(br_matrix3t* A, const br_matrix4* B)`**

Arguments: **`br_matrix3t * A`**

A pointer to the destination matrix (lower precision).

`const br_matrix4 * B`

A pointer to the source 4x4 matrix.

See Also: **BrMatrix4Copy3t()**.

Access & Maintenance

Members may be freely accessed. Maintenance is only required for length preserving matrixes that have been modified.

BrMatrix3tLPNormalise()

Description: Normalise a length preserving matrix.

Declaration: **void BrMatrix3tLPNormalise(br_matrix3t* A, const br_matrix3t* B)**

Arguments: **br_matrix3t * A**

A pointer to the destination matrix, which must not point to the source matrix.

const br_matrix3t * B

A pointer to the source matrix.

Effects: The destination matrix is the souce matrix adjusted to that it represents a length preserving transformation.

Remarks: This function is typically applied to a length preserving matrix which has undergone a long sequence of operations, to ensure that the final matrix is still truly length preserving (regardless of rounding errors).

See Also: **BrMatrix34LPNormalise()**.

Referencing & Lifetime

This structure may be freely referenced, though take care if there is potential to supply the same matrix as more than one argument to the same funciton.

Initialisation

No static initialisers are provided. However, four `BR_VECTOR3()` macros would serve as well. All other initialisation should use **BrMatrix3tCopy()** or any of the following initialisation functions.

BrMatrix3tIdentity()

Description: Set the specified matrix to the identity transformation matrix.

Declaration: **void BrMatrix3tIdentity(br_matrix3t* mat)**

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix.

Effects: Stores the identity matrix at the destination.

BrMatrix3tTranslate()

Description: Set the specified matrix to a matrix representing a specific translation.

Declaration: **void BrMatrix3tTranslate(br_matrix3t* mat, br_scalar dx, br_scalar dy, br_scalar dz)**

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix.

br_scalar dx

Translation component along the x axis.

br_scalar dy

Translation component along the y axis.

br_scalar dz

Translation component along the z axis.

See Also: **BrMatrix34Translate()**, **BrMatrix3tPreTranslate()**, **BrMatrix3tPostTranslate()**.

BrMatrix3tScale()

Description: Set the specified matrix to a matrix representing a specific scaling.

Declaration: **void BrMatrix3tScale(br_matrix3t* mat, br_scalar sx, br_scalar sy, br_scalar sz)**

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix.

br_scalar sx

Scaling component along the x axis.

br_scalar sy

Scaling component along the y axis.

br_scalar sz

Scaling component along the z axis.

See Also: **BrMatrix34Scale()**, **BrMatrix3tPreScale()**, **BrMatrix3tPostScale()**.

BrMatrix3tShearX()

Description: Set the specified matrix to a matrix representing a shear, invariant along the x axis. Thus values of y and z co-ordinates will be scaled in proportion to the value of the x co-ordinate.

Declaration: **`void BrMatrix3tShearX(br_matrix3t* mat, br_scalar sy, br_scalar sz)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the destination matrix.

`br_scalar sy`

Shear factor by which the x co-ordinate is included in the transformed y co-ordinate.

`br_scalar sz`

Shear factor by which the x co-ordinate is included in the transformed z co-ordinate.

See Also: **`BrMatrix34ShearX()`**, **`BrMatrix3tPreShearX()`**, **`BrMatrix3tPostShearX()`**.

BrMatrix3tShearY()

Description: Set the specified matrix to a matrix representing a shear, invariant along the y axis. Thus values of x and z co-ordinates will be scaled in proportion to the value of the y co-ordinate.

Declaration: **`void BrMatrix3tShearY(br_matrix3t* mat, br_scalar sx, br_scalar sz)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the destination matrix.

`br_scalar sx`

Shear factor by which the y co-ordinate is included in the transformed x co-ordinate.

`br_scalar sz`

Shear factor by which the y co-ordinate is included in the transformed z co-ordinate.

See Also: **`BrMatrix34ShearY()`**, **`BrMatrix3tPreShearY()`**, **`BrMatrix3tPostShearY()`**.

BrMatrix3tShearZ()

Description: Set the specified matrix to a matrix representing a shear, invariant along the z axis. Thus values of x and y co-ordinates will be scaled in proportion to the value of the z co-ordinate.

Declaration: **`void BrMatrix3tShearZ(br_matrix3t* mat, br_scalar sx, br_scalar sy)`**

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix.

br_scalar sx

Shear factor by which the z co-ordinate is included in the transformed x co-ordinate.

br_scalar sy

Shear factor by which the z co-ordinate is included in the transformed y co-ordinate.

See Also: **BrMatrix34ShearZ()**, **BrMatrix3tPreShearZ()**, **BrMatrix3tPostShearZ()**.

BrMatrix3tRotateX()

Description: Set the specified matrix to a matrix representing a rotation about the x axis through a specified angle.

Declaration: **void BrMatrix3tRotateX(br_matrix3t* mat, br_angle rx)**

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix.

br_angle rx

Rotation about the x axis.

See Also: **BrMatrix34RotateX()**, **BrMatrix3tPreRotateX()**, **BrMatrix3tPostRotateX()**.

BrMatrix3tRotateY()

Description: Set the specified matrix to a matrix representing a rotation about the y axis through a specified angle.

Declaration: **void BrMatrix3tRotateY(br_matrix3t* mat, br_angle ry)**

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix.

br_angle ry

Rotation about the y axis.

See Also: **BrMatrix34RotateY()**, **BrMatrix3tPreRotateY()**, **BrMatrix3tPostRotateY()**.

BrMatrix3tRotateZ()

Description: Set the specified matrix to a matrix representing a rotation about the x axis through a specified angle.

Declaration: **void BrMatrix3tRotateZ(br_matrix3t* mat, br_angle rz)**

Arguments: **`br_matrix3t * mat`**

A pointer to the destination matrix.

`br_angle rz`

Rotation about the z axis.

See Also: **`BrMatrix34RotateZ()`**, **`BrMatrix3tPreRotateZ()`**, **`BrMatrix3tPostRotateZ()`**.

BrMatrix3tRotate()

Description: Set the specified matrix to a matrix representing a rotation about a given axis vector through a specified angle.

Declaration: **`void BrMatrix3tRotate(br_matrix3t* mat, br_angle r, const br_vector3* a)`**

Arguments: **`br_matrix3t * mat`**

A pointer to the destination matrix.

`br_angle r`

Rotation about the specified axis vector.

`const br_vector3 * a`

The arbitrary (normalised) axis vector about which the rotation occurs.

See Also: **`BrMatrix34PreRotate()`**, **`BrMatrix34PostRotate()`**, **`BrMatrix3tPreRotate()`**, **`BrMatrix3tPostRotate()`**.

BrMatrix3tScreenMatrixSet()

Description: Set the specified matrix to be the current model to screen matrix used by the PlayStation hardware.

Declaration: **`void BrMatrix3tRotate(const br_matrix3t* mat)`**

Arguments: **`const br_matrix3t * mat`**

A pointer to the source matrix.

Effects: The current hardware (GTE) matrix is set to the source matrix.

br_matrix34

The Structure

Consult the Technical Reference Manual for full details of the **br_matrix34** structure..

Copy/Assign

The following additional function is relevant to the **br_matrix34** structure.

BrMatrix34Copy3t()

Description: Copy a 3x4 low precision matrix to a higher precision 3x4 matrix.

Declaration: **void BrMatrix34Copy3t(br_matrix3t* A, const br_matrix34* B)**

Arguments: **br_matrix34 * A**

A pointer to the destination matrix (high precision).

const br_matrix3t * B

A pointer to the source 3x4 matrix (low precision).

See Also: **BrMatrix3tCopy34()**.

br_matrix4

The Structure

Consult the Technical Reference Manual for full details of the **br_matrix4** structure.

Copy/Assign

The following additional function is relevant to the **br_matrix4** structure.

BrMatrix4Copy3t()

Description: Copy a 3x4 low precision matrix to a higher precision 4x4 matrix.

Declaration: **void BrMatrix4Copy3t(br_matrixt* A, const br_matrix34* B)**

Arguments: **br_matrix4 * A**

A pointer to the destination matrix (high precision).

const br_matrix3t * B

A pointer to the source 3x4 matrix (low precision).

Effects: The source is copied into the destination, and the fourth column of the destination is set to the implicit (0,0,0,1) column vector.

See Also: **BrMatrix3tCopy34()**.

br_model

The Structure

Consult the Technical Reference Manual for full details of the **br_model** structure.

Import & Export

BrFmtTMDLoadMany()

Description: Load a model in the Sony TMD format.

Declaration: **br_uint_32 BrFmtTMDLoadMany(const char* filename, const br_model** models, br_uint_16 num)**

Arguments: **const char * filename**

Name of the file containing the model or models to load.

const br_model ** models

A non-NULL pointer to an array of pointers to models.

br_uint_16 num

Maximum number of models to load.

Effects: Searches for filename. Material references are resolved by a registry search. A **br_material_find_cbfn** hook function may be used to generate new materials if this search fails. The search pattern used is a zero terminated hexadecimal string representing the material flags.

Colour map references are also resolved by a a registry search. A **br_map_find_cbfn** hook function may be used to generate new pixel maps if this search fails. The search pattern used is a zero terminated hexadecimal string representing the pixel map flags.

Result: **br_uint_32**

Returns the number of models loaded successfully. The pointer array if supplied, is filled with pointers to the loaded models.

Remarks: Only models composed of triangular polygons are loaded. Sprites, lines and quadrilateral primitives are ignored.

See Also: **BrMaterialFindHook()**, **BrMapFindHook()**.

br_pixelmap

The Structure

Consult the Technical Reference Manual for full details of the **br_pixelmap** structure.

The ***typedef***

(See *pixelmap.h* for precise declaration and ordering)

```
br_uint_8               type
br_uint_16              tpage
br_uint_16              clut
```

Members

br_uint_16 type

This member defines the type of data stored for each pixel in the pixel map. The various types have values defined by the following symbols:

Pixel Map Type	**Pixel Map Behaviour**
BR_PMT_INDEX_4	4 bit index into a colour map (16 colours)
BR_PMT_INDEX_8	8 bit index into a colour map (256 colours)
BR_PMT_BGR_555	16 bit ‘true colour’ RGB, 5 bits each colour, 1 bit transparency rate.

Pixel Map Type	**32 Bit Pixel Value Encoding**	**First Four Bytes[a] of Left Hand Pixel**
BR_PMT_INDEX_4	0000000000000000000000000000iiii	iiii.... ....
BR_PMT_INDEX_8	000000000000000000000000iiiiiiii	iiiiiiii ....
BR_PMT_BGR_555	0000000000000000tbbbbbgggggrrrrr	gggrrrrr tbbbbbgg ....

[a] The left hand byte is the byte atpixels .

All values are written with the most significant bit to the left.

The Encoding column represents the 32 bit value to be supplied as colour to functions such as **BrPixelmapPixelSet()** . The last column shows how the first pixel on a row will appear in the first four bytes indexed from pixels . The dots represent further pixels. The ordering of bytes pixel maps is independent of word byte order.

br_uint_16 tpage

This member determines indicates the encoded location of the pixelmap’s pixels within the frame buffer.

See BrPixelmapTPageSet().

br_uint_16 clut

This member determines indicates the encoded location of the pixelmap's clut within the frame buffer. This member is only used when using 4 or 8 bit indexed pixelmaps.

See `BrPixelmapClutSet()`.

Import & Export

BrFmtTIMLoad()

Description: Load a pixel map in Sony TIM format.

Declaration: **`br_pixelmap *BrFmtTIMLoad(const char* filename, br_uint_32 flags)`**

Arguments: **`const char * filename`**

Name of the file containing the pixel map to load.

`br_uint_32 flags`

Zero (ignored).

Effects: Searches for `filename`. 4 bit indexed, 8 bit indexed, 16 bit high colour and 24 bit true colour images are supported. Note however, bitmaps are in blue-green-red format for the PlayStation, not the traditional BRender red-green-blue format.

Result: **`br_pixelmap *`**

A pointer to the pixel map loaded.

See Also: **`BrFmtGifLoad()`**, **`BrFmtIFFLoad()`**, **`BrFmtBMPLoad()`**, **`BrFmtTGALoad()`**.

br_transform

The Structure

This is BRender's generic transform type, primarily used to specify a transformation from one actor's space to another's. In an actor it represents the transform to be applied to co-ordinates (such as of a model) in its space to bring them into the co-ordinate space of its parent. The structure has been extended for use with dedicated PlayStation hardware.

The ***`typedef`***

(See *`transform.h`* for precise declaration and ordering)

Transform Type

`br_uint_16`	**`type`**	***Specifies how the transformation is represented***

Translation Transform

`br_vector3`	**`t.translate.t`**	***Translation for the Translation transform type***

Euler Transform

`br_vector3`	**`t.euler.t`**	***Translation for the Euler transform type***
`br_euler`	**`t.euler.e`**	***Euler angle set of the Euler transform type***

Look Up Transform

`br_vector3`	**`t.loop_up.t`**	***Translation for the Loop Up transform type***
`br_vector3`	**`t.look_up.look`**	***Look-at vector for the Look Up transform type***
`br_vector3`	**`t.look_up.up`**	***Look-up vector for the Look Up transform type***

Quaternion Transform

`br_vector3`	**`t.quat.t`**	***Translation for the Quaternion transform type***
`br_quat`	**`t.quat.q`**	***Quaternion rotation for the Quaternion transform***

Matrix Transform

`br_matrix34`	**`t.mat`**	***Translation for Length Preserving and Non-Length Preserving Matrix transform types***

Matrix Transform (Low Precision for PlayStation)

`br_matrix3t`	**`t.mat3t.mat`**	***Translation for Matrix transform type***

Members

See the Technical Reference Manual for a precise definition of members.

`br_uint_16 type`

This member defines which other members of the transform structure have meaning. It should never be modified directly except for initialisation purposes. Refer to **`BrTransformToTransform()`** for details of how to convert from one transform to another.

This member may have any one of the following values:

Value Symbol	Meaning
BR_TRANSFORM_IDENTITY	The transform is the identity.
BR_TRANSFORM_TRANSLATION	The transform is a translation only (held in `t.translate.t`).
BR_TRANSFORM_EULER	The transform is represented by a Euler angle set (`t.euler.e`) and a translation (`t.euler.t`).
BR_TRANSFORM_LOOK_UP	The transform is represented by a look-at vector (`t.look_up.look`), an up vector (`t.look_up.up`) and a translation (`t.look_up.t`).
BR_TRANSFORM_QUAT	The transform is represented by a quaternion (`t.quat.q`) and a translation (`t.quat.t`).
BR_TRANSFORM_MATRIX34	The transform is represented by a 3x4 affine matrix (`t.mat`), which is the most general representation.
BR_TRANSFORM_MATRIX34_LP	The transform is represented by a 3x4 length preserving matrix (`t.mat`).
BR_TRANSFORM_MATRIX3T	The transform is represented by a low precision 3x4 affine matrix (t.mat3t.mat) which is the most general representation.

br_matrix3t t.mat3t.mat

This member contains the 3D affine matrix representing the entire transform. It is recommended to use this transform type on the PlayStation as all **br_matrix3t** are performed using dedicated hardware.

Conversion

See the Technical Reference Manual for full details of conversion functions.

BrTransformToMatrix3t()

Description: Convert a generic transform to a 3D affine matrix (low precision), that would have the same transformational effect.

Declaration: **void BrTransformToMatrix3t(br_matrix3t* mat,**
const br_transform* xform)

Arguments: **br_matrix3t * mat**

A pointer to the destination matrix to receive the conversion.

const br_transform * xform

A pointer to the source generic transform.

Effects: See **BrTransformToMatrix34()**.

PlayStation Programming Tips

General Performance Issues

The following is a list of general programming issues which the applications programmer must be aware of affecting the performance of the PlayStation.

The CPU

The host processor (CPU) is a customised variant of an R3000A running at 33.8688 MHz. It two co-processors attached. The GTE (geometry engine) is used to accelerate certain mathematical operations needed for 3D algorithms, and the GPU (rendering engine) operates upon primitive drawing instructions to place pixels into the frame buffer.

- The CPU has not floating point unit. All floating point operations are performed in software.
- The CPU has a 4KByte instruction cache, but no data cache. The instruction cache will not act as a mixed instruction/data cache.
- The CPU has an internal 1KByte of mapped memory which can be used by an application programmer as a data cache. Instructions may not be executed from this ‘scratch pad’. This ‘scratch pad’ has one cycle read and write operations.

The PlayStation has 2MBytes of main memory installed. The CPU has no virtual memory management hardware so the relationship between physical and logical memory is fixed. The frame buffer memory is a 2-dimensional address space which can only be accessed by the GPU.

- Main memory accessing is slow.
- The GTE and GPU are clocked at a higher frequency than the CPU. They have been optimised for certain operations and can out perform the CPU.
- The co-processors rely on the CPU for movement of data to and from main memory.

Please consult the Sony Reference for further details of the PlayStation system specification.

The Main Memory

The CPU has a read buffer (R buffer) consisting of four 4 byte registers for reading data. It takes four clock cycles to read from memory to the R buffer with a further one cycle from the R buffer to the CPU. The main memory is divided into 1KByte pages. Subsequent reads from the same memory page will only take two cycle. A page miss (reading from a new page) will require four cycles again.

- It takes four cycles to load a 32 bit (dword) value into an R buffer register. Subsequent reads from the same memory page take two cycles.
- It takes one cycle to transfer data from an R buffer register to the CPU.

- A read resulting in a page 'miss' (requiring a new memory page) will take four cycles.

The CPU also has a write buffer (W buffer) consisting of four 4 byte registers which operate as a FIFO queue. A write operation to the W buffer will take one cycle, so the CPU can execute a new instruction without waiting for the write operation to memory to be completed. It takes four cycles for data to be written from the W buffer to main memory. An instruction that causes data to be read from the address where data in the W buffer will be written will cause the W buffer to be flushed before the instruction is executed. The CPU will wait until a slot is available in the W buffer before writing data.

- It takes one cycle to write from the CPU to the W buffer
- It takes four cycles to write data from the W buffer to the main memory.
- The CPU will not wait for data to be written to main memory, unless a read instruction requires data from the W buffer before it has been committed to main memory.
- If the W buffer is full, the CPU will wait for data to be transferred to main memory.

To optimise memory accessing by the CPU, observe these guidelines.

- Interleave read and write operations to the same memory page.
- Avoid a large number of memory writes.
- Try to group related data in the same memory page.
- Use an efficient memory allocation/deallocation scheme. Memory management will affect performance.

The CPU Instruction Cache

The I cache (instruction cache) is used to preload instructions. The 4KByte cache is divided into 256 'lines', each 16 bytes in length. Each line can contain 4 instructions. When reading an instruction, the CPU checks the I cache to see if the instruction is present. If it is present, the CPU reads the instruction. This takes one clock cycle. If the instruction is not present, the CPU loads the target line into the cache and then the required instruction from the cache. This takes between four and seven cycles to load a cache line, and one cycle to load the instruction from the cache.

- An I cache 'hit' takes one cycle
- A 'miss' takes between five and eight cycles to load an instruction.

To minimise the frequency of cache 'misses', observe the following guidelines.

- Keep iterative sections of code short without function calls to non-local code. Keep relevant code within the same module.

The CPU 'scratch pad'

The scratch pad is 1KByte of memory in the CPU available for use by an application programmer. Read and write operations to this memory take one cycle. Instructions may not be executed from this memory. The scratch pad is not available for DMA transfers.

- The scratch pad can be used as a data cache. Cache management is performed by the application, not theCPU.

The C Compiler

The compiler supplied by Sony is GNU C.

- Ensure C functions have no more than four arguments. The C compiler supplied by Sony passes the first four arguments in registers, and the rest on the stack. This means many unnecessary memory accesses.
- Time critical sections of code may have to be written in assembler. Take care when hand optimising assembler code. MIPS assembler code can be very inefficient when instruction ordering is not optimal. If in doubt, see how the compiler generates a similar segment of C code with full optimisations switched on.
- Unless you are a confident MIPS assembler programmer, trust the C compiler to optimise large sections of code more efficiently.
- The R3000 has 32 registers. Use this to your advantage with both C and assembler code. For example, define local variables as:

```
register unsigned long counter; /* temporary counter */
```

- Avoid byte or word (2 byte) variable definitions. The cost of memory allocation for dword (4byte) allocations against the time considerations for the extra code generated by the C compiler must be evaluated.
- Avoid redundant code. Remember every extra instruction will affect the execution speed of your program in two ways. Time is spent both executing the instruction and the I cache performance may be badly impacted by inflated code. For example, note the timings between unoptimised and optimised code generated by the C compiler.

Programming the CPU and co-processors

- Avoid transfer between main memory and the frame buffer. Although this is necessary, avoid performing such transfers within time critical sections of code, such as interrutps.
- Avoid dynamic data allocation and preparation. A balance must be made between memory allocation and time spent generating data. Prepare as much data as possible 'off-line'.
- The GPU may be able to push pixels onto the screen very quickly, but the R3000

host will nearly always struggle to keep the rendering pipeline full. The GTE performs many operations very quickly to aid the host, but memory accessing alone can be costly when generating and transferring rendering data to the GPU.

- Avoid blocking the CPU. Use call-backs and interrupts to indicate when certain events have taken place such as vertical synchronization.
- The maximum size of a GPU primitive is 64 bytes. Any primitive larger than this will fail to be decoded correctly by the GPU due to limited buffer size.
- T Cache. TheGPU texture cache is 2KBytes. Using texture map sizes which fit within this cache will increase the performance of texture mapping (including sprite) operations. Allow space for the clut in this cache when using 4 or 8 bit indexed texture mapping.
- Whilst it is desirable to have a different texture map on each polygon in a model, placing one texture across many polygons will both increase texture mapping speed and reduce memory requirements in the frame buffer. However, due to the limited size of texel coordinates within a texture page imposed by the PlayStation hardware, this is not always a simple task.
- Supervise the designers and artists. However nice 256x256 16 bit texture maps are to look at, they really don't help. Models with lots of polygons aren't advisable either.

BRender Performance Issues

The following is a list of BRender programming issues which the application programmer must be aware of affecting the performance of the PlayStation.

- Consult the Technical Reference Manual for a detailed explanation of the BRender API.
- Avoid manipulating the model data during time critical sections of code. To obtain fast execution speeds, BRender pre-prepares as much model data as possible before performing rendering operations. Any changes to material flags will only be reflected after a **BrModelUpdate()**, which is a time consuming process. This includes modifications to vertices, mapping co-ordinates, material and colour map attributes.
- Avoid unnecessary nodes in an actor hierarchy. Actors of type BR_ACTOR_NONE may be useful to enable clear visualisation of an actor hierarchy, but each node in the hierarchy will reduce performance.
- Use custom model call-backs where possible. Extra processing may be performed at a hierarchy node by using the custom model call-back for operations such as sprites within the 3D world.
- Do not use the Sony higher level 3D library functions included in LIBGS.
- LIBGTE and LIBGPU functions and macros may be used freely in conjunction with BRender. Use the GTE macros wherever possible.
- Use a double buffered memory scheme if possible. An application will execute faster at the expense of memory if double buffering is used.
- Use an efficient memory allocation/deallocation scheme. All BRender dynamic memory allocation can be redirected. The memory allocators supplied by Sony may be both faulty and slow.
- Do not use polygon sub-division unless absolutely necessary. Recursive sub-

division is a costly process.

- Try not to use lights in a scene. Pre-lit models and textures can be visually more effective than lit models. Only models (polygons) moving relative to the light source need to be lit. Generally, a large number of polygons in a scene do not move relative to the light source and so can be prelit.
- Reduce the number of materials used on each model. This will not only provide a speed increase, but reduce memory requirements.
- Reduce the number of models in a scene. A small number of models with a large number of polygons is rendered faster than a large number of models with a small number of polygons.
- Models may not be referenced more than once by actors in a hierarchy. Although multiple instancing of models is possible with BRender on other platforms, it is not currently possible on the PlayStation.
- Models must be generated from triangle polygon primitives. Quadrilateral primitives are not currently supported.
- Order tables are automatically cleared when first encountered upon hierarchy traversal. There is no need to explicitly clear any order table used in a hierarchy.
- The primary order table is used for the insertion of scenery primitives into a scene.
- Use custom model call-backs for level of detail control over models in a scene. Sprites are a very useful alternative for models which are too small to be distinguished in a scene.
- Use `BR_ACTOR_BOUNDS` and `BR_ACTOR_BOUNDS_CORRECT` actor type nodes in a hierarchy to perform object culling more effectively. Discarding entire models is more effective than individual polygons.
- Use `BR_TRANSFORM_MATRIX3T` actor transform types wherever possible in a hierarchy as operations on the **`br_matrix3t`** matrix type are performed using dedicated PlayStation hardware.
- The error handler and file handler supplied with the example code is just a demonstration of how to write such systems. Feel free to re-write both systems to suit your needs more effectively.
- Examine all of the electronic documentation files included with the distribution.

BRender Programming Tips

To insert custom user rendering primitives into an order table shared with BRender primitves, it may be necessay to reset the current texture page and texture window for the desired primitive.

```
struct {
   brps_draw_mode mode;
   brps_prim_sprite primitive;
} user_sprite; /* size of this struct must be less than 16 dwords */
brps_rectangle twin;
...
BrPSRectangleSet(&twin, 0, 0, 255, 255);
BrPSPrimDrawModeSet(&user_sprite.mode, 1, 0, tpage, &twin);
BrPSPrimSpriteSet(&user_sprite.primitive);
BrPSPrimMerge(&user_sprite.mode, &user_sprite.primitive);
```

Example Minimal Program

The following code is an example minimal program describing the minimum functionality required by an application using BRender on the PlayStation. Many useful features for the applications programmer are not present in this example for clarity reasons. See the example tutorial code for more detail.

```
/* Include header files needed for the Sony Libraries */
#include <sys/types.h>
#include <r3000.h>
#include <asm.h>
#include <kernel.h>
#include <libgte.h>
#include <libgpu.h>
#include <libetc.h>

/* Include header files needed for BRender */
#include “brender.h”
#include “psio.h”

void main(void)
{
  brps_draw_environment draw[2];
  brps_display_environment display[2];
  br_pixelmap *frame_buffer, screen[2];
  br_actor *root, *camera, *light, *actor;
  br_uint_8 db_index;

  br_uint_16 counter;

  /* Initialise BRender */
  BrBegin();

  /* Initialise PlayStation rendering and display hardware */
  BrPSGeomInit();
  BrPSGraphReset(0);
  BrPSGraphDebugSet(0);

  /* define default double buffered drawing and display environments */
  BrPSDrawEnvironmentDefine(draw, 0, 0, 320, 240);
  BrPSDrawEnvironmentDefine(draw + 1, 320, 0, 320, 240);
  BrPSDisplayEnvironmentDefine(display, 320, 0, 320, 240);
  BrPSDisplayEnvironmentDefine(display + 1, 0, 0, 320, 240);

  /* set automatic screen clear on double buffer swap */
  draw[0].clear_screen = draw[1].clear_screen = 1;
  BrPSDrawEnvironmentRGBSet(draw, 80, 80, 180);
  BrPSDrawEnvironmentRGBSet(draw + 1, 80, 80, 180);

  /* Initialise BRender support library */
  frame_buffer = PSGfxBegin(draw, display, NULL, 0,
     PS_GFX_DOUBLE_BUFFER);

/* Obtain pixelmap pointers to double buffered screens */
screen[0] = BrPixelmapMatch(frame_buffer,
     BR_PMMATCH_OFFSCREEN);
screen[1] = BrPixelmapMatch(frame_buffer,
     BR_PMMATCH_OFFSCREEN);

BrPSDisplayMaskSet(1);

/* Initialise BRender z sort renderer */
BrZsBegin(NULL, 0);

/* Define hierarchy */
root = BrActorAllocate(BR_ACTOR_NONE, NULL);
/* Transform type BR_TRANSFORM_MATRIX3T is used as operations are performed with
hardware */
root->t.type = BR_TRANSFORM_MATRIX3T;
BrMatrix3tIdentity(&root->t.t.mat3t.mat);

/* define perspective camera */
camera = BrActorAdd(root,
     BrActorAllocate(BR_ACTOR_CAMERA, NULL));
((br_camera *)camera->type_data)->type =
     BR_CAMERA_PERSPECTIVE);
((br_camera *)camera->type_data)->aspect =
     BR_SCALAR(1.21);
((br_camera *)camera->type_data)->hither_z =
     BR_SCALAR(0.05);
((br_camera *)camera->type_data)->yon_z = BR_SCALAR(100);
camera->t.type = BR_TRANSFORM_MATRIX3T;
BrMatrix3tTranslate(&camera->t.t.mat3t.mat, 0, 0,
     BR_SCALAR(6));

light = BrActorAdd(root, BrActorAllocate(BR_ACTOR_LIGHT,
     NULL));
light->t.type = BR_TRANSFORM_MATRIX3T;
/* use a white light */
((br_light *)light->type_data)->colour =
     BR_COLOUR_RGB(255, 255, 255);
BrMatrix3tIdentity(&light->t.t.mat3t.mat);

BrLightEnable(light);

/* set an order table at hierarchy root */
BrZsActorOrderTableSet(root, BrZsOrderTableAllcoate(127,
     0, 0, BR_SORT_FIRST_VERTEX, BR_ORDER_TABLE_NEW_BOUNDS
     | BR_ORDER_TABLE_SORT_NEAR));

/* allocate a model actor, using default model and inherited order table */
actor = BrActorAdd(root, BrActorAllocate(BR_ACTOR_MODEL,
                NULL));
        actor->t.type = BR_TRANSFORM_MATRIX3T;
        BrMatrix3tTranslate(&actor->t.t.mat3t.mat, 0, 0, 0);

        /* main loop */
        for(counter = 0; counter < 360; counter += 2) {
                db_index = PSGfxDoubleBufferIndexRead();

                /* Update actor in hierarchy */
                BrMatrix3tRotateY(&actor->t.t.mat3t.mat,
                        BR_ANGLE_DEG(counter));

                /* Render scene */
                BrZsSceneRender(root, camera, screen[db_index], 0);

                /* Wait for drawing to finish */
                DrawSync(0);
                /* Wait until next frame flyback */
                VSync(0);

                /* Swap double buffered environments */
                BrPixelmapDoubleBuffer(frame_buffer, screen[db_index]);
        }

        /* Terminate z sort renderer */
        BrZsEnd();
        /* Terminate support library */
        PSGfxEnd();
        /* Terminate BRender libraries */
        BrEnd();
}
```

```
Example Minimal Program
```