libgxm Reference

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

Introductory Notes

Note on Hardware Versions and Terminology

Note on Hardware Versions and Terminology

Part of the design of libgxm is to ensure that the transition between different hardware revisions is smooth. As such, while libgxm provides direct access to many GPU data structures, most access is through opaque sets of control words because the structures differ in implementation between hardware revisions. These are termed ‘opaque structures’ in this document.

Rendering API

Data Types

SceGxmAttributeFormat

The vertex attribute formats.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmAttributeFormat {
    SCE_GXM_ATTRIBUTE_FORMAT_U8,
    SCE_GXM_ATTRIBUTE_FORMAT_S8,
    SCE_GXM_ATTRIBUTE_FORMAT_U16,
    SCE_GXM_ATTRIBUTE_FORMAT_S16,
    SCE_GXM_ATTRIBUTE_FORMAT_U8N,
    SCE_GXM_ATTRIBUTE_FORMAT_S8N,
    SCE_GXM_ATTRIBUTE_FORMAT_U16N,
    SCE_GXM_ATTRIBUTE_FORMAT_S16N,
    SCE_GXM_ATTRIBUTE_FORMAT_F16,
    SCE_GXM_ATTRIBUTE_FORMAT_F32,
    SCE_GXM_ATTRIBUTE_FORMAT_UNTYPED
} SceGxmAttributeFormat;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_ATTRIBUTE_FORMAT_U8	N/A	8-bit unsigned integer
SCE_GXM_ATTRIBUTE_FORMAT_S8	N/A	8-bit signed integer
SCE_GXM_ATTRIBUTE_FORMAT_U16	N/A	16-bit unsigned integer
SCE_GXM_ATTRIBUTE_FORMAT_S16	N/A	16-bit signed integer
SCE_GXM_ATTRIBUTE_FORMAT_U8N	N/A	8-bit unsigned integer normalized to [0,1] range
SCE_GXM_ATTRIBUTE_FORMAT_S8N	N/A	8-bit signed integer normalized to [-1,1] range
SCE_GXM_ATTRIBUTE_FORMAT_U16N	N/A	16-bit unsigned integer normalized to [0,1] range
SCE_GXM_ATTRIBUTE_FORMAT_S16N	N/A	16-bit signed integer normalized to [-1,1] range
SCE_GXM_ATTRIBUTE_FORMAT_F16	N/A	16-bit half precision floating point
SCE_GXM_ATTRIBUTE_FORMAT_F32	N/A	32-bit single precision floating point
SCE_GXM_ATTRIBUTE_FORMAT_UNTYPED	N/A	32-bit untyped data for use with offline vertex unpack

Description

The vertex attribute formats.

SceGxmColorBaseFormat

The base formats for color surfaces.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmColorBaseFormat {
    SCE_GXM_COLOR_BASE_FORMAT_U8U8U8U8 = 0x00000000U,
    SCE_GXM_COLOR_BASE_FORMAT_U8U8U8 = 0x10000000U,
    SCE_GXM_COLOR_BASE_FORMAT_U5U6U5 = 0x30000000U,
    SCE_GXM_COLOR_BASE_FORMAT_U1U5U5U5 = 0x40000000U,
    SCE_GXM_COLOR_BASE_FORMAT_U4U4U4U4 = 0x50000000U,
    SCE_GXM_COLOR_BASE_FORMAT_U8U3U3U2 = 0x60000000U,
    SCE_GXM_COLOR_BASE_FORMAT_F16 = 0xf0000000U,
    SCE_GXM_COLOR_BASE_FORMAT_F16F16 = 0x00800000U,
    SCE_GXM_COLOR_BASE_FORMAT_F32 = 0x10800000U,
    SCE_GXM_COLOR_BASE_FORMAT_S16 = 0x20800000U,
    SCE_GXM_COLOR_BASE_FORMAT_S16S16 = 0x30800000U,
    SCE_GXM_COLOR_BASE_FORMAT_U16 = 0x40800000U,
    SCE_GXM_COLOR_BASE_FORMAT_U16U16 = 0x50800000U,
    SCE_GXM_COLOR_BASE_FORMAT_U2U10U10U10 = 0x60800000U,
    SCE_GXM_COLOR_BASE_FORMAT_U8 = 0x80800000U,
    SCE_GXM_COLOR_BASE_FORMAT_S8 = 0x90800000U,
    SCE_GXM_COLOR_BASE_FORMAT_S5S5U6 = 0xa0800000U,
    SCE_GXM_COLOR_BASE_FORMAT_U8U8 = 0xb0800000U,
    SCE_GXM_COLOR_BASE_FORMAT_S8S8 = 0xc0800000U,
    SCE_GXM_COLOR_BASE_FORMAT_S8S8S8S8 = 0xe0800000U,
    SCE_GXM_COLOR_BASE_FORMAT_F16F16F16F16 = 0x01000000U,
    SCE_GXM_COLOR_BASE_FORMAT_F32F32 = 0x11000000U,
    SCE_GXM_COLOR_BASE_FORMAT_F11F11F10 = 0x21000000U,
    SCE_GXM_COLOR_BASE_FORMAT_SE5M9M9M9 = 0x31000000U,
    SCE_GXM_COLOR_BASE_FORMAT_U2F10F10F10 = 0x41000000U
} SceGxmColorBaseFormat;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_COLOR_BASE_FORMAT_U8U8U8U8`	`0x00000000U`	32-bit format, 4x 8-bit unsigned integer
`SCE_GXM_COLOR_BASE_FORMAT_U8U8U8`	`0x10000000U`	24-bit packed format, 3x 8-bit unsigned integer
`SCE_GXM_COLOR_BASE_FORMAT_U5U6U5`	`0x30000000U`	16-bit format, 5-bit unsigned, 6-bit unsigned and 5-bit unsigned integer
`SCE_GXM_COLOR_BASE_FORMAT_U1U5U5U5`	`0x40000000U`	16-bit format, 1-bit unsigned and 3x 5-bit unsigned integer
`SCE_GXM_COLOR_BASE_FORMAT_U4U4U4U4`	`0x50000000U`	16-bit format, 4x 4-bit unsigned integer
`SCE_GXM_COLOR_BASE_FORMAT_U8U3U3U2`	`0x60000000U`	16-bit format, 8-bit unsigned, 3-bit unsigned, 3-bit unsigned and 2-bit unsigned integer
`SCE_GXM_COLOR_BASE_FORMAT_F16`	`0xf0000000U`	16-bit format, 16-bit s1e5m10 floating point
`SCE_GXM_COLOR_BASE_FORMAT_F16F16`	`0x00800000U`	32-bit format, 2x 16-bit s1e5m10 floating point

Macro	Value	Description
SCE_GXM_COLOR_BASE_FORMAT_F32	0x10800000U	32-bit format, 32-bit floating point
SCE_GXM_COLOR_BASE_FORMAT_S16	0x20800000U	16-bit format, 16-bit signed integer
SCE_GXM_COLOR_BASE_FORMAT_S16S16	0x30800000U	32-bit format, 2x 16-bit signed integer
SCE_GXM_COLOR_BASE_FORMAT_U16	0x40800000U	16-bit format, 16-bit unsigned integer
SCE_GXM_COLOR_BASE_FORMAT_U16U16	0x50800000U	32-bit format, 2x 16-bit unsigned integer
SCE_GXM_COLOR_BASE_FORMAT_U2U10U10U10	0x60800000U	32-bit format, 2-bit unsigned and 3x 10-bit unsigned integer
SCE_GXM_COLOR_BASE_FORMAT_U8	0x80800000U	8-bit format, 8-bit unsigned integer
SCE_GXM_COLOR_BASE_FORMAT_S8	0x90800000U	8-bit format, 8-bit signed integer
SCE_GXM_COLOR_BASE_FORMAT_S5S5U6	0xa0800000U	16-bit format, 5-bit signed, 5-bit signed and 6-bit unsigned integer
SCE_GXM_COLOR_BASE_FORMAT_U8U8	0xb0800000U	16-bit format, 2x 8-bit unsigned integer
SCE_GXM_COLOR_BASE_FORMAT_S8S8	0xc0800000U	16-bit format, 2x 8-bit signed integer
SCE_GXM_COLOR_BASE_FORMAT_S8S8S8S8	0xe0800000U	32-bit format, 4x 8-bit signed integer
SCE_GXM_COLOR_BASE_FORMAT_F16F16F16F16	0x01000000U	64-bit format, 4x 16-bit s1e5m10 floating point
SCE_GXM_COLOR_BASE_FORMAT_F32F32	0x11000000U	64-bit format, 2x 32-bit floating point
SCE_GXM_COLOR_BASE_FORMAT_F11F11F10	0x21000000U	32-bit format, 2x 11-bit s0e5m6 floating point and 10-bit s0e5m5 floating point
SCE_GXM_COLOR_BASE_FORMAT_SE5M9M9M9	0x31000000U	32-bit format, 5-bit shared exponent and 3x 9-bit floating point mantissa
SCE_GXM_COLOR_BASE_FORMAT_U2F10F10F10	0x41000000U	32-bit format, 2-bit unsigned integer and 3x 10-bit s0e5m5 floating point

Description

The base formats for color surfaces. A color format is made from (bitwise) combining a base format with a compatible swizzle.

SceGxmColorFormat

The color formats.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmColorFormat {
    SCE_GXM_COLOR_FORMAT_U8U8U8U8_ABGR = SCE_GXM_COLOR_BASE_FORMAT_U8U8U8U8 |
    SCE_GXM_COLOR_SWIZZLE4_ABGR,
    SCE_GXM_COLOR_FORMAT_U8U8U8U8_ARGB = SCE_GXM_COLOR_BASE_FORMAT_U8U8U8U8 |
    SCE_GXM_COLOR_SWIZZLE4_ARGB,
    SCE_GXM_COLOR_FORMAT_U8U8U8U8_RGBA = SCE_GXM_COLOR_BASE_FORMAT_U8U8U8U8 |
    SCE_GXM_COLOR_SWIZZLE4_RGBA,
    SCE_GXM_COLOR_FORMAT_U8U8U8U8_BGRA = SCE_GXM_COLOR_BASE_FORMAT_U8U8U8U8 |
    SCE_GXM_COLOR_SWIZZLE4_BGRA,
    SCE_GXM_COLOR_FORMAT_U8U8U8_BGR = SCE_GXM_COLOR_BASE_FORMAT_U8U8U8 |
    SCE_GXM_COLOR_SWIZZLE3_BGR,
    SCE_GXM_COLOR_FORMAT_U8U8U8_RGB = SCE_GXM_COLOR_BASE_FORMAT_U8U8U8 |
    SCE_GXM_COLOR_SWIZZLE3_RGB,
    SCE_GXM_COLOR_FORMAT_U5U6U5_BGR = SCE_GXM_COLOR_BASE_FORMAT_U5U6U5 |
    SCE_GXM_COLOR_SWIZZLE3_BGR,
    SCE_GXM_COLOR_FORMAT_U5U6U5_RGB = SCE_GXM_COLOR_BASE_FORMAT_U5U6U5 |
    SCE_GXM_COLOR_SWIZZLE3_RGB,
    SCE_GXM_COLOR_FORMAT_U1U5U5U5_ABGR = SCE_GXM_COLOR_BASE_FORMAT_U1U5U5U5 |
    SCE_GXM_COLOR_SWIZZLE4_ABGR,
    SCE_GXM_COLOR_FORMAT_U1U5U5U5_ARGB = SCE_GXM_COLOR_BASE_FORMAT_U1U5U5U5 |
    SCE_GXM_COLOR_SWIZZLE4_ARGB,
    SCE_GXM_COLOR_FORMAT_U5U5U5U1_RGBA = SCE_GXM_COLOR_BASE_FORMAT_U1U5U5U5 |
    SCE_GXM_COLOR_SWIZZLE4_RGBA,
    SCE_GXM_COLOR_FORMAT_U5U5U5U1_BGRA = SCE_GXM_COLOR_BASE_FORMAT_U1U5U5U5 |
    SCE_GXM_COLOR_SWIZZLE4_BGRA,
    SCE_GXM_COLOR_FORMAT_U4U4U4U4_ABGR = SCE_GXM_COLOR_BASE_FORMAT_U4U4U4U4 |
    SCE_GXM_COLOR_SWIZZLE4_ABGR,
    SCE_GXM_COLOR_FORMAT_U4U4U4U4_ARGB = SCE_GXM_COLOR_BASE_FORMAT_U4U4U4U4 |
    SCE_GXM_COLOR_SWIZZLE4_ARGB,
    SCE_GXM_COLOR_FORMAT_U4U4U4U4_RGBA = SCE_GXM_COLOR_BASE_FORMAT_U4U4U4U4 |
    SCE_GXM_COLOR_SWIZZLE4_RGBA,
    SCE_GXM_COLOR_FORMAT_U4U4U4U4_BGRA = SCE_GXM_COLOR_BASE_FORMAT_U4U4U4U4 |
    SCE_GXM_COLOR_SWIZZLE4_BGRA,
    SCE_GXM_COLOR_FORMAT_U8U3U3U2_ARGB = SCE_GXM_COLOR_BASE_FORMAT_U8U3U3U2,
    SCE_GXM_COLOR_FORMAT_F16_R = SCE_GXM_COLOR_BASE_FORMAT_F16 |
    SCE_GXM_COLOR_SWIZZLE1_R,
    SCE_GXM_COLOR_FORMAT_F16_G = SCE_GXM_COLOR_BASE_FORMAT_F16 |
    SCE_GXM_COLOR_SWIZZLE1_G,
    SCE_GXM_COLOR_FORMAT_F16F16_GR = SCE_GXM_COLOR_BASE_FORMAT_F16F16 |
    SCE_GXM_COLOR_SWIZZLE2_GR,
    SCE_GXM_COLOR_FORMAT_F16F16_RG = SCE_GXM_COLOR_BASE_FORMAT_F16F16 |
    SCE_GXM_COLOR_SWIZZLE2_RG,
    SCE_GXM_COLOR_FORMAT_F32_R = SCE_GXM_COLOR_BASE_FORMAT_F32 |
    SCE_GXM_COLOR_SWIZZLE1_R,
    SCE_GXM_COLOR_FORMAT_S16_R = SCE_GXM_COLOR_BASE_FORMAT_S16 |
    SCE_GXM_COLOR_SWIZZLE1_R,
    SCE_GXM_COLOR_FORMAT_S16_G = SCE_GXM_COLOR_BASE_FORMAT_S16 |
    SCE_GXM_COLOR_SWIZZLE1_G,
    SCE_GXM_COLOR_FORMAT_S16S16_GR = SCE_GXM_COLOR_BASE_FORMAT_S16S16 |
    SCE_GXM_COLOR_SWIZZLE2_GR,
    SCE_GXM_COLOR_FORMAT_S16S16_RG = SCE_GXM_COLOR_BASE_FORMAT_S16S16 |
    SCE_GXM_COLOR_SWIZZLE2_RG,
    SCE_GXM_COLOR_FORMAT_U16_R = SCE_GXM_COLOR_BASE_FORMAT_U16 |
    SCE_GXM_COLOR_SWIZZLE1_R,
    SCE_GXM_COLOR_FORMAT_U16_G = SCE_GXM_COLOR_BASE_FORMAT_U16 |
    SCE_GXM_COLOR_SWIZZLE1_G,
    SCE_GXM_COLOR_FORMAT_U16U16_GR = SCE_GXM_COLOR_BASE_FORMAT_U16U16 |
    SCE_GXM_COLOR_SWIZZLE2_GR,
    SCE_GXM_COLOR_FORMAT_U16U16_RG = SCE_GXM_COLOR_BASE_FORMAT_U16U16 |
    SCE_GXM_COLOR_SWIZZLE2_RG,
    SCE_GXM_COLOR_FORMAT_U2U10U10U10_ABGR =
    SCE_GXM_COLOR_BASE_FORMAT_U2U10U10U10 | SCE_GXM_COLOR_SWIZZLE4_ABGR,
    SCE_GXM_COLOR_FORMAT_U2U10U10U10_ARGB =
    SCE_GXM_COLOR_BASE_FORMAT_U2U10U10U10 | SCE_GXM_COLOR_SWIZZLE4_ARGB,
    SCE_GXM_COLOR_FORMAT_U10U10U10U2_RGBA =
    SCE_GXM_COLOR_BASE_FORMAT_U2U10U10U10 | SCE_GXM_COLOR_SWIZZLE4_RGBA,
    SCE_GXM_COLOR_FORMAT_U10U10U10U2_BGRA =
    SCE_GXM_COLOR_BASE_FORMAT_U2U10U10U10 | SCE_GXM_COLOR_SWIZZLE4_BGRA,
    SCE_GXM_COLOR_FORMAT_U8_R = SCE_GXM_COLOR_BASE_FORMAT_U8 |
    SCE_GXM_COLOR_SWIZZLE1_R,
    SCE_GXM_COLOR_FORMAT_U8_A = SCE_GXM_COLOR_BASE_FORMAT_U8 |
    SCE_GXM_COLOR_SWIZZLE1_A,
    SCE_GXM_COLOR_FORMAT_S8_R = SCE_GXM_COLOR_BASE_FORMAT_S8 |
    SCE_GXM_COLOR_SWIZZLE1_R,
    SCE_GXM_COLOR_FORMAT_S8_A = SCE_GXM_COLOR_BASE_FORMAT_S8 |
    SCE_GXM_COLOR_SWIZZLE1_A,
    SCE_GXM_COLOR_FORMAT_U6S5S5_BGR = SCE_GXM_COLOR_BASE_FORMAT_S5S5U6 |
    SCE_GXM_COLOR_SWIZZLE3_BGR,
    SCE_GXM_COLOR_FORMAT_S5S5U6_RGB = SCE_GXM_COLOR_BASE_FORMAT_S5S5U6 |
    SCE_GXM_COLOR_SWIZZLE3_RGB,
    SCE_GXM_COLOR_FORMAT_U8U8_GR = SCE_GXM_COLOR_BASE_FORMAT_U8U8 |
    SCE_GXM_COLOR_SWIZZLE2_GR,
    SCE_GXM_COLOR_FORMAT_U8U8_RG = SCE_GXM_COLOR_BASE_FORMAT_U8U8 |
    SCE_GXM_COLOR_SWIZZLE2_RG,
    SCE_GXM_COLOR_FORMAT_U8U8_RA = SCE_GXM_COLOR_BASE_FORMAT_U8U8 |
    SCE_GXM_COLOR_SWIZZLE2_RA,
    SCE_GXM_COLOR_FORMAT_U8U8_AR = SCE_GXM_COLOR_BASE_FORMAT_U8U8 |
    SCE_GXM_COLOR_SWIZZLE2_AR,
    SCE_GXM_COLOR_FORMAT_S8S8_GR = SCE_GXM_COLOR_BASE_FORMAT_S8S8 |
    SCE_GXM_COLOR_SWIZZLE2_GR,
    SCE_GXM_COLOR_FORMAT_S8S8_RG = SCE_GXM_COLOR_BASE_FORMAT_S8S8 |
    SCE_GXM_COLOR_SWIZZLE2_RG,
    SCE_GXM_COLOR_FORMAT_S8S8_RA = SCE_GXM_COLOR_BASE_FORMAT_S8S8 |
    SCE_GXM_COLOR_SWIZZLE2_RA,
    SCE_GXM_COLOR_FORMAT_S8S8_AR = SCE_GXM_COLOR_BASE_FORMAT_S8S8 |
    SCE_GXM_COLOR_SWIZZLE2_AR,
    SCE_GXM_COLOR_FORMAT_S8S8S8S8_ABGR = SCE_GXM_COLOR_BASE_FORMAT_S8S8S8S8 |
    SCE_GXM_COLOR_SWIZZLE4_ABGR,
    SCE_GXM_COLOR_FORMAT_S8S8S8S8_ARGB = SCE_GXM_COLOR_BASE_FORMAT_S8S8S8S8 |
    SCE_GXM_COLOR_SWIZZLE4_ARGB,
    SCE_GXM_COLOR_FORMAT_S8S8S8S8_RGBA = SCE_GXM_COLOR_BASE_FORMAT_S8S8S8S8 |
    SCE_GXM_COLOR_SWIZZLE4_RGBA,
    SCE_GXM_COLOR_FORMAT_S8S8S8S8_BGRA = SCE_GXM_COLOR_BASE_FORMAT_S8S8S8S8 |
    SCE_GXM_COLOR_SWIZZLE4_BGRA,
    SCE_GXM_COLOR_FORMAT_F16F16F16F16_ABGR =
    SCE_GXM_COLOR_BASE_FORMAT_F16F16F16F16 | SCE_GXM_COLOR_SWIZZLE4_ABGR,
    SCE_GXM_COLOR_FORMAT_F16F16F16F16_ARGB =
    SCE_GXM_COLOR_BASE_FORMAT_F16F16F16F16 | SCE_GXM_COLOR_SWIZZLE4_ARGB,
    SCE_GXM_COLOR_FORMAT_F16F16F16F16_RGBA =
    SCE_GXM_COLOR_BASE_FORMAT_F16F16F16F16 | SCE_GXM_COLOR_SWIZZLE4_RGBA,
    SCE_GXM_COLOR_FORMAT_F16F16F16F16_BGRA =
    SCE_GXM_COLOR_BASE_FORMAT_F16F16F16F16 | SCE_GXM_COLOR_SWIZZLE4_BGRA,
    SCE_GXM_COLOR_FORMAT_F32F32_GR = SCE_GXM_COLOR_BASE_FORMAT_F32F32 |
    SCE_GXM_COLOR_SWIZZLE2_GR,
    SCE_GXM_COLOR_FORMAT_F32F32_RG = SCE_GXM_COLOR_BASE_FORMAT_F32F32 |
    SCE_GXM_COLOR_SWIZZLE2_RG,
    SCE_GXM_COLOR_FORMAT_F10F11F11_BGR = SCE_GXM_COLOR_BASE_FORMAT_F11F11F10 |
    SCE_GXM_COLOR_SWIZZLE3_BGR,
    SCE_GXM_COLOR_FORMAT_F11F11F10_RGB = SCE_GXM_COLOR_BASE_FORMAT_F11F11F10 |
    SCE_GXM_COLOR_SWIZZLE3_RGB,
    SCE_GXM_COLOR_FORMAT_SE5M9M9M9_BGR = SCE_GXM_COLOR_BASE_FORMAT_SE5M9M9M9 |
    SCE_GXM_COLOR_SWIZZLE3_BGR,
    SCE_GXM_COLOR_FORMAT_SE5M9M9M9_RGB = SCE_GXM_COLOR_BASE_FORMAT_SE5M9M9M9 |
    SCE_GXM_COLOR_SWIZZLE3_RGB,
    SCE_GXM_COLOR_FORMAT_U2F10F10F10_ABGR =
    SCE_GXM_COLOR_BASE_FORMAT_U2F10F10F10 | SCE_GXM_COLOR_SWIZZLE4_ABGR,
    SCE_GXM_COLOR_FORMAT_U2F10F10F10_ARGB =
    SCE_GXM_COLOR_BASE_FORMAT_U2F10F10F10 | SCE_GXM_COLOR_SWIZZLE4_ARGB,
    SCE_GXM_COLOR_FORMAT_F10F10F10U2_RGBA =
    SCE_GXM_COLOR_BASE_FORMAT_U2F10F10F10 | SCE_GXM_COLOR_SWIZZLE4_RGBA,
    SCE_GXM_COLOR_FORMAT_F10F10F10U2_BGRA =
    SCE_GXM_COLOR_BASE_FORMAT_U2F10F10F10 | SCE_GXM_COLOR_SWIZZLE4_BGRA,
    SCE_GXM_COLOR_FORMAT_A8B8G8R8 = SCE_GXM_COLOR_FORMAT_U8U8U8U8_ABGR,
    SCE_GXM_COLOR_FORMAT_A8R8G8B8 = SCE_GXM_COLOR_FORMAT_U8U8U8U8_ARGB,
    SCE_GXM_COLOR_FORMAT_R5G6B5 = SCE_GXM_COLOR_FORMAT_U5U6U5_RGB,
    SCE_GXM_COLOR_FORMAT_A1R5G5B5 = SCE_GXM_COLOR_FORMAT_U1U5U5U5_ARGB,
    SCE_GXM_COLOR_FORMAT_A4R4G4B4 = SCE_GXM_COLOR_FORMAT_U4U4U4U4_ARGB,
    SCE_GXM_COLOR_FORMAT_A8 = SCE_GXM_COLOR_FORMAT_U8_A
} SceGxmColorFormat;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_COLOR_FORMAT_U8U8U8U8_ABGR`	`SCE_GXM_COLOR_BASE_FORMAT_U8U8U8U8 | SCE_GXM_COLOR_SWIZZLE4_ABGR`	Pixels are written to memory in U8U8U8U8 format in ABGR order.
`SCE_GXM_COLOR_FORMAT_U8U8U8U8_ARGB`	`SCE_GXM_COLOR_BASE_FORMAT_U8U8U8U8 | SCE_GXM_COLOR_SWIZZLE4_ARGB`	Pixels are written to memory in U8U8U8U8 format in ARGB order.
`SCE_GXM_COLOR_FORMAT_U8U8U8U8_RGBA`	`SCE_GXM_COLOR_BASE_FORMAT_U8U8U8U8 | SCE_GXM_COLOR_SWIZZLE4_RGBA`	Pixels are written to memory in U8U8U8U8 format in RGBA order.
`SCE_GXM_COLOR_FORMAT_U8U8U8U8_BGRA`	`SCE_GXM_COLOR_BASE_FORMAT_U8U8U8U8 | SCE_GXM_COLOR_SWIZZLE4_BGRA`	Pixels are written to memory in U8U8U8U8 format in BGRA order.
`SCE_GXM_COLOR_FORMAT_U8U8U8_BGR`	`SCE_GXM_COLOR_BASE_FORMAT_U8U8U8 | SCE_GXM_COLOR_SWIZZLE3_BGR`	Pixels are written to memory in packed 24-bit U8U8U8 format in BGR order, A is discarded.
`SCE_GXM_COLOR_FORMAT_U8U8U8_RGB`	`SCE_GXM_COLOR_BASE_FORMAT_U8U8U8 | SCE_GXM_COLOR_SWIZZLE3_RGB`	Pixels are written to memory in packed 24-bit U8U8U8 format in RGB order, A is discarded.

Macro	Value	Description
SCE_GXM_COLOR_ FORMAT_U5U6U5_BGR	SCE_GXM_COLOR_ BASE_FORMAT_ U5U6U5 | SCE_GXM_COLOR_ SWIZZLE3_BGR	Pixels are written to memory in U5U6U5 format in BGR order, A is discarded.
SCE_GXM_COLOR_ FORMAT_U5U6U5_RGB	SCE_GXM_COLOR_ BASE_FORMAT_ U5U6U5 | SCE_GXM_COLOR_ SWIZZLE3_RGB	Pixels are written to memory in U5U6U5 format in RGB order, A is discarded.
SCE_GXM_COLOR_ FORMAT_U1U5U5U5_ABGR	SCE_GXM_COLOR_ BASE_FORMAT_ U1U5U5U5 | SCE_GXM_COLOR_ SWIZZLE4_ABGR	Pixels are written to memory in U1U5U5U5 format in ABGR order.
SCE_GXM_COLOR_ FORMAT_U1U5U5U5_ARGB	SCE_GXM_COLOR_ BASE_FORMAT_ U1U5U5U5 | SCE_GXM_COLOR_ SWIZZLE4_ARGB	Pixels are written to memory in U1U5U5U5 format in ARGB order.
SCE_GXM_COLOR_ FORMAT_U5U5U5U1_RGBA	SCE_GXM_COLOR_ BASE_FORMAT_ U1U5U5U5 | SCE_GXM_COLOR_ SWIZZLE4_RGBA	Pixels are written to memory in U5U5U5U1 format in RGBA order.
SCE_GXM_COLOR_ FORMAT_U5U5U5U1_BGRA	SCE_GXM_COLOR_ BASE_FORMAT_ U1U5U5U5 | SCE_GXM_COLOR_ SWIZZLE4_BGRA	Pixels are written to memory in U5U5U5U1 format in BGRA order.
SCE_GXM_COLOR_ FORMAT_U4U4U4U4_ABGR	SCE_GXM_COLOR_ BASE_FORMAT_ U4U4U4U4 | SCE_GXM_COLOR_ SWIZZLE4_ABGR	Pixels are written to memory in U4U4U4U4 format in ABGR order.
SCE_GXM_COLOR_ FORMAT_U4U4U4U4_ARGB	SCE_GXM_COLOR_ BASE_FORMAT_ U4U4U4U4 | SCE_GXM_COLOR_ SWIZZLE4_ARGB	Pixels are written to memory in U4U4U4U4 format in ARGB order.
SCE_GXM_COLOR_ FORMAT_U4U4U4U4_RGBA	SCE_GXM_COLOR_ BASE_FORMAT_ U4U4U4U4 | SCE_GXM_COLOR_ SWIZZLE4_RGBA	Pixels are written to memory in U4U4U4U4 format in RGBA order.
SCE_GXM_COLOR_ FORMAT_U4U4U4U4_BGRA	SCE_GXM_COLOR_ BASE_FORMAT_ U4U4U4U4 | SCE_GXM_COLOR_ SWIZZLE4_BGRA	Pixels are written to memory in U4U4U4U4 format in BGRA order.
SCE_GXM_COLOR_ FORMAT_U8U3U3U2_ARGB	SCE_GXM_COLOR_ BASE_FORMAT_ U8U3U3U2	Pixels are written to memory in U8U3U3U2 format in ARGB order.
SCE_GXM_COLOR_ FORMAT_F16_R	SCE_GXM_COLOR_ BASE_FORMAT_ F16 | SCE_GXM_COLOR_ SWIZZLE1_R	Pixels are written to memory in F16 format using the R component only, G is discarded.

Macro	Value	Description
SCE_GXM_COLOR_ FORMAT_F16_G	SCE_GXM_COLOR_ BASE_FORMAT_ F16 | SCE_GXM_COLOR_ SWIZZLE1_G	Pixels are written to memory in F16 format using the G component only, R is discarded.
SCE_GXM_COLOR_ FORMAT_F16F16_GR	SCE_GXM_COLOR_ BASE_FORMAT_ F16F16 | SCE_GXM_COLOR_ SWIZZLE2_GR	Pixels are written to memory in F16F16 format in GR order.
SCE_GXM_COLOR_ FORMAT_F16F16_RG	SCE_GXM_COLOR_ BASE_FORMAT_ F16F16 | SCE_GXM_COLOR_ SWIZZLE2_RG	Pixels are written to memory in F16F16 format in RG order.
SCE_GXM_COLOR_ FORMAT_F32_R	SCE_GXM_COLOR_ BASE_FORMAT_ F32 | SCE_GXM_COLOR_ SWIZZLE1_R	Pixels are written to memory in F32 format using the R component only.
SCE_GXM_COLOR_ FORMAT_S16_R	SCE_GXM_COLOR_ BASE_FORMAT_ S16 | SCE_GXM_COLOR_ SWIZZLE1_R	Pixels are written to memory in S16 format using the R component only, G is discarded.
SCE_GXM_COLOR_ FORMAT_S16_G	SCE_GXM_COLOR_ BASE_FORMAT_ S16 | SCE_GXM_COLOR_ SWIZZLE1_G	Pixels are written to memory in S16 format using the G component only, R is discarded.
SCE_GXM_COLOR_ FORMAT_S16S16_GR	SCE_GXM_COLOR_ BASE_FORMAT_ S16S16 | SCE_GXM_COLOR_ SWIZZLE2_GR	Pixels are written to memory in S16S16 format in GR order.
SCE_GXM_COLOR_ FORMAT_S16S16_RG	SCE_GXM_COLOR_ BASE_FORMAT_ S16S16 | SCE_GXM_COLOR_ SWIZZLE2_RG	Pixels are written to memory in S16S16 format in RG order.
SCE_GXM_COLOR_ FORMAT_U16_R	SCE_GXM_COLOR_ BASE_FORMAT_ U16 | SCE_GXM_COLOR_ SWIZZLE1_R	Pixels are written to memory in U16 format using the R component only, G is discarded.
SCE_GXM_COLOR_ FORMAT_U16_G	SCE_GXM_COLOR_ BASE_FORMAT_ U16 | SCE_GXM_COLOR_ SWIZZLE1_G	Pixels are written to memory in U16 format using the G component only, R is discarded.
SCE_GXM_COLOR_ FORMAT_U16U16_GR	SCE_GXM_COLOR_ BASE_FORMAT_ U16U16 | SCE_GXM_COLOR_ SWIZZLE2_GR	Pixels are written to memory in U16U16 format in GR order.
SCE_GXM_COLOR_ FORMAT_U16U16_RG	SCE_GXM_COLOR_ BASE_FORMAT_ U16U16 | SCE_GXM_COLOR_ SWIZZLE2_RG	Pixels are written to memory in U16U16 format in RG order.
SCE_GXM_COLOR_ FORMAT_U2U10U10U10_ABGR	SCE_GXM_COLOR_ BASE_FORMAT_ U2U10U10U10 | SCE_GXM_COLOR_ SWIZZLE4_ABGR	Pixels are written to memory in U2U10U10U10 format in ABGR order.

Macro	Value	Description
SCE_GXM_COLOR_ FORMAT_U2U10U10U10_ARGB	SCE_GXM_COLOR_ BASE_FORMAT_ U2U10U10U10 | SCE_GXM_COLOR_ SWIZZLE4_ARGB	Pixels are written to memory in U2U10U10U10 format in ARGB order.
SCE_GXM_COLOR_ FORMAT_U10U10U10U2_RGBA	SCE_GXM_COLOR_ BASE_FORMAT_ U2U10U10U10 | SCE_GXM_COLOR_ SWIZZLE4_RGBA	Pixels are written to memory in U10U10U10U2 format in RGBA order.
SCE_GXM_COLOR_ FORMAT_U10U10U10U2_BGRA	SCE_GXM_COLOR_ BASE_FORMAT_ U2U10U10U10 | SCE_GXM_COLOR_ SWIZZLE4_BGRA	Pixels are written to memory in U10U10U10U2 format in BGRA order.
SCE_GXM_COLOR_ FORMAT_U8_R	SCE_GXM_COLOR_ BASE_FORMAT_ U8 | SCE_GXM_COLOR_ SWIZZLE1_R	Pixels are written to memory in U8 format using the R component only, A B and G are discarded.
SCE_GXM_COLOR_ FORMAT_U8_A	SCE_GXM_COLOR_ BASE_FORMAT_ U8 | SCE_GXM_COLOR_ SWIZZLE1_A	Pixels are written to memory in U8 format using the A component only, B G and R are discarded.
SCE_GXM_COLOR_ FORMAT_S8_R	SCE_GXM_COLOR_ BASE_FORMAT_ S8 | SCE_GXM_COLOR_ SWIZZLE1_R	Pixels are written to memory in S8 format using the R component only, A B and G are discarded.
SCE_GXM_COLOR_ FORMAT_S8_A	SCE_GXM_COLOR_ BASE_FORMAT_ S8 | SCE_GXM_COLOR_ SWIZZLE1_A	Pixels are written to memory in S8 format using the A component only, B G and R are discarded.
SCE_GXM_COLOR_ FORMAT_U6S5S5_BGR	SCE_GXM_COLOR_ BASE_FORMAT_ S5S5U6 | SCE_GXM_COLOR_ SWIZZLE3_BGR	Pixels are written to memory in U6S5S5 format in BGR order, A is discarded.
SCE_GXM_COLOR_ FORMAT_S5S5U6_RGB	SCE_GXM_COLOR_ BASE_FORMAT_ S5S5U6 | SCE_GXM_COLOR_ SWIZZLE3_RGB	Pixels are written to memory in S5S5U6 format in RGB order, A is discarded.
SCE_GXM_COLOR_ FORMAT_U8U8_GR	SCE_GXM_COLOR_ BASE_FORMAT_ U8U8 | SCE_GXM_COLOR_ SWIZZLE2_GR	Pixels are written to memory in U8U8 format in GR order, A and B are discarded.
SCE_GXM_COLOR_ FORMAT_U8U8_RG	SCE_GXM_COLOR_ BASE_FORMAT_ U8U8 | SCE_GXM_COLOR_ SWIZZLE2_RG	Pixels are written to memory in U8U8 format in RG order, A and B are discarded.
SCE_GXM_COLOR_ FORMAT_U8U8_RA	SCE_GXM_COLOR_ BASE_FORMAT_ U8U8 | SCE_GXM_COLOR_ SWIZZLE2_RA	Pixels are written to memory in U8U8 format in RA order, B and G are discarded.
SCE_GXM_COLOR_ FORMAT_U8U8_AR	SCE_GXM_COLOR_ BASE_FORMAT_ U8U8 | SCE_GXM_COLOR_ SWIZZLE2_AR	Pixels are written to memory in U8U8 format in AR order, B and G are discarded.

Macro	Value	Description
SCE_GXM_COLOR_ FORMAT_S8S8_GR	SCE_GXM_COLOR_ BASE_FORMAT_ S8S8 | SCE_GXM_COLOR_ SWIZZLE2_GR	Pixels are written to memory in S8S8 format in GR order, A and B are discarded.
SCE_GXM_COLOR_ FORMAT_S8S8_RG	SCE_GXM_COLOR_ BASE_FORMAT_ S8S8 | SCE_GXM_COLOR_ SWIZZLE2_RG	Pixels are written to memory in S8S8 format in RG order, A and B are discarded.
SCE_GXM_COLOR_ FORMAT_S8S8_RA	SCE_GXM_COLOR_ BASE_FORMAT_ S8S8 | SCE_GXM_COLOR_ SWIZZLE2_RA	Pixels are written to memory in S8S8 format in RA order, B and G are discarded.
SCE_GXM_COLOR_ FORMAT_S8S8_AR	SCE_GXM_COLOR_ BASE_FORMAT_ S8S8 | SCE_GXM_COLOR_ SWIZZLE2_AR	Pixels are written to memory in S8S8 format in AR order, B and G are discarded.
SCE_GXM_COLOR_ FORMAT_S8S8S8S8_ABGR	SCE_GXM_COLOR_ BASE_FORMAT_ S8S8S8S8 | SCE_GXM_COLOR_ SWIZZLE4_ABGR	Pixels are written to memory in S8S8S8S8 format in ABGR order.
SCE_GXM_COLOR_ FORMAT_S8S8S8S8_ARGB	SCE_GXM_COLOR_ BASE_FORMAT_ S8S8S8S8 | SCE_GXM_COLOR_ SWIZZLE4_ARGB	Pixels are written to memory in S8S8S8S8 format in ARGB order.
SCE_GXM_COLOR_ FORMAT_S8S8S8S8_RGBA	SCE_GXM_COLOR_ BASE_FORMAT_ S8S8S8S8 | SCE_GXM_COLOR_ SWIZZLE4_RGBA	Pixels are written to memory in S8S8S8S8 format in RGBA order.
SCE_GXM_COLOR_ FORMAT_S8S8S8S8_BGRA	SCE_GXM_COLOR_ BASE_FORMAT_ S8S8S8S8 | SCE_GXM_COLOR_ SWIZZLE4_BGRA	Pixels are written to memory in S8S8S8S8 format in BGRA order.
SCE_GXM_COLOR_ FORMAT_F16F16F16F16_ABGR	SCE_GXM_COLOR_ BASE_FORMAT_ F16F16F16F16 | SCE_GXM_COLOR_ SWIZZLE4_ABGR	Pixels are written to memory in F16F16F16F16 format in ABGR order.
SCE_GXM_COLOR_ FORMAT_F16F16F16F16_ARGB	SCE_GXM_COLOR_ BASE_FORMAT_ F16F16F16F16 | SCE_GXM_COLOR_ SWIZZLE4_ARGB	Pixels are written to memory in F16F16F16F16 format in ARGB order.
SCE_GXM_COLOR_ FORMAT_F16F16F16F16_RGBA	SCE_GXM_COLOR_ BASE_FORMAT_ F16F16F16F16 | SCE_GXM_COLOR_ SWIZZLE4_RGBA	Pixels are written to memory in F16F16F16F16 format in RGBA order.
SCE_GXM_COLOR_ FORMAT_F16F16F16F16_BGRA	SCE_GXM_COLOR_ BASE_FORMAT_ F16F16F16F16 | SCE_GXM_COLOR_ SWIZZLE4_BGRA	Pixels are written to memory in F16F16F16F16 format in BGRA order.

Macro	Value	Description
SCE_GXM_COLOR_ FORMAT_F32F32_GR	SCE_GXM_COLOR_ BASE_FORMAT_ F32F32 | SCE_GXM_COLOR_ SWIZZLE2_GR	Pixels are written to memory in F32F32 format in GR order.
SCE_GXM_COLOR_ FORMAT_F32F32_RG	SCE_GXM_COLOR_ BASE_FORMAT_ F32F32 | SCE_GXM_COLOR_ SWIZZLE2_RG	Pixels are written to memory in F32F32 format in RG order.
SCE_GXM_COLOR_ FORMAT_F10F11F11_BGR	SCE_GXM_COLOR_ BASE_FORMAT_ F11F11F10 | SCE_GXM_COLOR_ SWIZZLE3_BGR	Pixels are written to memory in F10F11F11 format in BGR order, A is discarded.
SCE_GXM_COLOR_ FORMAT_F11F11F10_RGB	SCE_GXM_COLOR_ BASE_FORMAT_ F11F11F10 | SCE_GXM_COLOR_ SWIZZLE3_RGB	Pixels are written to memory in F11F11F10 format in RGB order, A is discarded.
SCE_GXM_COLOR_ FORMAT_SE5M9M9M9_BGR	SCE_GXM_COLOR_ BASE_FORMAT_ SE5M9M9M9 | SCE_GXM_COLOR_ SWIZZLE3_BGR	Pixels are written to memory in SE5M9M9M9 format in BGR order, A is discarded.
SCE_GXM_COLOR_ FORMAT_SE5M9M9M9_RGB	SCE_GXM_COLOR_ BASE_FORMAT_ SE5M9M9M9 | SCE_GXM_COLOR_ SWIZZLE3_RGB	Pixels are written to memory in SE5M9M9M9 format in RGB order, A is discarded.
SCE_GXM_COLOR_ FORMAT_U2F10F10F10_ABGR	SCE_GXM_COLOR_ BASE_FORMAT_ U2F10F10F10 | SCE_GXM_COLOR_ SWIZZLE4_ABGR	Pixels are written to memory in U2F10F10F10 format in ABGR order.
SCE_GXM_COLOR_ FORMAT_U2F10F10F10_ARGB	SCE_GXM_COLOR_ BASE_FORMAT_ U2F10F10F10 | SCE_GXM_COLOR_ SWIZZLE4_ARGB	Pixels are written to memory in U2F10F10F10 format in ARGB order.
SCE_GXM_COLOR_ FORMAT_F10F10F10U2_RGBA	SCE_GXM_COLOR_ BASE_FORMAT_ U2F10F10F10 | SCE_GXM_COLOR_ SWIZZLE4_RGBA	Pixels are written to memory in F10F10F10U2 format in RGBA order.
SCE_GXM_COLOR_ FORMAT_F10F10F10U2_BGRA	SCE_GXM_COLOR_ BASE_FORMAT_ U2F10F10F10 | SCE_GXM_COLOR_ SWIZZLE4_BGRA	Pixels are written to memory in F10F10F10U2 format in BGRA order.
SCE_GXM_COLOR_ FORMAT_A8B8G8R8	SCE_GXM_COLOR_ FORMAT_U8U8U8U8_ABGR	Legacy name for SCE_GXM_COLOR_FORMAT_U8U8U8U8_ABGR.
SCE_GXM_COLOR_ FORMAT_A8R8G8B8	SCE_GXM_COLOR_ FORMAT_U8U8U8U8_ARGB	Legacy name for SCE_GXM_COLOR_FORMAT_U8U8U8U8_ARGB.
SCE_GXM_COLOR_ FORMAT_R5G6B5	SCE_GXM_COLOR_ FORMAT_U5U6U5_RGB	Legacy name for SCE_GXM_COLOR_FORMAT_U5U6U5_RGB.
SCE_GXM_COLOR_ FORMAT_A1R5G5B5	SCE_GXM_COLOR_ FORMAT_U1U5U5U5_ARGB	Legacy name for SCE_GXM_COLOR_FORMAT_U1U5U5U5_ARGB.

Macro	Value	Description
SCE_GXM_COLOR_FORMAT_A4R4G4B4	SCE_GXM_COLOR_FORMAT_U4U4U4U4_ARGB	Legacy name for SCE_GXM_COLOR_FORMAT_U4U4U4U4_ARGB.
SCE_GXM_COLOR_FORMAT_A8	SCE_GXM_COLOR_FORMAT_U8_A	Legacy name for SCE_GXM_COLOR_FORMAT_U8_A.

Description

The color formats. These are split into two sections: the full list of all color formats supported by the hardware, followed by some legacy defines for convenience. The full list uses a standard syntax of `FORMAT_SWIZZLE`.

The format part of the name is written for high-to-low bit ordering assuming the value is in a register. Note that registers are stored in memory in a little-endian format.

For 4 and 3-component formats in memory, the swizzle part of the name is the component ordering in the value stored to memory. For example, a pixel of format `SCE_GXM_COLOR_FORMAT_U4U4U4U4_ABGR` would have A in the high 4 bits and R in the low 4 bits if the 16-bit value was loaded into a register.

For 2 and 1-component formats in memory, the format in memory is always GR or R and the swizzle represents the selection from the ABGR value of the current pixel. For example, the format `SCE_GXM_COLOR_FORMAT_U8_A` would write the A component of each pixel to memory when the tile is finished.

For a full table of all color base formats, swizzles and supported output register formats please refer to the *GPU User's Guide*, Appendix A.

SceGxmColorSurface

Represents the destination for tile values.

Definition

```
#include <gxm/structs.h>
typedef struct SceGxmColorSurface {
   uint32_t pbeSidebandWord;
   uint32_t pbeEmitWords[SCE_GXM_PBE_EMIT_WORD_COUNT];
   uint32_t outputRegisterSize;
   SceGxmTexture backgroundTex;
} SceGxmColorSurface;
```

Members

pbeSidebandWord	An opaque sideband word.
pbeEmitWords	Opaque emit words.
outputRegisterSize	Output register size.
backgroundTex	Precomputed background object texture control words.

Description

Represents the destination for tile values.

Notes

Must only be modified using provided API calls.

SceGxmColorSurfaceDitherMode

The color surface dither mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmColorSurfaceDitherMode {
   SCE_GXM_COLOR_SURFACE_DITHER_DISABLED = 0x00000000U,
   SCE_GXM_COLOR_SURFACE_DITHER_ENABLED = 0x00000008U
} SceGxmColorSurfaceDitherMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_COLOR_SURFACE_DITHER_DISABLED`	`0x00000000U`	Dithering is disabled.
`SCE_GXM_COLOR_SURFACE_DITHER_ENABLED`	`0x00000008U`	Dithering is enabled.

Description

The color surface dither mode.

SceGxmColorSurfaceGammaMode

The color surface gamma mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmColorSurfaceGammaMode {
    SCE_GXM_COLOR_SURFACE_GAMMA_NONE = 0x00000000U,
    SCE_GXM_COLOR_SURFACE_GAMMA_R = 0x00001000U,
    SCE_GXM_COLOR_SURFACE_GAMMA_GR = 0x00003000U,
    SCE_GXM_COLOR_SURFACE_GAMMA_BGR = 0x00001000U
} SceGxmColorSurfaceGammaMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_COLOR_SURFACE_GAMMA_NONE`	`0x00000000U`	No gamma correction on pixel write.
`SCE_GXM_COLOR_SURFACE_GAMMA_R`	`0x00001000U`	Gamma correction is performed for the R component on pixel write.
`SCE_GXM_COLOR_SURFACE_GAMMA_GR`	`0x00003000U`	Gamma correction is performed for the G and R components on pixel write.
`SCE_GXM_COLOR_SURFACE_GAMMA_BGR`	`0x00001000U`	Gamma correction is performed for the B, G, and R components on pixel write.

Description

The color surface gamma mode.

Notes

`SCE_GXM_COLOR_SURFACE_GAMMA_R` and `SCE_GXM_COLOR_SURFACE_GAMMA_BGR` enumerations intentionally share the same value. The implied meaning of the value changes depending on the color surface format being used. Please see the *GPU User's Guide* for details.

SceGxmColorSurfaceScaleMode

Color surface scaling mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmColorSurfaceScaleMode {
    SCE_GXM_COLOR_SURFACE_SCALE_NONE = 0x00000000U,
    SCE_GXM_COLOR_SURFACE_SCALE_MSAA_DOWNSCALE = 0x00000001U
} SceGxmColorSurfaceScaleMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_COLOR_SURFACE_SCALE_NONE`	`0x00000000U`	Do not apply scaling. This enumerator may not be used when rendering with a 2xMSAA render target.
`SCE_GXM_COLOR_SURFACE_SCALE_MSAA_DOWNSCALE`	`0x00000001U`	Downscale samples to pixel level, or perform a 2x2 downscale when rendering without MSAA.

Description

Color surface scaling mode. Specifies the scaling to perform before a whole tile of shaded pixels is stored back to memory.

When using `SCE_GXM_MULTISAMPLE_4X`, scaling is optional. When enabled, the 2x2 block of samples for each pixel are merged into a single pixel before being stored to memory. When disabled, raw samples are stored to memory.

When using `SCE_GXM_MULTISAMPLE_2X`, scaling must be enabled. The 2x1 block of samples for each pixel are merged to a single pixel before being stored to memory.

When using `SCE_GXM_MULTISAMPLE_NONE`, scaling is optional. When enabled, a 2x2 downscale operation is applied to the pixels before they are stored to memory. When disabled, pixels are stored to memory directly.

SceGxmColorSurfaceType

The color surface memory layout types.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmColorSurfaceType {
    SCE_GXM_COLOR_SURFACE_LINEAR = 0x00000000U,
    SCE_GXM_COLOR_SURFACE_TILED = 0x04000000U,
    SCE_GXM_COLOR_SURFACE_SWIZZLED = 0x08000000U
} SceGxmColorSurfaceType;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_COLOR_SURFACE_LINEAR`	`0x00000000U`	The color surface uses a linear memory layout.
`SCE_GXM_COLOR_SURFACE_TILED`	`0x04000000U`	The color surface uses a tiled memory layout.
`SCE_GXM_COLOR_SURFACE_SWIZZLED`	`0x08000000U`	The color surface uses a swizzled memory layout.

Description

The color surface memory layout types.

SceGxmColorSwizzle1Mode

Defines the 1-component color format swizzles.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmColorSwizzle1Mode {
    SCE_GXM_COLOR_SWIZZLE1_R = 0x00000000U,
    SCE_GXM_COLOR_SWIZZLE1_G = 0x00100000U,
    SCE_GXM_COLOR_SWIZZLE1_A = 0x00100000U
} SceGxmColorSwizzle1Mode;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_COLOR_SWIZZLE1_R	0x00000000U	Color format uses the R component only (A, B and G are ignored)
SCE_GXM_COLOR_SWIZZLE1_G	0x00100000U	Color format uses the G component only (A, B and R are ignored)
SCE_GXM_COLOR_SWIZZLE1_A	0x00100000U	Color format uses the A component only (B, G and R are ignored)

Description

Defines the 1-component color format swizzles. Note that SCE_GXM_COLOR_SWIZZLE1_G and SCE_GXM_COLOR_SWIZZLE1_A intentionally have the same value. This is due to some color formats only supporting R or G, whereas others support on R or A.

SceGxmColorSwizzle2Mode

Defines the 2-component color format swizzles.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmColorSwizzle2Mode {
    SCE_GXM_COLOR_SWIZZLE2_GR = 0x00000000U,
    SCE_GXM_COLOR_SWIZZLE2_RG = 0x00100000U,
    SCE_GXM_COLOR_SWIZZLE2_RA = 0x00200000U,
    SCE_GXM_COLOR_SWIZZLE2_AR = 0x00300000U
} SceGxmColorSwizzle2Mode;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_COLOR_SWIZZLE2_GR	0x00000000U	Color format written in GR order (A and B are ignored)
SCE_GXM_COLOR_SWIZZLE2_RG	0x00100000U	Color format written in RG order (A and B are ignored)
SCE_GXM_COLOR_SWIZZLE2_RA	0x00200000U	Color format written in RA order (B and G are ignored)
SCE_GXM_COLOR_SWIZZLE2_AR	0x00300000U	Color format written in AR order (B and G are ignored)

Description

Defines the 2-component color format swizzles.

SceGxmColorSwizzle3Mode

Defines the 3-component color format swizzles.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmColorSwizzle3Mode {
    SCE_GXM_COLOR_SWIZZLE3_BGR = 0x00000000U,
    SCE_GXM_COLOR_SWIZZLE3_RGB = 0x00100000U
} SceGxmColorSwizzle3Mode;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_COLOR_SWIZZLE3_BGR	0x00000000U	Color format written in BGR order (A is ignored)
SCE_GXM_COLOR_SWIZZLE3_RGB	0x00100000U	Color format written in RGB order (A is ignored)

Description

Defines the 3-component color format swizzles.

SceGxmColorSwizzle4Mode

Defines the 4-component color format swizzles.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmColorSwizzle4Mode {
    SCE_GXM_COLOR_SWIZZLE4_ABGR = 0x00000000U,
    SCE_GXM_COLOR_SWIZZLE4_ARGB = 0x00100000U,
    SCE_GXM_COLOR_SWIZZLE4_RGBA = 0x00200000U,
    SCE_GXM_COLOR_SWIZZLE4_BGRA = 0x00300000U
} SceGxmColorSwizzle4Mode;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_COLOR_SWIZZLE4_ABGR	0x00000000U	Color format written in ABGR order.
SCE_GXM_COLOR_SWIZZLE4_ARGB	0x00100000U	Color format written in ARGB order.
SCE_GXM_COLOR_SWIZZLE4_RGBA	0x00200000U	Color format written in RGBA order.
SCE_GXM_COLOR_SWIZZLE4_BGRA	0x00300000U	Color format written in BGRA order.

Description

Defines the 4-component color format swizzles.

SceGxmCommandList

Represents a group of draw calls built up using a deferred context.

Definition

```
#include <gxm/structs.h>
typedef struct SceGxmCommandList {
    uint32_t data[SCE_GXM_COMMAND_LIST_WORD_COUNT];
} SceGxmCommandList;
```

Members

data	Opaque contents.

Description

Represents a group of draw calls built up using a deferred context.

SceGxmContext

The opaque data structure for a rendering context.

Definition

```
#include <gxm/context.h>
typedef struct SceGxmContext;
```

Description

The opaque data structure for a rendering context. The rendering context may be an immediate context or a deferred context. Many libgxm functions, such as those involved with setting state and drawing, are supported on both immediate and deferred contexts, but some functions, such as those involved with scenes or command lists, require a context of a specific type. These functions will be clearly marked in their reference documentation.

Call `sceGxmGetContextType()` to determine whether a context is a deferred or an immediate context.

SceGxmContextParams

The parameters for creating the immediate context.

Definition

```
#include <gxm/context.h>
typedef struct SceGxmContextParams {
    void *hostMem;
    uint32_t hostMemSize;
    void *vdmRingBufferMem;
    uint32_t vdmRingBufferMemSize;
    void *vertexRingBufferMem;
    uint32_t vertexRingBufferMemSize;
    void *fragmentRingBufferMem;
    uint32_t fragmentRingBufferMemSize;
    void *fragmentUsseRingBufferMem;
    uint32_t fragmentUsseRingBufferMemSize;
    uint32_t fragmentUsseRingBufferOffset;
} SceGxmContextParams;
```

Members

hostMem	Host memory for the `SceGxmContext` structure. This should be standard cached CPU memory, such as that returned by libc malloc. This should be aligned to 4 bytes.
hostMemSize	The size of the host memory pointed to by *hostMem*. The minimum size this memory can be is defined by `SCE_GXM_MINIMUM_CONTEXT_HOST_MEM_SIZE`. This should be aligned to 4 bytes.
vdmRingBufferMem	Memory for the VDM ring buffer. This should be mapped to the GPU with read access. A sensible default size is `SCE_GXM_DEFAULT_VDM_RING_BUFFER_SIZE`. This should be aligned to 4 bytes.
vdmRingBufferMemSize	The size in bytes of the VDM ring buffer memory pointed to by *vdmRingBufferMem*. This should be aligned to 4 bytes.
vertexRingBufferMem	The memory for the vertex ring buffer. This should be mapped to the GPU with read access. A sensible default size is `SCE_GXM_DEFAULT_VERTEX_RING_BUFFER_SIZE`. This should be aligned to 4 bytes.
vertexRingBufferMemSize	The size in bytes of the vertex ring buffer memory pointed to by *vertexRingBufferMem*. This should be aligned to 4 bytes.
fragmentRingBufferMem	The memory for the fragment ring buffer. This should be mapped to the GPU with read access. A sensible default size is `SCE_GXM_DEFAULT_FRAGMENT_RING_BUFFER_SIZE`. This should be aligned to 4 bytes.
fragmentRingBufferMemSize	The size in bytes of the fragment ring buffer memory pointed to by *fragmentRingBufferMem*. This should be aligned to 4 bytes.
fragmentUsseRingBufferMem	The memory for the fragment USSE ring buffer. This should be mapped as fragment USSE code. A sensible default size is `SCE_GXM_DEFAULT_FRAGMENT_USSE_RING_BUFFER_SIZE`. This should be aligned to 4 bytes.

`fragmentUsseRingBufferMemSize`	The size in bytes of the fragment USSE ring buffer memory pointed to by `fragmentUsseRingBufferMem`. This should be aligned to 4 bytes.
`fragmentUsseRingBufferOffset`	The USSE offset of the start of the fragment USSE ring buffer memory pointed to by `fragmentUsseRingBufferMem`.

Description

The parameters for creating the immediate context.

SceGxmContextType

The type of a rendering context.

Definition

```
#include <gxm/context.h>
typedef enum SceGxmContextType {
    SCE_GXM_CONTEXT_TYPE_IMMEDIATE,
    SCE_GXM_CONTEXT_TYPE_DEFERRED
} SceGxmContextType;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_CONTEXT_TYPE_IMMEDIATE`	N/A	The immediate context, which will have been created with `sceGxmCreateContext()`.
`SCE_GXM_CONTEXT_TYPE_DEFERRED`	N/A	A deferred context, which will have been created with `sceGxmCreateDeferredContext()`.

Description

The type of a rendering context.

SceGxmCullMode

The backface culling modes.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmCullMode {
    SCE_GXM_CULL_NONE = 0x00000000U,
    SCE_GXM_CULL_CW = 0x00000001U,
    SCE_GXM_CULL_CCW = 0x00000002U
} SceGxmCullMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_CULL_NONE`	`0x00000000U`	No culling.
`SCE_GXM_CULL_CW`	`0x00000001U`	Cull triangles with clockwise window coordinates.
`SCE_GXM_CULL_CCW`	`0x00000002U`	Cull triangles with counter-clockwise window coordinates.

Description

The backface culling modes.

SceGxmDeferredContextParams

The parameters for creating a deferred context.

Definition

```
#include <gxm/context.h>
typedef struct SceGxmDeferredContextParams {
    void *hostMem;
    uint32_t hostMemSize;
    SceGxmDeferredContextCallback vdmCallback;
    SceGxmDeferredContextCallback vertexCallback;
    SceGxmDeferredContextCallback fragmentCallback;
    void *userData;
    void *vdmBufferMem;
    uint32_t vdmBufferMemSize;
    void *vertexBufferMem;
    uint32_t vertexBufferMemSize;
    void *fragmentBufferMem;
    uint32_t fragmentBufferMemSize;
} SceGxmDeferredContextParams;
```

Members

hostMem	Host memory for the `SceGxmContext` structure. This should be standard cached CPU memory, such as that returned by libc malloc. This should be aligned to 4 bytes.
hostMemSize	The size of the host memory pointed to by *hostMem*. This size must be at least `SCE_GXM_MINIMUM_CONTEXT_HOST_MEM_SIZE`. This should be aligned to 4 bytes.
vdmCallback	The callback function called by a deferred context when memory is required for VDM stream entries.
vertexCallback	The callback function called by a deferred context when memory is required for supporting vertex data structures.
fragmentCallback	The callback function called by a deferred context when memory is required for supporting fragment data structures.
userData	Optional user data pointer which is passed to *vdmCallback*, *vertexCallback* and *fragmentCallback* functions.
vdmBufferMem	Optional memory for the initial VDM buffer. This should be mapped to the GPU with read access. If `NULL` is specified then memory must be provided through the callback pointed to by *vdmCallback*. This should be aligned to 4 bytes.
vdmBufferMemSize	The size in bytes of the VDM buffer memory pointed to by *vdmBufferMem* or 0 if *vdmBufferMem* is `NULL`. If non-zero, the size must be at least `SCE_GXM_MINIMUM_DEFERRED_CONTEXT_BUFFER_SIZE`. This should be aligned to 4 bytes.
vertexBufferMem	Optional memory for the initial vertex buffer. This should be mapped to the GPU with read access. If `NULL` is specified then memory must be provided through the callback pointed to by *vertexCallback*. This should be aligned to 4 bytes.
vertexBufferMemSize	The size in bytes of the vertex buffer memory pointed to by *vertexBufferMem* or 0 if *vertexBufferMem* is `NULL`. If non-zero, the size must be at least `SCE_GXM_MINIMUM_DEFERRED_CONTEXT_BUFFER_SIZE`. This should be aligned to 4 bytes.

fragmentBufferMem	Optional memory for the initial fragment buffer. This should be mapped to the GPU with read access. If `NULL` is specified then memory must be provided through the callback pointed to by *fragmentCallback*. This should be aligned to 4 bytes.
fragmentBufferMemSize	The size in bytes of the fragment buffer memory pointed to by *fragmentBufferMem* or 0 if *fragmentBufferMem* is `NULL`. If non-zero, the size must be at least `SCE_GXM_MINIMUM_DEFERRED_CONTEXT_BUFFER_SIZE`. This should be aligned to 4 bytes.

Description

The parameters for creating a deferred context.

SceGxmDepthFunc

The depth compare functions.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmDepthFunc {
    SCE_GXM_DEPTH_FUNC_NEVER = 0x00000000U,
    SCE_GXM_DEPTH_FUNC_LESS = 0x00400000U,
    SCE_GXM_DEPTH_FUNC_EQUAL = 0x00800000U,
    SCE_GXM_DEPTH_FUNC_LESS_EQUAL = 0x00c00000U,
    SCE_GXM_DEPTH_FUNC_GREATER = 0x01000000U,
    SCE_GXM_DEPTH_FUNC_NOT_EQUAL = 0x01400000U,
    SCE_GXM_DEPTH_FUNC_GREATER_EQUAL = 0x01800000U,
    SCE_GXM_DEPTH_FUNC_ALWAYS = 0x01c00000U
} SceGxmDepthFunc;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_DEPTH_FUNC_NEVER	0x00000000U	Never pass.
SCE_GXM_DEPTH_FUNC_LESS	0x00400000U	Pass when input depth is less than stored depth.
SCE_GXM_DEPTH_FUNC_EQUAL	0x00800000U	Pass when input depth is equal to stored depth.
SCE_GXM_DEPTH_FUNC_LESS_EQUAL	0x00c00000U	Pass when input depth is less than or equal to stored depth.
SCE_GXM_DEPTH_FUNC_GREATER	0x01000000U	Pass when input depth is greater than stored depth.
SCE_GXM_DEPTH_FUNC_NOT_EQUAL	0x01400000U	Pass when input depth is not equal to stored depth.
SCE_GXM_DEPTH_FUNC_GREATER_EQUAL	0x01800000U	Pass when input depth is greater than or equal to stored depth.
SCE_GXM_DEPTH_FUNC_ALWAYS	0x01c00000U	Always pass.

Description

The depth compare functions. To ensure that depth data can be preserved during partial render, when the current scene is using a depth/stencil surface that does not contain depth data as part of the format, only the depth functions SCE_GXM_DEPTH_FUNC_NEVER and SCE_GXM_DEPTH_FUNC_ALWAYS may be used.

SceGxmDepthStencilForceLoadMode

Depth/stencil surface force load mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmDepthStencilForceLoadMode {
    SCE_GXM_DEPTH_STENCIL_FORCE_LOAD_DISABLED = 0x00000000U,
    SCE_GXM_DEPTH_STENCIL_FORCE_LOAD_ENABLED = 0x00000002U
} SceGxmDepthStencilForceLoadMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_DEPTH_STENCIL_FORCE_LOAD_DISABLED`	`0x00000000U`	No forced load, depth values are uninitialized at the start of each tile.
`SCE_GXM_DEPTH_STENCIL_FORCE_LOAD_ENABLED`	`0x00000002U`	Forced load, depth values are loaded from memory at the start of each tile.

Description

Depth/stencil surface force load mode.

SceGxmDepthStencilForceStoreMode

Depth/stencil surface force store mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmDepthStencilForceStoreMode {
    SCE_GXM_DEPTH_STENCIL_FORCE_STORE_DISABLED = 0x00000000U,
    SCE_GXM_DEPTH_STENCIL_FORCE_STORE_ENABLED = 0x00000004U
} SceGxmDepthStencilForceStoreMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_DEPTH_STENCIL_FORCE_STORE_DISABLED`	`0x00000000U`	No forced store, depth values are not stored to memory at the end of each tile.
`SCE_GXM_DEPTH_STENCIL_FORCE_STORE_ENABLED`	`0x00000004U`	Forced store, depth values are stored to memory at the end of each tile.

Description

Depth/stencil surface force store mode.

SceGxmDepthStencilFormat

The depth/stencil surface formats.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmDepthStencilFormat {
    SCE_GXM_DEPTH_STENCIL_FORMAT_DF32 = 0x00044000U,
    SCE_GXM_DEPTH_STENCIL_FORMAT_S8 = 0x00022000U,
    SCE_GXM_DEPTH_STENCIL_FORMAT_DF32_S8 = 0x00066000U,
    SCE_GXM_DEPTH_STENCIL_FORMAT_DF32M = 0x000CC000U,
    SCE_GXM_DEPTH_STENCIL_FORMAT_DF32M_S8 = 0x000EE000U,
    SCE_GXM_DEPTH_STENCIL_FORMAT_S8D24 = 0x01266000U,
    SCE_GXM_DEPTH_STENCIL_FORMAT_D16 = 0x02444000U
} SceGxmDepthStencilFormat;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_DEPTH_STENCIL_FORMAT_DF32`	`0x00044000U`	32-bit floating point Z only
`SCE_GXM_DEPTH_STENCIL_FORMAT_S8`	`0x00022000U`	8-bit stencil only
`SCE_GXM_DEPTH_STENCIL_FORMAT_DF32_S8`	`0x00066000U`	Separate 32-bit floating point Z and 8-bit stencil.
`SCE_GXM_DEPTH_STENCIL_FORMAT_DF32M`	`0x000CC000U`	32-bit floating point Z with mask in sign bit
`SCE_GXM_DEPTH_STENCIL_FORMAT_DF32M_S8`	`0x000EE000U`	Separate 32-bit floating point Z with mask in sign bit and 8-bit stencil.
`SCE_GXM_DEPTH_STENCIL_FORMAT_S8D24`	`0x01266000U`	Packed 24-bit Z and 8-bit stencil.
`SCE_GXM_DEPTH_STENCIL_FORMAT_D16`	`0x02444000U`	16-bit Z only

Description

The depth/stencil surface formats.

SceGxmDepthStencilSurface

Controls how depth and stencil values are loaded from memory at the start of each tile, and how they are saved to memory at the end of each tile.

Definition

```
#include <gxm/structs.h>
typedef struct SceGxmDepthStencilSurface {
   uint32_t zlsControl;
   void *depthData;
   void *stencilData;
   float backgroundDepth;
   uint32_t backgroundControl;
} SceGxmDepthStencilSurface;
```

Members

zlsControl	An opaque control word.
depthData	A pointer to the depth data or `NULL`.
stencilData	A pointer to the stencil data or `NULL`.
backgroundDepth	The 32-bit floating point background object depth value.
backgroundControl	An opaque background object control register.

Description

Controls how depth and stencil values are loaded from memory at the start of each tile, and how they are saved to memory at the end of each tile.

Notes

Must only be modified using provided API calls.

SceGxmDepthStencilSurfaceType

The depth/stencil surface memory layout types.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmDepthStencilSurfaceType {
    SCE_GXM_DEPTH_STENCIL_SURFACE_LINEAR = 0x00000000U,
    SCE_GXM_DEPTH_STENCIL_SURFACE_TILED = 0x00011000U
} SceGxmDepthStencilSurfaceType;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_DEPTH_STENCIL_SURFACE_LINEAR`	`0x00000000U`	Depth/stencil surface uses a linear memory layout.
`SCE_GXM_DEPTH_STENCIL_SURFACE_TILED`	`0x00011000U`	Depth/stencil surface uses a tiled memory layout.

Description

The depth/stencil surface memory layout types.

SceGxmDepthWriteMode

Depth write enable mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmDepthWriteMode {
    SCE_GXM_DEPTH_WRITE_DISABLED = 0x00100000U,
    SCE_GXM_DEPTH_WRITE_ENABLED = 0x00000000U
} SceGxmDepthWriteMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_DEPTH_WRITE_DISABLED`	`0x00100000U`	Depth writes disabled.
`SCE_GXM_DEPTH_WRITE_ENABLED`	`0x00000000U`	Depth writes enabled.

Description

Depth write enable mode. Note that this setting only affects depth writes to the local cache used for each tile as it is processed. To then ensure that tiles are stored to memory, the appropriate `SceGxmDepthStencilForceStoreMode` should be set on the depth/stencil surface.

SceGxmEdgeEnableFlags

The edge enable bits for primitives of type SCE_GXM_PRIMITIVE_TRIANGLE_EDGES.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmEdgeEnableFlags {
    SCE_GXM_EDGE_ENABLE_01 = 0x00000100U,
    SCE_GXM_EDGE_ENABLE_12 = 0x00000200U,
    SCE_GXM_EDGE_ENABLE_20 = 0x00000400U
} SceGxmEdgeEnableFlags;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_EDGE_ENABLE_01	0x00000100U	Enable edge 0-1.
SCE_GXM_EDGE_ENABLE_12	0x00000200U	Enable edge 1-2.
SCE_GXM_EDGE_ENABLE_20	0x00000400U	Enable edge 2-0.

Description

The edge enable bits for primitives of type SCE_GXM_PRIMITIVE_TRIANGLE_EDGES. Additional detail can be found in the *GPU User's Guide*.

SceGxmErrorCode

A typedef to clarify when a return value is an error code.

Definition

```
#include <gxm/error.h>
typedef int32_t SceGxmErrorCode;
```

Description

A typedef to clarify when a return value is an error code.

SceGxmFragmentProgram

The data structure for fragment programs.

Definition

```
#include <gxm/fragment_program.h>
typedef struct SceGxmFragmentProgram;
```

Description

The data structure for fragment programs. This structure is currently opaque, filled out internally by the shader patcher.

SceGxmFragmentProgramMode

Fragment program enable mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmFragmentProgramMode {
   SCE_GXM_FRAGMENT_PROGRAM_DISABLED = 0x00200000U,
   SCE_GXM_FRAGMENT_PROGRAM_ENABLED = 0x00000000U
} SceGxmFragmentProgramMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_FRAGMENT_PROGRAM_DISABLED`	`0x00200000U`	Fragment program disabled, no pixels shaded.
`SCE_GXM_FRAGMENT_PROGRAM_ENABLED`	`0x00000000U`	Fragment program enabled, pixels shaded.

Description

Fragment program enable mode. When the fragment program is disabled, only the depth/stencil test is performed. No pixels are shaded. Fragment program can be enabled or disabled independently when using two-sided rendering.

SceGxmIndexFormat

The index format used in draw calls.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmIndexFormat {
    SCE_GXM_INDEX_FORMAT_U16 = 0x00000000U,
    SCE_GXM_INDEX_FORMAT_U32 = 0x01000000U
} SceGxmIndexFormat;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_INDEX_FORMAT_U16`	`0x00000000U`	Uses 16-bit indices.
`SCE_GXM_INDEX_FORMAT_U32`	`0x01000000U`	Uses 32-bit indices (only the low 24 bits are used)

Description

The index format used in draw calls.

SceGxmIndexSource

The index source type for indexing into vertex streams.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmIndexSource {
    SCE_GXM_INDEX_SOURCE_INDEX_16BIT = 0x00000000U,
    SCE_GXM_INDEX_SOURCE_INDEX_32BIT = 0x00000001U,
    SCE_GXM_INDEX_SOURCE_INSTANCE_16BIT = 0x00000002U,
    SCE_GXM_INDEX_SOURCE_INSTANCE_32BIT = 0x00000003U
} SceGxmIndexSource;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_INDEX_SOURCE_INDEX_16BIT`	`0x00000000U`	The stream is indexed using the index values and all values must be less than 64K. However, they can use either U16 or U32 format in memory.
`SCE_GXM_INDEX_SOURCE_INDEX_32BIT`	`0x00000001U`	The stream is indexed using the index values.
`SCE_GXM_INDEX_SOURCE_INSTANCE_16BIT`	`0x00000002U`	The stream is indexed using the instance number and can only be used with draw calls of up to 64K instances.
`SCE_GXM_INDEX_SOURCE_INSTANCE_32BIT`	`0x00000003U`	The stream is indexed using the instance number.

Description

The index source type for indexing into vertex streams. The 16-bit index sources generate more efficient PDS code since they only need to implement a 16-bit multiply. However, this imposes restrictions on either the maximum index values or maximum instance counts. Note that it is valid to use `SCE_GXM_INDEX_FORMAT_U32` indices with `SCE_GXM_INDEX_SOURCE_INDEX_16BIT` as long as the actual values are not larger than 16 bit.

SceGxmInitializeFlags

Flags for libgxm initialization.

Definition

```
#include <gxm/init.h>
typedef enum SceGxmInitializeFlags {
    SCE_GXM_INITIALIZE_FLAG_DISPLAY_QUEUE_THREAD_AFFINITY_CPU_0 =
    0x00000000U,
    SCE_GXM_INITIALIZE_FLAG_DISPLAY_QUEUE_THREAD_AFFINITY_CPU_1 =
    0x00010000U,
    SCE_GXM_INITIALIZE_FLAG_DISPLAY_QUEUE_THREAD_AFFINITY_CPU_2 = 0x00020000U
} SceGxmInitializeFlags;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_INITIALIZE_FLAG_DISPLAY_QUEUE_THREAD_AFFINITY_CPU_0	0x00000000U	Display queue thread and callback will run on user CPU 0.
SCE_GXM_INITIALIZE_FLAG_DISPLAY_QUEUE_THREAD_AFFINITY_CPU_1	0x00010000U	Display queue thread and callback will run on user CPU 1.
SCE_GXM_INITIALIZE_FLAG_DISPLAY_QUEUE_THREAD_AFFINITY_CPU_2	0x00020000U	Display queue thread and callback will run on user CPU 2.

Description

Flags for libgxm initialization.

SceGxmInitializeParams

The initialization parameters for the library.

Definition

```
#include <gxm/init.h>
typedef struct SceGxmInitializeParams {
    uint32_t flags;
    uint32_t displayQueueMaxPendingCount;
    SceGxmDisplayQueueCallback displayQueueCallback;
    uint32_t displayQueueCallbackDataSize;
    uint32_t parameterBufferSize;
} SceGxmInitializeParams;
```

Members

flags	Flags from `SceGxmInitializeFlags`.
displayQueueMaxPendingCount	The maximum number of pending display swaps to allow before blocking. This is usually a low number, such as 2 or 3.
displayQueueCallback	The callback function to use to display a buffer. This function is called when the GPU has completed rendering. It is responsible for flipping the display buffer and blocking until the flip operation is completed. After the function returns, the GPU will be allowed to continue. This means there is the potential for the old display buffer to be overwritten immediately. Since the libgxm context is single threaded, no libgxm context functions or synchronization functions should be called from this callback function; otherwise undefined behavior could occur. In particular, neither the `sceGxmNotificationWait()` synchronization function or any function that takes a libgxm context (such as `sceGxmBeginScene()`, `sceGxmDraw()` or `sceGxmFinish()`) should be called. The expected behavior is to call `sceDisplaySetFrameBuf()` to enqueue a new display buffer address. A call to `sceDisplayWaitSetFrameBuf()` should follow this if the flip operation was called with `SCE_DISPLAY_UPDATETIMING_NEXTVSYNC`. This ensures that future GPU operations on the old front buffer do not start until the new front buffer is being displayed.
displayQueueCallbackDataSize	The size of the data that needs to be passed to the callback function. Storage will be allocated to ensure this data can be copied to the display queue.The total size of the storage of `displayQueueMaxPendingCount*` `displayQueueCallbackDataSize` must not exceed 512 bytes.
parameterBufferSize	The size of parameter buffer to allocate.

Description

The initialization parameters for the library.

SceGxmLineFillLastPixelMode

Line last pixel fill mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmLineFillLastPixelMode {
   SCE_GXM_LINE_FILL_LAST_PIXEL_DISABLED = 0x00000000U,
   SCE_GXM_LINE_FILL_LAST_PIXEL_ENABLED = 0x00080000U
} SceGxmLineFillLastPixelMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_LINE_FILL_LAST_PIXEL_DISABLED`	`0x00000000U`	The last pixel of a line is not rendered.
`SCE_GXM_LINE_FILL_LAST_PIXEL_ENABLED`	`0x00080000U`	The last pixel of a line is rendered.

Description

Line last pixel fill mode. When enabled, the last pixel of a line is filled.

SceGxmMemoryAttribFlags

Flags that can be used when mapping memory using sceGxmMapMemory().

Definition

```
#include <gxm/memory.h>
typedef enum SceGxmMemoryAttribFlags {
    SCE_GXM_MEMORY_ATTRIB_READ = 0x00000001,
    SCE_GXM_MEMORY_ATTRIB_WRITE = 0x00000002
} SceGxmMemoryAttribFlags;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_MEMORY_ATTRIB_READ`	`0x00000001`	The GPU is permitted to read.
`SCE_GXM_MEMORY_ATTRIB_WRITE`	`0x00000002`	The GPU is permitted to write.

Description

Flags that can be used when mapping memory using sceGxmMapMemory().

SceGxmMidSceneFlags

Mid-scene flush flags for sceGxmMidSceneFlush().

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmMidSceneFlags {
    SCE_GXM_MIDSCENE_PRESERVE_DEFAULT_UNIFORM_BUFFERS = 0x00000001U
} SceGxmMidSceneFlags;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_MIDSCENE_PRESERVE_DEFAULT_UNIFORM_BUFFERS	0x00000001U	Preserve the vertex default uniform buffer contents during the mid-scene flush.

Description

Mid-scene flush flags for sceGxmMidSceneFlush().

SceGxmMultisampleMode

The multisample modes.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmMultisampleMode {
    SCE_GXM_MULTISAMPLE_NONE,
    SCE_GXM_MULTISAMPLE_2X,
    SCE_GXM_MULTISAMPLE_4X
} SceGxmMultisampleMode;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_MULTISAMPLE_NONE	N/A	Single central sample.
SCE_GXM_MULTISAMPLE_2X	N/A	2x diagonal samples.
SCE_GXM_MULTISAMPLE_4X	N/A	4x rotated grid.

Description

The multisample modes.

SceGxmNotification

This struct describes a GPU notification, which occurs when a scene completes on either the vertex or fragment pipeline.

Definition

```
#include <gxm/structs.h>
typedef struct SceGxmNotification {
   uint32_t *address;
   uint32_t value;
} SceGxmNotification;
```

Members

address	The address that will be written to by the GPU.
value	The 32-bit value that will be written.

Description

This struct describes a GPU notification, which occurs when a scene completes on either the vertex or fragment pipeline. These notifications must use addresses within the notification region allocated at initialization time. Once libgxm has been initialized, the base address of the region can be queried using `sceGxmGetNotificationRegion()`.

SceGxmOutputRegisterFormat

When using a fragment program that does not declare its output format in the shader code, one of these formats may be selected as the format to use for the `COLOR0` output.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmOutputRegisterFormat {
    SCE_GXM_OUTPUT_REGISTER_FORMAT_DECLARED,
    SCE_GXM_OUTPUT_REGISTER_FORMAT_UCHAR4,
    SCE_GXM_OUTPUT_REGISTER_FORMAT_CHAR4,
    SCE_GXM_OUTPUT_REGISTER_FORMAT_USHORT2,
    SCE_GXM_OUTPUT_REGISTER_FORMAT_SHORT2,
    SCE_GXM_OUTPUT_REGISTER_FORMAT_HALF4,
    SCE_GXM_OUTPUT_REGISTER_FORMAT_HALF2,
    SCE_GXM_OUTPUT_REGISTER_FORMAT_FLOAT2,
    SCE_GXM_OUTPUT_REGISTER_FORMAT_FLOAT
} SceGxmOutputRegisterFormat;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_OUTPUT_REGISTER_FORMAT_DECLARED`	N/A	Use the output format declared in the shader code.
`SCE_GXM_OUTPUT_REGISTER_FORMAT_UCHAR4`	N/A	Perform a normalized pack to unsigned char4.
`SCE_GXM_OUTPUT_REGISTER_FORMAT_CHAR4`	N/A	Perform a normalized pack to char4.
`SCE_GXM_OUTPUT_REGISTER_FORMAT_USHORT2`	N/A	Perform a normalized pack to unsigned short2.
`SCE_GXM_OUTPUT_REGISTER_FORMAT_SHORT2`	N/A	Perform a normalized pack to short2.
`SCE_GXM_OUTPUT_REGISTER_FORMAT_HALF4`	N/A	Perform a pack to half4.
`SCE_GXM_OUTPUT_REGISTER_FORMAT_HALF2`	N/A	Perform a pack to half2.
`SCE_GXM_OUTPUT_REGISTER_FORMAT_FLOAT2`	N/A	Perform a pack to float2.
`SCE_GXM_OUTPUT_REGISTER_FORMAT_FLOAT`	N/A	Perform a pack to float.

Description

When using a fragment program that does not declare its output format in the shader code, one of these formats may be selected as the format to use for the `COLOR0` output. This format must match the output register size of the color surface. For details of which color surface formats support which output register formats, please see the *GPU User's Guide*.

SceGxmOutputRegisterSize

Output register size used by the color surface.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmOutputRegisterSize {
    SCE_GXM_OUTPUT_REGISTER_SIZE_32BIT = 0x00000000U,
    SCE_GXM_OUTPUT_REGISTER_SIZE_64BIT = 0x00000001U
} SceGxmOutputRegisterSize;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_OUTPUT_REGISTER_SIZE_32BIT`	`0x00000000U`	Output register size is 32-bit.
`SCE_GXM_OUTPUT_REGISTER_SIZE_64BIT`	`0x00000001U`	Output register size is 64-bit.

Description

Output register size used by the color surface. This field instructs the GPU to ensure that there is either exactly 32 bits or 64 bits of on-chip storage (called the "output register") per pixel.

SceGxmPassType

The pass type of a fragment program.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmPassType {
    SCE_GXM_PASS_TYPE_OPAQUE = 0x00000000U,
    SCE_GXM_PASS_TYPE_TRANSLUCENT = 0x02000000U,
    SCE_GXM_PASS_TYPE_DISCARD = 0x04000000U,
    SCE_GXM_PASS_TYPE_MASK_UPDATE = 0x06000000U,
    SCE_GXM_PASS_TYPE_DEPTH_REPLACE = 0x0A000000U
} SceGxmPassType;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_PASS_TYPE_OPAQUE	0x00000000U	Opaque with no discard or depth replace.
SCE_GXM_PASS_TYPE_TRANSLUCENT	0x02000000U	Translucent with no discard or depth replace.
SCE_GXM_PASS_TYPE_DISCARD	0x04000000U	Translucent with discard but no depth replace.
SCE_GXM_PASS_TYPE_MASK_UPDATE	0x06000000U	Fragment program updates mask bit only.
SCE_GXM_PASS_TYPE_DEPTH_REPLACE	0x0A000000U	Depth replace used, can also be translucent or use discard.

Description

The pass type of a fragment program.

SceGxmPolygonMode

The polygon filling modes.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmPolygonMode {
    SCE_GXM_POLYGON_MODE_TRIANGLE_FILL = 0x00000000U,
    SCE_GXM_POLYGON_MODE_LINE = 0x00008000U,
    SCE_GXM_POLYGON_MODE_POINT_10UV = 0x00010000U,
    SCE_GXM_POLYGON_MODE_POINT = 0x00018000U,
    SCE_GXM_POLYGON_MODE_POINT_01UV = 0x00020000U,
    SCE_GXM_POLYGON_MODE_TRIANGLE_LINE = 0x00028000U,
    SCE_GXM_POLYGON_MODE_TRIANGLE_POINT = 0x00030000U
} SceGxmPolygonMode;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_POLYGON_MODE_TRIANGLE_FILL	0x00000000U	Triangle polygons with shaded interior.
SCE_GXM_POLYGON_MODE_LINE	0x00008000U	Lines shaded as lines.
SCE_GXM_POLYGON_MODE_POINT_10UV	0x00010000U	Point sprites using UV coordinate range (0,1) to (1,0)
SCE_GXM_POLYGON_MODE_POINT	0x00018000U	Point sprites using supplied UV values.
SCE_GXM_POLYGON_MODE_POINT_01UV	0x00020000U	Point sprites using UV coordinate range (0,0) to (1,1)
SCE_GXM_POLYGON_MODE_TRIANGLE_LINE	0x00028000U	Triangle polygons with edges shaded only.
SCE_GXM_POLYGON_MODE_TRIANGLE_POINT	0x00030000U	Triangle polygons with vertices shaded only.

Description

The polygon filling modes.

SceGxmPrecomputedDraw

The precomputed draw command.

Definition

```
#include <gxm/precomputation.h>
typedef struct SceGxmPrecomputedDraw {
    uint32_t data[SCE_GXM_PRECOMPUTED_DRAW_WORD_COUNT];
} SceGxmPrecomputedDraw;
```

Members

data	Opaque contents.

Description

The precomputed draw command. Allows for a draw call to be done using precomputed data, which reduces CPU overheads.

SceGxmPrecomputedFragmentState

The precomputed fragment state.

Definition

```
#include <gxm/precomputation.h>
typedef struct SceGxmPrecomputedFragmentState {
    uint32_t data[SCE_GXM_PRECOMPUTED_FRAGMENT_STATE_WORD_COUNT];
} SceGxmPrecomputedFragmentState;
```

Members

data	Opaque contents.

Description

The precomputed fragment state. Allows for fragment secondary and primary updates to be precomputed, which reduces CPU overheads.

SceGxmPrecomputedVertexState

The precomputed vertex state.

Definition

```
#include <gxm/precomputation.h>
typedef struct SceGxmPrecomputedVertexState {
   uint32_t data[SCE_GXM_PRECOMPUTED_VERTEX_STATE_WORD_COUNT];
} SceGxmPrecomputedVertexState;
```

Members

data	Opaque contents.

Description

The precomputed vertex state. Allows for vertex secondary updates to be precomputed, which reduces CPU overheads.

SceGxmPrimitiveType

The primitive types.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmPrimitiveType {
    SCE_GXM_PRIMITIVE_TRIANGLES = 0x00000000U,
    SCE_GXM_PRIMITIVE_LINES = 0x04000000U,
    SCE_GXM_PRIMITIVE_POINTS = 0x08000000U,
    SCE_GXM_PRIMITIVE_TRIANGLE_STRIP = 0x0c000000U,
    SCE_GXM_PRIMITIVE_TRIANGLE_FAN = 0x10000000U,
    SCE_GXM_PRIMITIVE_TRIANGLE_EDGES = 0x14000000U
} SceGxmPrimitiveType;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_PRIMITIVE_TRIANGLES	0x00000000U	Indexed triangle list.
SCE_GXM_PRIMITIVE_LINES	0x04000000U	Indexed line list.
SCE_GXM_PRIMITIVE_POINTS	0x08000000U	Indexed point list.
SCE_GXM_PRIMITIVE_TRIANGLE_STRIP	0x0c000000U	Indexed triangle strip.
SCE_GXM_PRIMITIVE_TRIANGLE_FAN	0x10000000U	Indexed triangle fan.
SCE_GXM_PRIMITIVE_TRIANGLE_EDGES	0x14000000U	Indexed triangle edge list.

Description

The primitive types.

SceGxmRegionClipMode

The tile level clipping modes.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmRegionClipMode {
    SCE_GXM_REGION_CLIP_NONE = 0x00000000U,
    SCE_GXM_REGION_CLIP_OUTSIDE = 0x80000000U
} SceGxmRegionClipMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_REGION_CLIP_NONE`	`0x00000000U`	No tiles are clipped.
`SCE_GXM_REGION_CLIP_OUTSIDE`	`0x80000000U`	Tiles outside the region are clipped.

Description

The tile level clipping modes.

SceGxmRenderTarget

The opaque structure for render targets.

Definition

```
#include <gxm/structs.h>
typedef struct SceGxmRenderTarget;
```

Description

The opaque structure for render targets.

SceGxmRenderTargetFlags

Initialization flags for render target creation.

Definition

```
#include <gxm/render_target.h>
typedef enum SceGxmRenderTargetFlags {
    SCE_GXM_RENDER_TARGET_CUSTOM_MULTISAMPLE_LOCATIONS = 0x00000001U,
    SCE_GXM_RENDER_TARGET_MACROTILE_SYNC = 0x00000002U,
    SCE_GXM_RENDER_TARGET_USE_DISPLAY_QUEUE_PARAMS = 0x00000010U
} SceGxmRenderTargetFlags;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_RENDER_TARGET_CUSTOM_MULTISAMPLE_LOCATIONS`	`0x00000001U`	Set custom multisample locations.
`SCE_GXM_RENDER_TARGET_MACROTILE_SYNC`	`0x00000002U`	Synchronize between macrotiles.
`SCE_GXM_RENDER_TARGET_USE_DISPLAY_QUEUE_PARAMS`	`0x00000010U`	Use the display queue parameters to tune CPU->GPU pipelining.

Description

Initialization flags for render target creation. See the documentation for `SceGxmRenderTargetParams` for details of each flag.

SceGxmRenderTargetParams

The initialization parameters for a render target.

Definition

```
#include <gxm/render_target.h>
typedef struct SceGxmRenderTargetParams {
    uint32_t flags;
    uint16_t width;
    uint16_t height;
    uint16_t scenesPerFrame;
    uint16_t multisampleMode;
    uint32_t multisampleLocations;
    SceUID driverMemBlock;
} SceGxmRenderTargetParams;
```

Members

flags	Bitwise combined flags from `SceGxmRenderTargetFlags`.
width	The width of the render target in pixels.
height	The height of the render target in pixels.
scenesPerFrame	The expected number of scenes per frame, in the range `[1,SCE_GXM_MAX_SCENES_PER_RENDERTARGET]`.
multisampleMode	A value from the `SceGxmMultisampleMode` enum.
multisampleLocations	If enabled in the flags, the multisample locations to use.
driverMemBlock	The uncached LPDDR memblock for the render target GPU data structures or `SCE_UID_INVALID_UID` to specify memory should be allocated in libgxm.

Description

The initialization parameters for a render target.

If `SCE_GXM_RENDER_TARGET_CUSTOM_MULTISAMPLE_LOCATIONS` is specified as part of the flags, the multisample locations may be overridden. These are specified as 4-bit x and y coordinates on a 16x16 grid over each pixel. These coordinates should be written to the *multisampleLocations* field from the low nybble to the high nybble.

For `SCE_GXM_MULTISAMPLE_NONE`, only the low 8 bits are used.

If `SCE_GXM_RENDER_TARGET_CUSTOM_MULTISAMPLE_LOCATIONS` is not set, the default value is `0x00000088` for the centre pixel at `(8,8)`.

For `SCE_GXM_MULTISAMPLE_2X`, only the low 16 bits are used.

If `SCE_GXM_RENDER_TARGET_CUSTOM_MULTISAMPLE_LOCATIONS` is not set, the default value is `0x0000cc44` for a diagonal 2-sample pattern of `(4,4), (12,12)`. A regular grid pattern would be `0x0000c848`.

For `SCE_GXM_MULTISAMPLE_4X`, the full word is used.

If `SCE_GXM_RENDER_TARGET_CUSTOM_MULTISAMPLE_LOCATIONS` is not set, the default value is `0xeaa26e26` for a 4-sample rotated grid pattern of `(6,2), (14,6), (2,10), (10,14)`. A regular grid pattern would be `0xccc44c44`.

If `SCE_GXM_RENDER_TARGET_MACROTILE_SYNC` is specified as part of the flags, the GPU firmware will synchronize between each macrotile of the render target. The idea of this feature is to allow color surface data from previous macrotiles to be safely read within the current scene. This avoids expensive scene changes in the GPU firmware. A macrotile is a rectangular group of tiles which must be a multiple of 4 tiles in width and height. A render target consists of a grid of up to 16 identical macrotiles. The macrotile synchronization feature is intended to be used to efficiently perform low-resolution, multi-pass, post-processing operations within a single scene. Instead of implementing post-processing using a separate render target for each stage, this feature allows each macrotile of a single render target to be used as a separate stage of the process, with all rendering performed in a single scene. Although this approach has a much lower firmware overhead than using separate scenes there is a small cost to each synchronization point. This means that the `SCE_GXM_RENDER_TARGET_MACROTILE_SYNC` flag should only be used if the synchronization points are actually required.

When using the `SCE_GXM_RENDER_TARGET_MACROTILE_SYNC` flag, the number of macrotiles in the X and Y direction must be provided as follows:

```
flags   = SCE_GXM_RENDER_TARGET_MACROTILE_SYNC | (
countX << SCE_GXM_RENDER_TARGET_MACROTILE_COUNT_X_SHIFT)| (
countY << SCE_GXM_RENDER_TARGET_MACROTILE_COUNT_Y_SHIFT);
```

The values of `countX` and `countY` must be between 1 and 4, but they cannot both be 1 simultaneously. The size of each macrotile is inferred by dividing the width and height of the render target by these count values. The width and height of each macrotile must be a multiple of 128 pixels and cannot be larger than 1024 pixels. MSAA cannot be used when using macrotile synchronization, and the macrotile counts cannot be specified unless the macrotile synchronization flag is also specified.

When macrotile synchronization is used, the color surface data from previous macrotiles may be safely read through the texture unit while shading the current macrotile. Macrotiles are rendered in vertical scanline order through the render target. For example, when `countX` and `countY` both have value 4, the order is as follows:

```
1       5       9       13
2       6       10      14
3       7       11      15
4       8       12      16
```

The macrotile order when `countX` is 3 and `countY` is 2 is as follows:

```
1       3       5
2       4       6
```

Synchronization is not provided within each macrotile; tiles within a single macrotile will complete in an arbitrary order after being shaded by any GPU core. Because of hardware restrictions, internal render target memory is allocated for either the 2x2 mode or 4x4 mode. This means macrotile configurations that are larger than 2x2 all consume the memory of a 4x4 configuration. However, macrotiles outside of `countX` and `countY` are skipped by the GPU firmware, and these values should be chosen in order to minimize the number of unused macrotiles. To reduce the render target memory footprint, it is recommended that the macrotile mode be 2x2 when 4 or fewer macrotiles are required. For example, 2x2 should be used in preference to 1x4.

In order to use color surface data safely from previous macrotiles, the following rules must be adhered to:

- Color surface data for each macrotile must lie in separate 64-byte system level cache (SLC) lines. A simple example that adheres to this rule is a linear color surface with a start address that is aligned to a 64-byte boundary. Because all macrotiles are a multiple of 128 pixels in width and height, it is sufficient to use a linear color surface with a start address that is aligned to a 64 byte boundary. Care must be taken to ensure that texture filtering does not load cache lines from other macrotiles. It is sufficient to alias each macrotile as a separate `SceGxmTexture` with a CLAMP address mode to ensure only that macrotile's texels are loaded.
- All primitives that lie in macrotiles 1 to (N - 1) must be drawn before any primitives that lie in macrotile N are drawn. This ensures that split scenes, due either to ring buffer pressure or partial rendering, do not change the visual results.

Note that macrotile synchronization only makes the color surface data from previous macrotiles available to be read. In particular, it does not perform synchronization of depth/stencil values. Reading of the depth/stencil data from previous macrotiles should not be performed because there is a chance that tile data will be read before the depth/stencil writes have completed.

A custom valid region cannot be used when rendering a scene using a render target that uses macrotile synchronization. This restriction may be lifted in a future SDK.

SceGxmSceneFlags

Scene flags passed to sceGxmBeginScene().

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmSceneFlags {
    SCE_GXM_SCENE_FRAGMENT_SET_DEPENDENCY = 0x00000001U,
    SCE_GXM_SCENE_VERTEX_WAIT_FOR_DEPENDENCY = 0x00000002U,
    SCE_GXM_SCENE_FRAGMENT_TRANSFER_SYNC = 0x00000004U,
    SCE_GXM_SCENE_VERTEX_TRANSFER_SYNC = 0x00000008U
} SceGxmSceneFlags;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_SCENE_FRAGMENT_SET_DEPENDENCY	0x00000001U	The fragment processing of this scene is used as a dependency for the vertex processing of the next scene to set SCE_GXM_SCENE_VERTEX_WAIT_FOR_DEPENDENCY.
SCE_GXM_SCENE_VERTEX_WAIT_FOR_DEPENDENCY	0x00000002U	Do not start the vertex processing for this scene until the fragment processing for the last scene to set SCE_GXM_SCENE_FRAGMENT_SET_DEPENDENCY is complete.
SCE_GXM_SCENE_FRAGMENT_TRANSFER_SYNC	0x00000004U	The fragment processing is strongly ordered with transfers that have SCE_GXM_TRANSFER_FRAGMENT_SYNC set.
SCE_GXM_SCENE_VERTEX_TRANSFER_SYNC	0x00000008U	The vertex processing is strongly ordered with transfers that have SCE_GXM_TRANSFER_VERTEX_SYNC set.

Description

Scene flags passed to sceGxmBeginScene(). Note that it is valid to set both SCE_GXM_SCENE_FRAGMENT_SET_DEPENDENCY and SCE_GXM_SCENE_VERTEX_WAIT_FOR_DEPENDENCY. The scene does not end up depending on itself because the vertex wait flag is processed first.

SceGxmStencilFunc

The stencil compare functions.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmStencilFunc {
    SCE_GXM_STENCIL_FUNC_NEVER = 0x00000000U,
    SCE_GXM_STENCIL_FUNC_LESS = 0x02000000U,
    SCE_GXM_STENCIL_FUNC_EQUAL = 0x04000000U,
    SCE_GXM_STENCIL_FUNC_LESS_EQUAL = 0x06000000U,
    SCE_GXM_STENCIL_FUNC_GREATER = 0x08000000U,
    SCE_GXM_STENCIL_FUNC_NOT_EQUAL = 0x0a000000U,
    SCE_GXM_STENCIL_FUNC_GREATER_EQUAL = 0x0c000000U,
    SCE_GXM_STENCIL_FUNC_ALWAYS = 0x0e000000U
} SceGxmStencilFunc;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_STENCIL_FUNC_NEVER	0x00000000U	Never pass.
SCE_GXM_STENCIL_FUNC_LESS	0x02000000U	Pass when (reference & mask) is less than (stencil & mask).
SCE_GXM_STENCIL_FUNC_EQUAL	0x04000000U	Pass when (reference & mask) is equal to (stencil & mask).
SCE_GXM_STENCIL_FUNC_LESS_EQUAL	0x06000000U	Pass when (reference & mask) is less than or equal to (stencil & mask).
SCE_GXM_STENCIL_FUNC_GREATER	0x08000000U	Pass when (reference & mask) is greater than (stencil & mask).
SCE_GXM_STENCIL_FUNC_NOT_EQUAL	0x0a000000U	Pass when (reference & mask) is not equal to (stencil & mask).
SCE_GXM_STENCIL_FUNC_GREATER_EQUAL	0x0c000000U	Pass when (reference & mask) is greater than or equal to (stencil & mask).
SCE_GXM_STENCIL_FUNC_ALWAYS	0x0e000000U	Always pass.

Description

The stencil compare functions. When a mask update fragment program is used, stencil testing is bypassed and bit 25 of these values specifies the mask update behaviour. The mask bit is cleared when bit 25 is 0 and set when bit 25 is 1.

To ensure that stencil data can be preserved during partial render, when the current scene is using a depth/stencil surface that does not contain stencil data as part of the format, only the stencil functions SCE_GXM_STENCIL_FUNC_NEVER and SCE_GXM_STENCIL_FUNC_ALWAYS may be used. These stencil modes should be used to clear or set the mask bit when drawing geometry using a mask update fragment program, since they are both allowed when stencil data is not present.

See sceGxmShaderPatcherCreateMaskUpdateFragmentProgram() for more information about the mask bit.

SceGxmStencilOp

The stencil operations.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmStencilOp {
    SCE_GXM_STENCIL_OP_KEEP = 0x00000000U,
    SCE_GXM_STENCIL_OP_ZERO = 0x00000001U,
    SCE_GXM_STENCIL_OP_REPLACE = 0x00000002U,
    SCE_GXM_STENCIL_OP_INCR = 0x00000003U,
    SCE_GXM_STENCIL_OP_DECR = 0x00000004U,
    SCE_GXM_STENCIL_OP_INVERT = 0x00000005U,
    SCE_GXM_STENCIL_OP_INCR_WRAP = 0x00000006U,
    SCE_GXM_STENCIL_OP_DECR_WRAP = 0x00000007U
} SceGxmStencilOp;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_STENCIL_OP_KEEP	0x00000000U	Keep the current stencil buffer value.
SCE_GXM_STENCIL_OP_ZERO	0x00000001U	Set the stencil buffer value to 0.
SCE_GXM_STENCIL_OP_REPLACE	0x00000002U	Replace the stencil buffer value with the stencil ref value.
SCE_GXM_STENCIL_OP_INCR	0x00000003U	Increment the stencil buffer value, clamp to 255.
SCE_GXM_STENCIL_OP_DECR	0x00000004U	Decrement the stencil buffer value, clamp to 0.
SCE_GXM_STENCIL_OP_INVERT	0x00000005U	Bitwise invert the stencil buffer value.
SCE_GXM_STENCIL_OP_INCR_WRAP	0x00000006U	Increment the stencil buffer value, wrap from 255 to 0.
SCE_GXM_STENCIL_OP_DECR_WRAP	0x00000007U	Decrement the stencil buffer value, wrap from 0 to 255.

Description

The stencil operations.

SceGxmSyncObject

The opaque structure for sync objects.

Definition

```
#include <gxm/structs.h>
typedef struct SceGxmSyncObject;
```

Description

The opaque structure for sync objects.

SceGxmTexture

Direct representation of texture control words.

Definition

```
#include <gxm/structs.h>
typedef struct SceGxmTexture {
    uint32_t controlWords[SCE_GXM_NUM_TEXTURE_CONTROL_WORDS];
} SceGxmTexture;
```

Members

controlWords	Texture control words.

Description

Direct representation of texture control words. Textures are fully described by the four 32-bit words of data associated with a PDS DOUTT instruction. They can be defined by using the libgxm texture API or by using the defines in `gxm\texture_defs.h`. Please see the *GPU User's Guide* for more details.

Texture control words can be validated in libgxm using `sceGxmTextureValidate()`.

SceGxmTextureAddrMode

The texture addressing mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureAddrMode {
    SCE_GXM_TEXTURE_ADDR_REPEAT = 0x00000000U,
    SCE_GXM_TEXTURE_ADDR_MIRROR = 0x00000001U,
    SCE_GXM_TEXTURE_ADDR_CLAMP = 0x00000002U,
    SCE_GXM_TEXTURE_ADDR_MIRROR_CLAMP = 0x00000003U,
    SCE_GXM_TEXTURE_ADDR_REPEAT_IGNORE_BORDER = 0x00000004U,
    SCE_GXM_TEXTURE_ADDR_CLAMP_FULL_BORDER = 0x00000005U,
    SCE_GXM_TEXTURE_ADDR_CLAMP_IGNORE_BORDER = 0x00000006U,
    SCE_GXM_TEXTURE_ADDR_CLAMP_HALF_BORDER = 0x00000007U
} SceGxmTextureAddrMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TEXTURE_ADDR_REPEAT`	`0x00000000U`	Addresses the texture using the fractional part of the uv. This results in the texture being repeated along this axis.
`SCE_GXM_TEXTURE_ADDR_MIRROR`	`0x00000001U`	Alternates between using f and (1 - f), where f is the fractional part of the uv. This results in the texture alternating between flipped and non-flipped as it repeats in this axis.
`SCE_GXM_TEXTURE_ADDR_CLAMP`	`0x00000002U`	Clamps the uv to [0, 1] and does not filter across texture edges when filtering is enabled. This produces a clamped-to-edge result.
`SCE_GXM_TEXTURE_ADDR_MIRROR_CLAMP`	`0x00000003U`	Clamps the absolute value of the uv to [0, 1] and does not filter across texture edges when filtering is enabled. This results in a texture that is mirrored once around 0.
`SCE_GXM_TEXTURE_ADDR_REPEAT_IGNORE_BORDER`	`0x00000004U`	Addresses the texture using the fractional part of the uv. This results in the texture being repeated along this axis. Border texels are ignored but must be present in the texture data.

Macro	Value	Description
SCE_GXM_TEXTURE_ADDR_CLAMP_FULL_BORDER	0x00000005U	Uses border texels for data outside of [0, 1]. Since the address is not clamped, if uv is outside of [0, 1], the filtered result can be 100% border texels.
SCE_GXM_TEXTURE_ADDR_CLAMP_IGNORE_BORDER	0x00000006U	Clamps the uv to [0, 1] and does not filter across texture edges when filtering is enabled. This produces a clamped-to-edge result. Border texels are ignored but must be present in the texture data.
SCE_GXM_TEXTURE_ADDR_CLAMP_HALF_BORDER	0x00000007U	Clamps the uv to [0, 1] and uses border texels for data outside of [0, 1] when filtering is enabled. Since the address is clamped, the filtered result can be at most 50% border texels (in this axis).

Description

The texture addressing mode. The first four modes (SCE_GXM_TEXTURE_ADDR_REPEAT, SCE_GXM_TEXTURE_ADDR_MIRROR, SCE_GXM_TEXTURE_ADDR_CLAMP, and SCE_GXM_TEXTURE_ADDR_MIRROR_CLAMP) are only for textures that do not provide border data. They cannot be used if border data is part of the texture data.

The second four modes (SCE_GXM_TEXTURE_ADDR_REPEAT_IGNORE_BORDER, SCE_GXM_TEXTURE_ADDR_CLAMP_IGNORE_BORDER, SCE_GXM_TEXTURE_ADDR_CLAMP_FULL_BORDER and SCE_GXM_TEXTURE_ADDR_CLAMP_HALF_BORDER) are only for textures of type SCE_GXM_TEXTURE_SWIZZLED or SCE_GXM_TEXTURE_SWIZZLED_ARBITRARY that provide border data. They cannot be used if border data is not part of the texture data or if the texture is not of type SCE_GXM_TEXTURE_SWIZZLED or SCE_GXM_TEXTURE_SWIZZLED_ARBITRARY.

The results of address modes SCE_GXM_TEXTURE_ADDR_REPEAT and SCE_GXM_TEXTURE_ADDR_REPEAT_IGNORE_BORDER are identical. The first mode is for textures where border data **is not** present, and the second is for textures where border data **is** present. Similarly, the results of address modes SCE_GXM_TEXTURE_ADDR_CLAMP and SCE_GXM_TEXTURE_ADDR_CLAMP_IGNORE_BORDER are identical, but the second mode is for textures with border data (even though the data is ignored).

More detail on texture addressing modes can be found in the *GPU User's Guide*.

SceGxmTextureBaseFormat

The base formats for textures.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureBaseFormat {
    SCE_GXM_TEXTURE_BASE_FORMAT_U8 = 0x00000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_S8 = 0x01000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 = 0x02000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U8U3U3U2 = 0x03000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U1U5U5U5 = 0x04000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U5U6U5 = 0x05000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_S5S5U6 = 0x06000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U8U8 = 0x07000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_S8S8 = 0x08000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U16 = 0x09000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_S16 = 0x0a000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_F16 = 0x0b000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8U8 = 0x0c000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 = 0x0d000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 = 0x0e000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U16U16 = 0x0f000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_S16S16 = 0x10000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_F16F16 = 0x11000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_F32 = 0x12000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_F32M = 0x13000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_X8S8S8U8 = 0x14000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_X8U24 = 0x15000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U32 = 0x17000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_S32 = 0x18000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_SE5M9M9M9 = 0x19000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_F11F11F10 = 0x1a000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16 = 0x1b000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 = 0x1c000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 = 0x1d000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_F32F32 = 0x1e000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U32U32 = 0x1f000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_PVRT2BPP = 0x80000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_PVRT4BPP = 0x81000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_PVRTII2BPP = 0x82000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_PVRTII4BPP = 0x83000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_UBC1 = 0x85000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_UBC2 = 0x86000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_UBC3 = 0x87000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_UBC4 = 0x88000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_SBC4 = 0x89000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_UBC5 = 0x8A000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_SBC5 = 0x8B000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P2 = 0x90000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P3 = 0x91000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 = 0x92000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_P4 = 0x94000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_P8 = 0x95000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8 = 0x98000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8 = 0x99000000U,
    SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 = 0x9a000000U
} SceGxmTextureBaseFormat;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TEXTURE_BASE_FORMAT_U8`	`0x00000000U`	8-bit format, 8-bit unsigned integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_S8`	`0x01000000U`	8-bit format, 8-bit signed integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4`	`0x02000000U`	16-bit format, 4x 4-bit unsigned integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_U8U3U3U2`	`0x03000000U`	16-bit format, 8-bit unsigned, 3-bit unsigned, 3-bit unsigned and 2-bit unsigned integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_U1U5U5U5`	`0x04000000U`	16-bit format, 1-bit unsigned and 3x 5-bit unsigned integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_U5U6U5`	`0x05000000U`	16-bit format, 5-bit unsigned, 6-bit unsigned and 5-bit unsigned integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_S5S5U6`	`0x06000000U`	16-bit format, 5-bit signed, 5-bit signed and 6-bit unsigned integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_U8U8`	`0x07000000U`	16-bit format, 2x 8-bit unsigned integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_S8S8`	`0x08000000U`	16-bit format, 2x 8-bit signed integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_U16`	`0x09000000U`	16-bit format, 16-bit unsigned integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_S16`	`0x0a000000U`	16-bit format, 16-bit signed integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_F16`	`0x0b000000U`	16-bit format, 16-bit s1e5m10 floating point.
`SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8U8`	`0x0c000000U`	32-bit format, 4x 8-bit unsigned integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8`	`0x0d000000U`	32-bit format, 4x 8-bit signed integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10`	`0x0e000000U`	32-bit format, 2-bit unsigned and 3x 10-bit unsigned integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_U16U16`	`0x0f000000U`	32-bit format, 2x 16-bit unsigned integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_S16S16`	`0x10000000U`	32-bit format, 2x 16-bit signed integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_F16F16`	`0x11000000U`	32-bit format, 2x 16-bit s1e5m10 floating point.
`SCE_GXM_TEXTURE_BASE_FORMAT_F32`	`0x12000000U`	32-bit format, 32-bit floating point.
`SCE_GXM_TEXTURE_BASE_FORMAT_F32M`	`0x13000000U`	32-bit format, 32-bit floating point with sign bit masked off.
`SCE_GXM_TEXTURE_BASE_FORMAT_X8S8S8U8`	`0x14000000U`	32-bit format, 8-bit unused, 8-bit signed, 8-bit signed and 8-bit unsigned integer.
`SCE_GXM_TEXTURE_BASE_FORMAT_X8U24`	`0x15000000U`	32-bit format, 8-bit unused, 24-bit unsigned integer.

Macro	Value	Description
SCE_GXM_TEXTURE_BASE_FORMAT_U32	0x17000000U	32-bit format, 32-bit unsigned integer.
SCE_GXM_TEXTURE_BASE_FORMAT_S32	0x18000000U	32-bit format, 32-bit signed integer.
SCE_GXM_TEXTURE_BASE_FORMAT_SE5M9M9M9	0x19000000U	32-bit format, 5-bit shared exponent and 3x 9-bit floating point mantissa.
SCE_GXM_TEXTURE_BASE_FORMAT_F11F11F10	0x1a000000U	32-bit format, 2x 11-bit s0e5m6 floating point and 10-bit s0e5m5 floating point.
SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16	0x1b000000U	64-bit format, 4x 16-bit s1e5m10 floating point.
SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16	0x1c000000U	64-bit format, 4x 16-bit unsigned integer.
SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16	0x1d000000U	64-bit format, 4x 16-bit signed integer.
SCE_GXM_TEXTURE_BASE_FORMAT_F32F32	0x1e000000U	64-bit format, 2x 32-bit floating point.
SCE_GXM_TEXTURE_BASE_FORMAT_U32U32	0x1f000000U	64-bit format, 2x 32-bit unsigned integer.
SCE_GXM_TEXTURE_BASE_FORMAT_PVRT2BPP	0x80000000U	Block compressed PVRT1, 2 bits per pixel mode.
SCE_GXM_TEXTURE_BASE_FORMAT_PVRT4BPP	0x81000000U	Block compressed PVRT1, 4 bits per pixel mode.
SCE_GXM_TEXTURE_BASE_FORMAT_PVRTII2BPP	0x82000000U	Block compressed PVRT2, 2 bits per pixel mode.
SCE_GXM_TEXTURE_BASE_FORMAT_PVRTII4BPP	0x83000000U	Block compressed PVRT2, 4 bits per pixel mode.
SCE_GXM_TEXTURE_BASE_FORMAT_UBC1	0x85000000U	Block compressed UBC1 (aka DXT1), 4 bits per pixel.
SCE_GXM_TEXTURE_BASE_FORMAT_UBC2	0x86000000U	Block compressed UBC2 (aka DXT3), 8 bits per pixel.
SCE_GXM_TEXTURE_BASE_FORMAT_UBC3	0x87000000U	Block compressed UBC3 (aka DXT5), 8 bits per pixel.
SCE_GXM_TEXTURE_BASE_FORMAT_UBC4	0x88000000U	Block compressed UBC4, 4 bits per pixel.
SCE_GXM_TEXTURE_BASE_FORMAT_SBC4	0x89000000U	Block compressed SBC4, 4 bits per pixel.
SCE_GXM_TEXTURE_BASE_FORMAT_UBC5	0x8A000000U	Block compressed UBC5, 8 bits per pixel.
SCE_GXM_TEXTURE_BASE_FORMAT_SBC5	0x8B000000U	Block compressed SBC5, 8 bits per pixel.
SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P2	0x90000000U	Y plane, interleaved UV plane.
SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P3	0x91000000U	Y plane, U plane, V plane.
SCE_GXM_TEXTURE_BASE_FORMAT_YUV422	0x92000000U	Interleaved YUV.
SCE_GXM_TEXTURE_BASE_FORMAT_P4	0x94000000U	Palettized format, 4-bit palette index per pixel.
SCE_GXM_TEXTURE_BASE_FORMAT_P8	0x95000000U	Palettized format, 8-bit palette index per pixel.
SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8	0x98000000U	24-bit packed format, 3x 8-bit unsigned integer.
SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8	0x99000000U	24-bit packed format, 3x 8-bit signed integer.

Macro	Value	Description
`SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10`	`0x9a000000U`	32-bit format, 2-bit unsigned integer and 3x 10-bit s0e5m5 floating point.

Description

The base formats for textures. A texture format is made from (bitwise) combining a base format with a compatible swizzle.

SceGxmTextureFilter

The texture filter mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureFilter {
    SCE_GXM_TEXTURE_FILTER_POINT = 0x00000000U,
    SCE_GXM_TEXTURE_FILTER_LINEAR = 0x00000001U
} SceGxmTextureFilter;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TEXTURE_FILTER_POINT`	`0x00000000U`	Point sampling.
`SCE_GXM_TEXTURE_FILTER_LINEAR`	`0x00000001U`	Linear filtering.

Description

The texture filter mode.

SceGxmTextureFormat

The texture formats.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureFormat {
    SCE_GXM_TEXTURE_FORMAT_U8_000R = SCE_GXM_TEXTURE_BASE_FORMAT_U8 |
    SCE_GXM_TEXTURE_SWIZZLE1_000R,
    SCE_GXM_TEXTURE_FORMAT_U8_111R = SCE_GXM_TEXTURE_BASE_FORMAT_U8 |
    SCE_GXM_TEXTURE_SWIZZLE1_111R,
    SCE_GXM_TEXTURE_FORMAT_U8_RRRR = SCE_GXM_TEXTURE_BASE_FORMAT_U8 |
    SCE_GXM_TEXTURE_SWIZZLE1_RRRR,
    SCE_GXM_TEXTURE_FORMAT_U8_0RRR = SCE_GXM_TEXTURE_BASE_FORMAT_U8 |
    SCE_GXM_TEXTURE_SWIZZLE1_0RRR,
    SCE_GXM_TEXTURE_FORMAT_U8_1RRR = SCE_GXM_TEXTURE_BASE_FORMAT_U8 |
    SCE_GXM_TEXTURE_SWIZZLE1_1RRR,
    SCE_GXM_TEXTURE_FORMAT_U8_R000 = SCE_GXM_TEXTURE_BASE_FORMAT_U8 |
    SCE_GXM_TEXTURE_SWIZZLE1_R000,
    SCE_GXM_TEXTURE_FORMAT_U8_R111 = SCE_GXM_TEXTURE_BASE_FORMAT_U8 |
    SCE_GXM_TEXTURE_SWIZZLE1_R111,
    SCE_GXM_TEXTURE_FORMAT_U8_R = SCE_GXM_TEXTURE_BASE_FORMAT_U8 |
    SCE_GXM_TEXTURE_SWIZZLE1_R,
    SCE_GXM_TEXTURE_FORMAT_S8_000R = SCE_GXM_TEXTURE_BASE_FORMAT_S8 |
    SCE_GXM_TEXTURE_SWIZZLE1_000R,
    SCE_GXM_TEXTURE_FORMAT_S8_111R = SCE_GXM_TEXTURE_BASE_FORMAT_S8 |
    SCE_GXM_TEXTURE_SWIZZLE1_111R,
    SCE_GXM_TEXTURE_FORMAT_S8_RRRR = SCE_GXM_TEXTURE_BASE_FORMAT_S8 |
    SCE_GXM_TEXTURE_SWIZZLE1_RRRR,
    SCE_GXM_TEXTURE_FORMAT_S8_0RRR = SCE_GXM_TEXTURE_BASE_FORMAT_S8 |
    SCE_GXM_TEXTURE_SWIZZLE1_0RRR,
    SCE_GXM_TEXTURE_FORMAT_S8_1RRR = SCE_GXM_TEXTURE_BASE_FORMAT_S8 |
    SCE_GXM_TEXTURE_SWIZZLE1_1RRR,
    SCE_GXM_TEXTURE_FORMAT_S8_R000 = SCE_GXM_TEXTURE_BASE_FORMAT_S8 |
    SCE_GXM_TEXTURE_SWIZZLE1_R000,
    SCE_GXM_TEXTURE_FORMAT_S8_R111 = SCE_GXM_TEXTURE_BASE_FORMAT_S8 |
    SCE_GXM_TEXTURE_SWIZZLE1_R111,
    SCE_GXM_TEXTURE_FORMAT_S8_R = SCE_GXM_TEXTURE_BASE_FORMAT_S8 |
    SCE_GXM_TEXTURE_SWIZZLE1_R,
    SCE_GXM_TEXTURE_FORMAT_U4U4U4U4_ABGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_U4U4U4U4_ARGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB,
    SCE_GXM_TEXTURE_FORMAT_U4U4U4U4_RGBA =
    SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA,
    SCE_GXM_TEXTURE_FORMAT_U4U4U4U4_BGRA =
    SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA,
    SCE_GXM_TEXTURE_FORMAT_X4U4U4U4_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_X4U4U4U4_1RGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB,
    SCE_GXM_TEXTURE_FORMAT_U4U4U4X4_RGB1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1,
    SCE_GXM_TEXTURE_FORMAT_U4U4U4X4_BGR1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1,
    SCE_GXM_TEXTURE_FORMAT_U8U3U3U2_ARGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_U8U3U3U2,
    SCE_GXM_TEXTURE_FORMAT_U1U5U5U5_ABGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_U1U5U5U5 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_U1U5U5U5_ARGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_U1U5U5U5 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB,
    SCE_GXM_TEXTURE_FORMAT_U5U5U5U1_RGBA =
    SCE_GXM_TEXTURE_BASE_FORMAT_U1U5U5U5 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA,
    SCE_GXM_TEXTURE_FORMAT_U5U5U5U1_BGRA =
    SCE_GXM_TEXTURE_BASE_FORMAT_U1U5U5U5 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA,
    SCE_GXM_TEXTURE_FORMAT_X1U5U5U5_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_U1U5U5U5 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_X1U5U5U5_1RGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_U1U5U5U5 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB,
    SCE_GXM_TEXTURE_FORMAT_U5U5U5X1_RGB1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_U1U5U5U5 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1,
    SCE_GXM_TEXTURE_FORMAT_U5U5U5X1_BGR1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_U1U5U5U5 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1,
    SCE_GXM_TEXTURE_FORMAT_U5U6U5_BGR = SCE_GXM_TEXTURE_BASE_FORMAT_U5U6U5 |
    SCE_GXM_TEXTURE_SWIZZLE3_BGR,
    SCE_GXM_TEXTURE_FORMAT_U5U6U5_RGB = SCE_GXM_TEXTURE_BASE_FORMAT_U5U6U5 |
    SCE_GXM_TEXTURE_SWIZZLE3_RGB,
    SCE_GXM_TEXTURE_FORMAT_U6S5S5_BGR = SCE_GXM_TEXTURE_BASE_FORMAT_S5S5U6 |
    SCE_GXM_TEXTURE_SWIZZLE3_BGR,
    SCE_GXM_TEXTURE_FORMAT_S5S5U6_RGB = SCE_GXM_TEXTURE_BASE_FORMAT_S5S5U6 |
    SCE_GXM_TEXTURE_SWIZZLE3_RGB,
    SCE_GXM_TEXTURE_FORMAT_U8U8_00GR = SCE_GXM_TEXTURE_BASE_FORMAT_U8U8 |
    SCE_GXM_TEXTURE_SWIZZLE2_00GR,
    SCE_GXM_TEXTURE_FORMAT_U8U8_GRRR = SCE_GXM_TEXTURE_BASE_FORMAT_U8U8 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRRR,
    SCE_GXM_TEXTURE_FORMAT_U8U8_RGGG = SCE_GXM_TEXTURE_BASE_FORMAT_U8U8 |
    SCE_GXM_TEXTURE_SWIZZLE2_RGGG,
    SCE_GXM_TEXTURE_FORMAT_U8U8_GRGR = SCE_GXM_TEXTURE_BASE_FORMAT_U8U8 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRGR,
    SCE_GXM_TEXTURE_FORMAT_U8U8_00RG = SCE_GXM_TEXTURE_BASE_FORMAT_U8U8 |
    SCE_GXM_TEXTURE_SWIZZLE2_00RG,
    SCE_GXM_TEXTURE_FORMAT_U8U8_GR = SCE_GXM_TEXTURE_BASE_FORMAT_U8U8 |
    SCE_GXM_TEXTURE_SWIZZLE2_GR,
    SCE_GXM_TEXTURE_FORMAT_S8S8_00GR = SCE_GXM_TEXTURE_BASE_FORMAT_S8S8 |
    SCE_GXM_TEXTURE_SWIZZLE2_00GR,
    SCE_GXM_TEXTURE_FORMAT_S8S8_GRRR = SCE_GXM_TEXTURE_BASE_FORMAT_S8S8 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRRR,
    SCE_GXM_TEXTURE_FORMAT_S8S8_RGGG = SCE_GXM_TEXTURE_BASE_FORMAT_S8S8 |
    SCE_GXM_TEXTURE_SWIZZLE2_RGGG,
    SCE_GXM_TEXTURE_FORMAT_S8S8_GRGR = SCE_GXM_TEXTURE_BASE_FORMAT_S8S8 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRGR,
    SCE_GXM_TEXTURE_FORMAT_S8S8_00RG = SCE_GXM_TEXTURE_BASE_FORMAT_S8S8 |
    SCE_GXM_TEXTURE_SWIZZLE2_00RG,
    SCE_GXM_TEXTURE_FORMAT_S8S8_GR = SCE_GXM_TEXTURE_BASE_FORMAT_S8S8 |
    SCE_GXM_TEXTURE_SWIZZLE2_GR,
    SCE_GXM_TEXTURE_FORMAT_U16_000R = SCE_GXM_TEXTURE_BASE_FORMAT_U16 |
    SCE_GXM_TEXTURE_SWIZZLE1_000R,
    SCE_GXM_TEXTURE_FORMAT_U16_111R = SCE_GXM_TEXTURE_BASE_FORMAT_U16 |
    SCE_GXM_TEXTURE_SWIZZLE1_111R,
    SCE_GXM_TEXTURE_FORMAT_U16_RRRR = SCE_GXM_TEXTURE_BASE_FORMAT_U16 |
    SCE_GXM_TEXTURE_SWIZZLE1_RRRR,
    SCE_GXM_TEXTURE_FORMAT_U16_0RRR = SCE_GXM_TEXTURE_BASE_FORMAT_U16 |
    SCE_GXM_TEXTURE_SWIZZLE1_0RRR,
    SCE_GXM_TEXTURE_FORMAT_U16_1RRR = SCE_GXM_TEXTURE_BASE_FORMAT_U16 |
    SCE_GXM_TEXTURE_SWIZZLE1_1RRR,
    SCE_GXM_TEXTURE_FORMAT_U16_R000 = SCE_GXM_TEXTURE_BASE_FORMAT_U16 |
    SCE_GXM_TEXTURE_SWIZZLE1_R000,
    SCE_GXM_TEXTURE_FORMAT_U16_R111 = SCE_GXM_TEXTURE_BASE_FORMAT_U16 |
    SCE_GXM_TEXTURE_SWIZZLE1_R111,
    SCE_GXM_TEXTURE_FORMAT_U16_R = SCE_GXM_TEXTURE_BASE_FORMAT_U16 |
    SCE_GXM_TEXTURE_SWIZZLE1_R,
    SCE_GXM_TEXTURE_FORMAT_S16_000R = SCE_GXM_TEXTURE_BASE_FORMAT_S16 |
    SCE_GXM_TEXTURE_SWIZZLE1_000R,
    SCE_GXM_TEXTURE_FORMAT_S16_111R = SCE_GXM_TEXTURE_BASE_FORMAT_S16 |
    SCE_GXM_TEXTURE_SWIZZLE1_111R,
    SCE_GXM_TEXTURE_FORMAT_S16_RRRR = SCE_GXM_TEXTURE_BASE_FORMAT_S16 |
    SCE_GXM_TEXTURE_SWIZZLE1_RRRR,
    SCE_GXM_TEXTURE_FORMAT_S16_0RRR = SCE_GXM_TEXTURE_BASE_FORMAT_S16 |
    SCE_GXM_TEXTURE_SWIZZLE1_0RRR,
    SCE_GXM_TEXTURE_FORMAT_S16_1RRR = SCE_GXM_TEXTURE_BASE_FORMAT_S16 |
    SCE_GXM_TEXTURE_SWIZZLE1_1RRR,
    SCE_GXM_TEXTURE_FORMAT_S16_R000 = SCE_GXM_TEXTURE_BASE_FORMAT_S16 |
    SCE_GXM_TEXTURE_SWIZZLE1_R000,
    SCE_GXM_TEXTURE_FORMAT_S16_R111 = SCE_GXM_TEXTURE_BASE_FORMAT_S16 |
    SCE_GXM_TEXTURE_SWIZZLE1_R111,
    SCE_GXM_TEXTURE_FORMAT_S16_R = SCE_GXM_TEXTURE_BASE_FORMAT_S16 |
    SCE_GXM_TEXTURE_SWIZZLE1_R,
    SCE_GXM_TEXTURE_FORMAT_F16_000R = SCE_GXM_TEXTURE_BASE_FORMAT_F16 |
    SCE_GXM_TEXTURE_SWIZZLE1_000R,
    SCE_GXM_TEXTURE_FORMAT_F16_111R = SCE_GXM_TEXTURE_BASE_FORMAT_F16 |
    SCE_GXM_TEXTURE_SWIZZLE1_111R,
    SCE_GXM_TEXTURE_FORMAT_F16_RRRR = SCE_GXM_TEXTURE_BASE_FORMAT_F16 |
    SCE_GXM_TEXTURE_SWIZZLE1_RRRR,
    SCE_GXM_TEXTURE_FORMAT_F16_0RRR = SCE_GXM_TEXTURE_BASE_FORMAT_F16 |
    SCE_GXM_TEXTURE_SWIZZLE1_0RRR,
    SCE_GXM_TEXTURE_FORMAT_F16_1RRR = SCE_GXM_TEXTURE_BASE_FORMAT_F16 |
    SCE_GXM_TEXTURE_SWIZZLE1_1RRR,
    SCE_GXM_TEXTURE_FORMAT_F16_R000 = SCE_GXM_TEXTURE_BASE_FORMAT_F16 |
    SCE_GXM_TEXTURE_SWIZZLE1_R000,
    SCE_GXM_TEXTURE_FORMAT_F16_R111 = SCE_GXM_TEXTURE_BASE_FORMAT_F16 |
    SCE_GXM_TEXTURE_SWIZZLE1_R111,
    SCE_GXM_TEXTURE_FORMAT_F16_R = SCE_GXM_TEXTURE_BASE_FORMAT_F16 |
    SCE_GXM_TEXTURE_SWIZZLE1_R,
    SCE_GXM_TEXTURE_FORMAT_U8U8U8U8_ABGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8U8 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_U8U8U8U8_ARGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8U8 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB,
    SCE_GXM_TEXTURE_FORMAT_U8U8U8U8_RGBA =
    SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8U8 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA,
    SCE_GXM_TEXTURE_FORMAT_U8U8U8U8_BGRA =
    SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8U8 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA,
    SCE_GXM_TEXTURE_FORMAT_X8U8U8U8_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8U8 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_X8U8U8U8_1RGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8U8 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB,
    SCE_GXM_TEXTURE_FORMAT_U8U8U8X8_RGB1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8U8 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1,
    SCE_GXM_TEXTURE_FORMAT_U8U8U8X8_BGR1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8U8 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1,
    SCE_GXM_TEXTURE_FORMAT_S8S8S8S8_ABGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_S8S8S8S8_ARGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB,
    SCE_GXM_TEXTURE_FORMAT_S8S8S8S8_RGBA =
    SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA,
    SCE_GXM_TEXTURE_FORMAT_S8S8S8S8_BGRA =
    SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA,
    SCE_GXM_TEXTURE_FORMAT_X8S8S8S8_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_X8S8S8S8_1RGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB,
    SCE_GXM_TEXTURE_FORMAT_S8S8S8X8_RGB1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1,
    SCE_GXM_TEXTURE_FORMAT_S8S8S8X8_BGR1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1,
    SCE_GXM_TEXTURE_FORMAT_U2U10U10U10_ABGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_U2U10U10U10_ARGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB,
    SCE_GXM_TEXTURE_FORMAT_U10U10U10U2_RGBA =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA,
    SCE_GXM_TEXTURE_FORMAT_U10U10U10U2_BGRA =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA,
    SCE_GXM_TEXTURE_FORMAT_X2U10U10U10_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_X2U10U10U10_1RGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB,
    SCE_GXM_TEXTURE_FORMAT_U10U10U10X2_RGB1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1,
    SCE_GXM_TEXTURE_FORMAT_U10U10U10X2_BGR1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1,
    SCE_GXM_TEXTURE_FORMAT_U16U16_00GR = SCE_GXM_TEXTURE_BASE_FORMAT_U16U16 |
    SCE_GXM_TEXTURE_SWIZZLE2_00GR,
    SCE_GXM_TEXTURE_FORMAT_U16U16_GRRR = SCE_GXM_TEXTURE_BASE_FORMAT_U16U16 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRRR,
    SCE_GXM_TEXTURE_FORMAT_U16U16_RGGG = SCE_GXM_TEXTURE_BASE_FORMAT_U16U16 |
    SCE_GXM_TEXTURE_SWIZZLE2_RGGG,
    SCE_GXM_TEXTURE_FORMAT_U16U16_GRGR = SCE_GXM_TEXTURE_BASE_FORMAT_U16U16 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRGR,
    SCE_GXM_TEXTURE_FORMAT_U16U16_00RG = SCE_GXM_TEXTURE_BASE_FORMAT_U16U16 |
    SCE_GXM_TEXTURE_SWIZZLE2_00RG,
    SCE_GXM_TEXTURE_FORMAT_U16U16_GR = SCE_GXM_TEXTURE_BASE_FORMAT_U16U16 |
    SCE_GXM_TEXTURE_SWIZZLE2_GR,
    SCE_GXM_TEXTURE_FORMAT_S16S16_00GR = SCE_GXM_TEXTURE_BASE_FORMAT_S16S16 |
    SCE_GXM_TEXTURE_SWIZZLE2_00GR,
    SCE_GXM_TEXTURE_FORMAT_S16S16_GRRR = SCE_GXM_TEXTURE_BASE_FORMAT_S16S16 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRRR,
    SCE_GXM_TEXTURE_FORMAT_S16S16_RGGG = SCE_GXM_TEXTURE_BASE_FORMAT_S16S16 |
    SCE_GXM_TEXTURE_SWIZZLE2_RGGG,
    SCE_GXM_TEXTURE_FORMAT_S16S16_GRGR = SCE_GXM_TEXTURE_BASE_FORMAT_S16S16 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRGR,
    SCE_GXM_TEXTURE_FORMAT_S16S16_00RG = SCE_GXM_TEXTURE_BASE_FORMAT_S16S16 |
    SCE_GXM_TEXTURE_SWIZZLE2_00RG,
    SCE_GXM_TEXTURE_FORMAT_S16S16_GR = SCE_GXM_TEXTURE_BASE_FORMAT_S16S16 |
    SCE_GXM_TEXTURE_SWIZZLE2_GR,
    SCE_GXM_TEXTURE_FORMAT_F16F16_00GR = SCE_GXM_TEXTURE_BASE_FORMAT_F16F16 |
    SCE_GXM_TEXTURE_SWIZZLE2_00GR,
    SCE_GXM_TEXTURE_FORMAT_F16F16_GRRR = SCE_GXM_TEXTURE_BASE_FORMAT_F16F16 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRRR,
    SCE_GXM_TEXTURE_FORMAT_F16F16_RGGG = SCE_GXM_TEXTURE_BASE_FORMAT_F16F16 |
    SCE_GXM_TEXTURE_SWIZZLE2_RGGG,
    SCE_GXM_TEXTURE_FORMAT_F16F16_GRGR = SCE_GXM_TEXTURE_BASE_FORMAT_F16F16 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRGR,
    SCE_GXM_TEXTURE_FORMAT_F16F16_00RG = SCE_GXM_TEXTURE_BASE_FORMAT_F16F16 |
    SCE_GXM_TEXTURE_SWIZZLE2_00RG,
    SCE_GXM_TEXTURE_FORMAT_F16F16_GR = SCE_GXM_TEXTURE_BASE_FORMAT_F16F16 |
    SCE_GXM_TEXTURE_SWIZZLE2_GR,
    SCE_GXM_TEXTURE_FORMAT_F32_000R = SCE_GXM_TEXTURE_BASE_FORMAT_F32 |
    SCE_GXM_TEXTURE_SWIZZLE1_000R,
    SCE_GXM_TEXTURE_FORMAT_F32_111R = SCE_GXM_TEXTURE_BASE_FORMAT_F32 |
    SCE_GXM_TEXTURE_SWIZZLE1_111R,
    SCE_GXM_TEXTURE_FORMAT_F32_RRRR = SCE_GXM_TEXTURE_BASE_FORMAT_F32 |
    SCE_GXM_TEXTURE_SWIZZLE1_RRRR,
    SCE_GXM_TEXTURE_FORMAT_F32_0RRR = SCE_GXM_TEXTURE_BASE_FORMAT_F32 |
    SCE_GXM_TEXTURE_SWIZZLE1_0RRR,
    SCE_GXM_TEXTURE_FORMAT_F32_1RRR = SCE_GXM_TEXTURE_BASE_FORMAT_F32 |
    SCE_GXM_TEXTURE_SWIZZLE1_1RRR,
    SCE_GXM_TEXTURE_FORMAT_F32_R000 = SCE_GXM_TEXTURE_BASE_FORMAT_F32 |
    SCE_GXM_TEXTURE_SWIZZLE1_R000,
    SCE_GXM_TEXTURE_FORMAT_F32_R111 = SCE_GXM_TEXTURE_BASE_FORMAT_F32 |
    SCE_GXM_TEXTURE_SWIZZLE1_R111,
    SCE_GXM_TEXTURE_FORMAT_F32_R = SCE_GXM_TEXTURE_BASE_FORMAT_F32 |
    SCE_GXM_TEXTURE_SWIZZLE1_R,
    SCE_GXM_TEXTURE_FORMAT_F32M_000R = SCE_GXM_TEXTURE_BASE_FORMAT_F32M |
    SCE_GXM_TEXTURE_SWIZZLE1_000R,
    SCE_GXM_TEXTURE_FORMAT_F32M_111R = SCE_GXM_TEXTURE_BASE_FORMAT_F32M |
    SCE_GXM_TEXTURE_SWIZZLE1_111R,
    SCE_GXM_TEXTURE_FORMAT_F32M_RRRR = SCE_GXM_TEXTURE_BASE_FORMAT_F32M |
    SCE_GXM_TEXTURE_SWIZZLE1_RRRR,
    SCE_GXM_TEXTURE_FORMAT_F32M_0RRR = SCE_GXM_TEXTURE_BASE_FORMAT_F32M |
    SCE_GXM_TEXTURE_SWIZZLE1_0RRR,
    SCE_GXM_TEXTURE_FORMAT_F32M_1RRR = SCE_GXM_TEXTURE_BASE_FORMAT_F32M |
    SCE_GXM_TEXTURE_SWIZZLE1_1RRR,
    SCE_GXM_TEXTURE_FORMAT_F32M_R000 = SCE_GXM_TEXTURE_BASE_FORMAT_F32M |
    SCE_GXM_TEXTURE_SWIZZLE1_R000,
    SCE_GXM_TEXTURE_FORMAT_F32M_R111 = SCE_GXM_TEXTURE_BASE_FORMAT_F32M |
    SCE_GXM_TEXTURE_SWIZZLE1_R111,
    SCE_GXM_TEXTURE_FORMAT_F32M_R = SCE_GXM_TEXTURE_BASE_FORMAT_F32M |
    SCE_GXM_TEXTURE_SWIZZLE1_R,
    SCE_GXM_TEXTURE_FORMAT_X8S8S8U8_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_X8S8S8U8 | SCE_GXM_TEXTURE_SWIZZLE3_BGR,
    SCE_GXM_TEXTURE_FORMAT_X8U8S8S8_1RGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_X8S8S8U8 | SCE_GXM_TEXTURE_SWIZZLE3_RGB,
    SCE_GXM_TEXTURE_FORMAT_X8U24_SD = SCE_GXM_TEXTURE_BASE_FORMAT_X8U24 |
    SCE_GXM_TEXTURE_SWIZZLE2_SD,
    SCE_GXM_TEXTURE_FORMAT_U24X8_DS = SCE_GXM_TEXTURE_BASE_FORMAT_X8U24 |
    SCE_GXM_TEXTURE_SWIZZLE2_DS,
    SCE_GXM_TEXTURE_FORMAT_U32_000R = SCE_GXM_TEXTURE_BASE_FORMAT_U32 |
    SCE_GXM_TEXTURE_SWIZZLE1_000R,
    SCE_GXM_TEXTURE_FORMAT_U32_111R = SCE_GXM_TEXTURE_BASE_FORMAT_U32 |
    SCE_GXM_TEXTURE_SWIZZLE1_111R,
    SCE_GXM_TEXTURE_FORMAT_U32_RRRR = SCE_GXM_TEXTURE_BASE_FORMAT_U32 |
    SCE_GXM_TEXTURE_SWIZZLE1_RRRR,
    SCE_GXM_TEXTURE_FORMAT_U32_0RRR = SCE_GXM_TEXTURE_BASE_FORMAT_U32 |
    SCE_GXM_TEXTURE_SWIZZLE1_0RRR,
    SCE_GXM_TEXTURE_FORMAT_U32_1RRR = SCE_GXM_TEXTURE_BASE_FORMAT_U32 |
    SCE_GXM_TEXTURE_SWIZZLE1_1RRR,
    SCE_GXM_TEXTURE_FORMAT_U32_R000 = SCE_GXM_TEXTURE_BASE_FORMAT_U32 |
    SCE_GXM_TEXTURE_SWIZZLE1_R000,
    SCE_GXM_TEXTURE_FORMAT_U32_R111 = SCE_GXM_TEXTURE_BASE_FORMAT_U32 |
    SCE_GXM_TEXTURE_SWIZZLE1_R111,
    SCE_GXM_TEXTURE_FORMAT_U32_R = SCE_GXM_TEXTURE_BASE_FORMAT_U32 |
    SCE_GXM_TEXTURE_SWIZZLE1_R,
    SCE_GXM_TEXTURE_FORMAT_S32_000R = SCE_GXM_TEXTURE_BASE_FORMAT_S32 |
    SCE_GXM_TEXTURE_SWIZZLE1_000R,
    SCE_GXM_TEXTURE_FORMAT_S32_111R = SCE_GXM_TEXTURE_BASE_FORMAT_S32 |
    SCE_GXM_TEXTURE_SWIZZLE1_111R,
    SCE_GXM_TEXTURE_FORMAT_S32_RRRR = SCE_GXM_TEXTURE_BASE_FORMAT_S32 |
    SCE_GXM_TEXTURE_SWIZZLE1_RRRR,
    SCE_GXM_TEXTURE_FORMAT_S32_0RRR = SCE_GXM_TEXTURE_BASE_FORMAT_S32 |
    SCE_GXM_TEXTURE_SWIZZLE1_0RRR,
    SCE_GXM_TEXTURE_FORMAT_S32_1RRR = SCE_GXM_TEXTURE_BASE_FORMAT_S32 |
    SCE_GXM_TEXTURE_SWIZZLE1_1RRR,
    SCE_GXM_TEXTURE_FORMAT_S32_R000 = SCE_GXM_TEXTURE_BASE_FORMAT_S32 |
    SCE_GXM_TEXTURE_SWIZZLE1_R000,
    SCE_GXM_TEXTURE_FORMAT_S32_R111 = SCE_GXM_TEXTURE_BASE_FORMAT_S32 |
    SCE_GXM_TEXTURE_SWIZZLE1_R111,
    SCE_GXM_TEXTURE_FORMAT_S32_R = SCE_GXM_TEXTURE_BASE_FORMAT_S32 |
    SCE_GXM_TEXTURE_SWIZZLE1_R,
    SCE_GXM_TEXTURE_FORMAT_SE5M9M9M9_BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_SE5M9M9M9 | SCE_GXM_TEXTURE_SWIZZLE3_BGR,
    SCE_GXM_TEXTURE_FORMAT_SE5M9M9M9_RGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_SE5M9M9M9 | SCE_GXM_TEXTURE_SWIZZLE3_RGB,
    SCE_GXM_TEXTURE_FORMAT_F10F11F11_BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_F11F11F10 | SCE_GXM_TEXTURE_SWIZZLE3_BGR,
    SCE_GXM_TEXTURE_FORMAT_F11F11F10_RGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_F11F11F10 | SCE_GXM_TEXTURE_SWIZZLE3_RGB,
    SCE_GXM_TEXTURE_FORMAT_F16F16F16F16_ABGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_F16F16F16F16_ARGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB,
    SCE_GXM_TEXTURE_FORMAT_F16F16F16F16_RGBA =
    SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA,
    SCE_GXM_TEXTURE_FORMAT_F16F16F16F16_BGRA =
    SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA,
    SCE_GXM_TEXTURE_FORMAT_X16F16F16F16_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_X16F16F16F16_1RGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB,
    SCE_GXM_TEXTURE_FORMAT_F16F16F16X16_RGB1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1,
    SCE_GXM_TEXTURE_FORMAT_F16F16F16X16_BGR1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1,
    SCE_GXM_TEXTURE_FORMAT_U16U16U16U16_ABGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_U16U16U16U16_ARGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB,
    SCE_GXM_TEXTURE_FORMAT_U16U16U16U16_RGBA =
    SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA,
    SCE_GXM_TEXTURE_FORMAT_U16U16U16U16_BGRA =
    SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA,
    SCE_GXM_TEXTURE_FORMAT_X16U16U16U16_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_X16U16U16U16_1RGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB,
    SCE_GXM_TEXTURE_FORMAT_U16U16U16X16_RGB1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1,
    SCE_GXM_TEXTURE_FORMAT_U16U16U16X16_BGR1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1,
    SCE_GXM_TEXTURE_FORMAT_S16S16S16S16_ABGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_S16S16S16S16_ARGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB,
    SCE_GXM_TEXTURE_FORMAT_S16S16S16S16_RGBA =
    SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA,
    SCE_GXM_TEXTURE_FORMAT_S16S16S16S16_BGRA =
    SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA,
    SCE_GXM_TEXTURE_FORMAT_X16S16S16S16_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_X16S16S16S16_1RGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB,
    SCE_GXM_TEXTURE_FORMAT_S16S16S16X16_RGB1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1,
    SCE_GXM_TEXTURE_FORMAT_S16S16S16X16_BGR1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1,
    SCE_GXM_TEXTURE_FORMAT_F32F32_00GR = SCE_GXM_TEXTURE_BASE_FORMAT_F32F32 |
    SCE_GXM_TEXTURE_SWIZZLE2_00GR,
    SCE_GXM_TEXTURE_FORMAT_F32F32_GRRR = SCE_GXM_TEXTURE_BASE_FORMAT_F32F32 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRRR,
    SCE_GXM_TEXTURE_FORMAT_F32F32_RGGG = SCE_GXM_TEXTURE_BASE_FORMAT_F32F32 |
    SCE_GXM_TEXTURE_SWIZZLE2_RGGG,
    SCE_GXM_TEXTURE_FORMAT_F32F32_GRGR = SCE_GXM_TEXTURE_BASE_FORMAT_F32F32 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRGR,
    SCE_GXM_TEXTURE_FORMAT_F32F32_00RG = SCE_GXM_TEXTURE_BASE_FORMAT_F32F32 |
    SCE_GXM_TEXTURE_SWIZZLE2_00RG,
    SCE_GXM_TEXTURE_FORMAT_F32F32_GR = SCE_GXM_TEXTURE_BASE_FORMAT_F32F32 |
    SCE_GXM_TEXTURE_SWIZZLE2_GR,
    SCE_GXM_TEXTURE_FORMAT_U32U32_00GR = SCE_GXM_TEXTURE_BASE_FORMAT_U32U32 |
    SCE_GXM_TEXTURE_SWIZZLE2_00GR,
    SCE_GXM_TEXTURE_FORMAT_U32U32_GRRR = SCE_GXM_TEXTURE_BASE_FORMAT_U32U32 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRRR,
    SCE_GXM_TEXTURE_FORMAT_U32U32_RGGG = SCE_GXM_TEXTURE_BASE_FORMAT_U32U32 |
    SCE_GXM_TEXTURE_SWIZZLE2_RGGG,
    SCE_GXM_TEXTURE_FORMAT_U32U32_GRGR = SCE_GXM_TEXTURE_BASE_FORMAT_U32U32 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRGR,
    SCE_GXM_TEXTURE_FORMAT_U32U32_00RG = SCE_GXM_TEXTURE_BASE_FORMAT_U32U32 |
    SCE_GXM_TEXTURE_SWIZZLE2_00RG,
    SCE_GXM_TEXTURE_FORMAT_U32U32_GR = SCE_GXM_TEXTURE_BASE_FORMAT_U32U32 |
    SCE_GXM_TEXTURE_SWIZZLE2_GR,
    SCE_GXM_TEXTURE_FORMAT_PVRT2BPP_ABGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_PVRT2BPP | SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_PVRT2BPP_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_PVRT2BPP | SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_PVRT4BPP_ABGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_PVRT4BPP | SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_PVRT4BPP_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_PVRT4BPP | SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_PVRTII2BPP_ABGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_PVRTII2BPP | SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_PVRTII2BPP_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_PVRTII2BPP | SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_PVRTII4BPP_ABGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_PVRTII4BPP | SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_PVRTII4BPP_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_PVRTII4BPP | SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_UBC1_ABGR = SCE_GXM_TEXTURE_BASE_FORMAT_UBC1 |
    SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_UBC1_1BGR = SCE_GXM_TEXTURE_BASE_FORMAT_UBC1 |
    SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_UBC2_ABGR = SCE_GXM_TEXTURE_BASE_FORMAT_UBC2 |
    SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_UBC2_1BGR = SCE_GXM_TEXTURE_BASE_FORMAT_UBC2 |
    SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_UBC3_ABGR = SCE_GXM_TEXTURE_BASE_FORMAT_UBC3 |
    SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_UBC3_1BGR = SCE_GXM_TEXTURE_BASE_FORMAT_UBC3 |
    SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_UBC4_000R = SCE_GXM_TEXTURE_BASE_FORMAT_UBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_000R,
    SCE_GXM_TEXTURE_FORMAT_UBC4_111R = SCE_GXM_TEXTURE_BASE_FORMAT_UBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_111R,
    SCE_GXM_TEXTURE_FORMAT_UBC4_RRRR = SCE_GXM_TEXTURE_BASE_FORMAT_UBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_RRRR,
    SCE_GXM_TEXTURE_FORMAT_UBC4_0RRR = SCE_GXM_TEXTURE_BASE_FORMAT_UBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_0RRR,
    SCE_GXM_TEXTURE_FORMAT_UBC4_1RRR = SCE_GXM_TEXTURE_BASE_FORMAT_UBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_1RRR,
    SCE_GXM_TEXTURE_FORMAT_UBC4_R000 = SCE_GXM_TEXTURE_BASE_FORMAT_UBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_R000,
    SCE_GXM_TEXTURE_FORMAT_UBC4_R111 = SCE_GXM_TEXTURE_BASE_FORMAT_UBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_R111,
    SCE_GXM_TEXTURE_FORMAT_UBC4_R = SCE_GXM_TEXTURE_BASE_FORMAT_UBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_R,
    SCE_GXM_TEXTURE_FORMAT_SBC4_000R = SCE_GXM_TEXTURE_BASE_FORMAT_SBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_000R,
    SCE_GXM_TEXTURE_FORMAT_SBC4_111R = SCE_GXM_TEXTURE_BASE_FORMAT_SBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_111R,
    SCE_GXM_TEXTURE_FORMAT_SBC4_RRRR = SCE_GXM_TEXTURE_BASE_FORMAT_SBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_RRRR,
    SCE_GXM_TEXTURE_FORMAT_SBC4_0RRR = SCE_GXM_TEXTURE_BASE_FORMAT_SBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_0RRR,
    SCE_GXM_TEXTURE_FORMAT_SBC4_1RRR = SCE_GXM_TEXTURE_BASE_FORMAT_SBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_1RRR,
    SCE_GXM_TEXTURE_FORMAT_SBC4_R000 = SCE_GXM_TEXTURE_BASE_FORMAT_SBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_R000,
    SCE_GXM_TEXTURE_FORMAT_SBC4_R111 = SCE_GXM_TEXTURE_BASE_FORMAT_SBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_R111,
    SCE_GXM_TEXTURE_FORMAT_SBC4_R = SCE_GXM_TEXTURE_BASE_FORMAT_SBC4 |
    SCE_GXM_TEXTURE_SWIZZLE1_R,
    SCE_GXM_TEXTURE_FORMAT_UBC5_00GR = SCE_GXM_TEXTURE_BASE_FORMAT_UBC5 |
    SCE_GXM_TEXTURE_SWIZZLE2_00GR,
    SCE_GXM_TEXTURE_FORMAT_UBC5_GRRR = SCE_GXM_TEXTURE_BASE_FORMAT_UBC5 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRRR,
    SCE_GXM_TEXTURE_FORMAT_UBC5_RGGG = SCE_GXM_TEXTURE_BASE_FORMAT_UBC5 |
    SCE_GXM_TEXTURE_SWIZZLE2_RGGG,
    SCE_GXM_TEXTURE_FORMAT_UBC5_GRGR = SCE_GXM_TEXTURE_BASE_FORMAT_UBC5 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRGR,
    SCE_GXM_TEXTURE_FORMAT_UBC5_00RG = SCE_GXM_TEXTURE_BASE_FORMAT_UBC5 |
    SCE_GXM_TEXTURE_SWIZZLE2_00RG,
    SCE_GXM_TEXTURE_FORMAT_UBC5_GR = SCE_GXM_TEXTURE_BASE_FORMAT_UBC5 |
    SCE_GXM_TEXTURE_SWIZZLE2_GR,
    SCE_GXM_TEXTURE_FORMAT_SBC5_00GR = SCE_GXM_TEXTURE_BASE_FORMAT_SBC5 |
    SCE_GXM_TEXTURE_SWIZZLE2_00GR,
    SCE_GXM_TEXTURE_FORMAT_SBC5_GRRR = SCE_GXM_TEXTURE_BASE_FORMAT_SBC5 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRRR,
    SCE_GXM_TEXTURE_FORMAT_SBC5_RGGG = SCE_GXM_TEXTURE_BASE_FORMAT_SBC5 |
    SCE_GXM_TEXTURE_SWIZZLE2_RGGG,
    SCE_GXM_TEXTURE_FORMAT_SBC5_GRGR = SCE_GXM_TEXTURE_BASE_FORMAT_SBC5 |
    SCE_GXM_TEXTURE_SWIZZLE2_GRGR,
    SCE_GXM_TEXTURE_FORMAT_SBC5_00RG = SCE_GXM_TEXTURE_BASE_FORMAT_SBC5 |
    SCE_GXM_TEXTURE_SWIZZLE2_00RG,
    SCE_GXM_TEXTURE_FORMAT_SBC5_GR = SCE_GXM_TEXTURE_BASE_FORMAT_SBC5 |
    SCE_GXM_TEXTURE_SWIZZLE2_GR,
    SCE_GXM_TEXTURE_FORMAT_YUV420P2_CSC0 =
    SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P2 | SCE_GXM_TEXTURE_SWIZZLE_YUV_CSC0,
    SCE_GXM_TEXTURE_FORMAT_YVU420P2_CSC0 =
    SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P2 | SCE_GXM_TEXTURE_SWIZZLE_YVU_CSC0,
    SCE_GXM_TEXTURE_FORMAT_YUV420P2_CSC1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P2 | SCE_GXM_TEXTURE_SWIZZLE_YUV_CSC1,
    SCE_GXM_TEXTURE_FORMAT_YVU420P2_CSC1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P2 | SCE_GXM_TEXTURE_SWIZZLE_YVU_CSC1,
    SCE_GXM_TEXTURE_FORMAT_YUV420P3_CSC0 =
    SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P3 | SCE_GXM_TEXTURE_SWIZZLE_YUV_CSC0,
    SCE_GXM_TEXTURE_FORMAT_YVU420P3_CSC0 =
    SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P3 | SCE_GXM_TEXTURE_SWIZZLE_YVU_CSC0,
    SCE_GXM_TEXTURE_FORMAT_YUV420P3_CSC1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P3 | SCE_GXM_TEXTURE_SWIZZLE_YUV_CSC1,
    SCE_GXM_TEXTURE_FORMAT_YVU420P3_CSC1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P3 | SCE_GXM_TEXTURE_SWIZZLE_YVU_CSC1,
    SCE_GXM_TEXTURE_FORMAT_YUYV422_CSC0 = SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 |
    SCE_GXM_TEXTURE_SWIZZLE_YUYV_CSC0,
    SCE_GXM_TEXTURE_FORMAT_YVYU422_CSC0 = SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 |
    SCE_GXM_TEXTURE_SWIZZLE_YVYU_CSC0,
    SCE_GXM_TEXTURE_FORMAT_UYVY422_CSC0 = SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 |
    SCE_GXM_TEXTURE_SWIZZLE_UYVY_CSC0,
    SCE_GXM_TEXTURE_FORMAT_VYUY422_CSC0 = SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 |
    SCE_GXM_TEXTURE_SWIZZLE_VYUY_CSC0,
    SCE_GXM_TEXTURE_FORMAT_YUYV422_CSC1 = SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 |
    SCE_GXM_TEXTURE_SWIZZLE_YUYV_CSC1,
    SCE_GXM_TEXTURE_FORMAT_YVYU422_CSC1 = SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 |
    SCE_GXM_TEXTURE_SWIZZLE_YVYU_CSC1,
    SCE_GXM_TEXTURE_FORMAT_UYVY422_CSC1 = SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 |
    SCE_GXM_TEXTURE_SWIZZLE_UYVY_CSC1,
    SCE_GXM_TEXTURE_FORMAT_VYUY422_CSC1 = SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 |
    SCE_GXM_TEXTURE_SWIZZLE_VYUY_CSC1,
    SCE_GXM_TEXTURE_FORMAT_P4_ABGR = SCE_GXM_TEXTURE_BASE_FORMAT_P4 |
    SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_P4_ARGB = SCE_GXM_TEXTURE_BASE_FORMAT_P4 |
    SCE_GXM_TEXTURE_SWIZZLE4_ARGB,
    SCE_GXM_TEXTURE_FORMAT_P4_RGBA = SCE_GXM_TEXTURE_BASE_FORMAT_P4 |
    SCE_GXM_TEXTURE_SWIZZLE4_RGBA,
    SCE_GXM_TEXTURE_FORMAT_P4_BGRA = SCE_GXM_TEXTURE_BASE_FORMAT_P4 |
    SCE_GXM_TEXTURE_SWIZZLE4_BGRA,
    SCE_GXM_TEXTURE_FORMAT_P4_1BGR = SCE_GXM_TEXTURE_BASE_FORMAT_P4 |
    SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_P4_1RGB = SCE_GXM_TEXTURE_BASE_FORMAT_P4 |
    SCE_GXM_TEXTURE_SWIZZLE4_1RGB,
    SCE_GXM_TEXTURE_FORMAT_P4_RGB1 = SCE_GXM_TEXTURE_BASE_FORMAT_P4 |
    SCE_GXM_TEXTURE_SWIZZLE4_RGB1,
    SCE_GXM_TEXTURE_FORMAT_P4_BGR1 = SCE_GXM_TEXTURE_BASE_FORMAT_P4 |
    SCE_GXM_TEXTURE_SWIZZLE4_BGR1,
    SCE_GXM_TEXTURE_FORMAT_P8_ABGR = SCE_GXM_TEXTURE_BASE_FORMAT_P8 |
    SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_P8_ARGB = SCE_GXM_TEXTURE_BASE_FORMAT_P8 |
    SCE_GXM_TEXTURE_SWIZZLE4_ARGB,
    SCE_GXM_TEXTURE_FORMAT_P8_RGBA = SCE_GXM_TEXTURE_BASE_FORMAT_P8 |
    SCE_GXM_TEXTURE_SWIZZLE4_RGBA,
    SCE_GXM_TEXTURE_FORMAT_P8_BGRA = SCE_GXM_TEXTURE_BASE_FORMAT_P8 |
    SCE_GXM_TEXTURE_SWIZZLE4_BGRA,
    SCE_GXM_TEXTURE_FORMAT_P8_1BGR = SCE_GXM_TEXTURE_BASE_FORMAT_P8 |
    SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_P8_1RGB = SCE_GXM_TEXTURE_BASE_FORMAT_P8 |
    SCE_GXM_TEXTURE_SWIZZLE4_1RGB,
    SCE_GXM_TEXTURE_FORMAT_P8_RGB1 = SCE_GXM_TEXTURE_BASE_FORMAT_P8 |
    SCE_GXM_TEXTURE_SWIZZLE4_RGB1,
    SCE_GXM_TEXTURE_FORMAT_P8_BGR1 = SCE_GXM_TEXTURE_BASE_FORMAT_P8 |
    SCE_GXM_TEXTURE_SWIZZLE4_BGR1,
    SCE_GXM_TEXTURE_FORMAT_U8U8U8_BGR = SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8 |
    SCE_GXM_TEXTURE_SWIZZLE3_BGR,
    SCE_GXM_TEXTURE_FORMAT_U8U8U8_RGB = SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8 |
    SCE_GXM_TEXTURE_SWIZZLE3_RGB,
    SCE_GXM_TEXTURE_FORMAT_S8S8S8_BGR = SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8 |
    SCE_GXM_TEXTURE_SWIZZLE3_BGR,
    SCE_GXM_TEXTURE_FORMAT_S8S8S8_RGB = SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8 |
    SCE_GXM_TEXTURE_SWIZZLE3_RGB,
    SCE_GXM_TEXTURE_FORMAT_U2F10F10F10_ABGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR,
    SCE_GXM_TEXTURE_FORMAT_U2F10F10F10_ARGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB,
    SCE_GXM_TEXTURE_FORMAT_F10F10F10U2_RGBA =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA,
    SCE_GXM_TEXTURE_FORMAT_F10F10F10U2_BGRA =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA,
    SCE_GXM_TEXTURE_FORMAT_X2F10F10F10_1BGR =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR,
    SCE_GXM_TEXTURE_FORMAT_X2F10F10F10_1RGB =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB,
    SCE_GXM_TEXTURE_FORMAT_F10F10F10X2_RGB1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1,
    SCE_GXM_TEXTURE_FORMAT_F10F10F10X2_BGR1 =
    SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1,
    SCE_GXM_TEXTURE_FORMAT_L8 = SCE_GXM_TEXTURE_FORMAT_U8_1RRR,
    SCE_GXM_TEXTURE_FORMAT_A8 = SCE_GXM_TEXTURE_FORMAT_U8_R000,
    SCE_GXM_TEXTURE_FORMAT_R8 = SCE_GXM_TEXTURE_FORMAT_U8_000R,
    SCE_GXM_TEXTURE_FORMAT_A4R4G4B4 = SCE_GXM_TEXTURE_FORMAT_U4U4U4U4_ARGB,
    SCE_GXM_TEXTURE_FORMAT_A1R5G5B5 = SCE_GXM_TEXTURE_FORMAT_U1U5U5U5_ARGB,
    SCE_GXM_TEXTURE_FORMAT_R5G6B5 = SCE_GXM_TEXTURE_FORMAT_U5U6U5_RGB,
    SCE_GXM_TEXTURE_FORMAT_A8L8 = SCE_GXM_TEXTURE_FORMAT_U8U8_GRRR,
    SCE_GXM_TEXTURE_FORMAT_L8A8 = SCE_GXM_TEXTURE_FORMAT_U8U8_RGGG,
    SCE_GXM_TEXTURE_FORMAT_G8R8 = SCE_GXM_TEXTURE_FORMAT_U8U8_00GR,
    SCE_GXM_TEXTURE_FORMAT_L16 = SCE_GXM_TEXTURE_FORMAT_U16_1RRR,
    SCE_GXM_TEXTURE_FORMAT_A16 = SCE_GXM_TEXTURE_FORMAT_U16_R000,
    SCE_GXM_TEXTURE_FORMAT_R16 = SCE_GXM_TEXTURE_FORMAT_U16_000R,
    SCE_GXM_TEXTURE_FORMAT_D16 = SCE_GXM_TEXTURE_FORMAT_U16_R,
    SCE_GXM_TEXTURE_FORMAT_LF16 = SCE_GXM_TEXTURE_FORMAT_F16_1RRR,
    SCE_GXM_TEXTURE_FORMAT_AF16 = SCE_GXM_TEXTURE_FORMAT_F16_R000,
    SCE_GXM_TEXTURE_FORMAT_RF16 = SCE_GXM_TEXTURE_FORMAT_F16_000R,
    SCE_GXM_TEXTURE_FORMAT_A8R8G8B8 = SCE_GXM_TEXTURE_FORMAT_U8U8U8U8_ARGB,
    SCE_GXM_TEXTURE_FORMAT_A8B8G8R8 = SCE_GXM_TEXTURE_FORMAT_U8U8U8U8_ABGR,
    SCE_GXM_TEXTURE_FORMAT_AF16LF16 = SCE_GXM_TEXTURE_FORMAT_F16F16_GRRR,
    SCE_GXM_TEXTURE_FORMAT_LF16AF16 = SCE_GXM_TEXTURE_FORMAT_F16F16_RGGG,
    SCE_GXM_TEXTURE_FORMAT_GF16RF16 = SCE_GXM_TEXTURE_FORMAT_F16F16_00GR,
    SCE_GXM_TEXTURE_FORMAT_LF32M = SCE_GXM_TEXTURE_FORMAT_F32M_1RRR,
    SCE_GXM_TEXTURE_FORMAT_AF32M = SCE_GXM_TEXTURE_FORMAT_F32M_R000,
    SCE_GXM_TEXTURE_FORMAT_RF32M = SCE_GXM_TEXTURE_FORMAT_F32M_000R,
    SCE_GXM_TEXTURE_FORMAT_DF32M = SCE_GXM_TEXTURE_FORMAT_F32M_R,
    SCE_GXM_TEXTURE_FORMAT_VYUY = SCE_GXM_TEXTURE_FORMAT_VYUY422_CSC0,
    SCE_GXM_TEXTURE_FORMAT_YVYU = SCE_GXM_TEXTURE_FORMAT_YVYU422_CSC0,
    SCE_GXM_TEXTURE_FORMAT_UBC1 = SCE_GXM_TEXTURE_FORMAT_UBC1_ABGR,
    SCE_GXM_TEXTURE_FORMAT_UBC2 = SCE_GXM_TEXTURE_FORMAT_UBC2_ABGR,
    SCE_GXM_TEXTURE_FORMAT_UBC3 = SCE_GXM_TEXTURE_FORMAT_UBC3_ABGR,
    SCE_GXM_TEXTURE_FORMAT_PVRT2BPP = SCE_GXM_TEXTURE_FORMAT_PVRT2BPP_ABGR,
    SCE_GXM_TEXTURE_FORMAT_PVRT4BPP = SCE_GXM_TEXTURE_FORMAT_PVRT4BPP_ABGR,
    SCE_GXM_TEXTURE_FORMAT_PVRTII2BPP =
    SCE_GXM_TEXTURE_FORMAT_PVRTII2BPP_ABGR,
    SCE_GXM_TEXTURE_FORMAT_PVRTII4BPP =
    SCE_GXM_TEXTURE_FORMAT_PVRTII4BPP_ABGR
} SceGxmTextureFormat;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_TEXTURE_ FORMAT_U8_000R	SCE_GXM_TEXTURE_ BASE_FORMAT_U8 | SCE_GXM_TEXTURE_ SWIZZLE1_000R	The U8 value is swizzled to 000R (in ABGR notation).

Macro	Value	Description
SCE_GXM_TEXTURE_FORMAT_U8_111R	SCE_GXM_TEXTURE_BASE_FORMAT_U8 | SCE_GXM_TEXTURE_SWIZZLE1_111R	The U8 value is swizzled to 111R (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_U8_RRRR	SCE_GXM_TEXTURE_BASE_FORMAT_U8 | SCE_GXM_TEXTURE_SWIZZLE1_RRRR	The U8 value is swizzled to RRRR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_U8_0RRR	SCE_GXM_TEXTURE_BASE_FORMAT_U8 | SCE_GXM_TEXTURE_SWIZZLE1_0RRR	The U8 value is swizzled to 0RRR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_U8_1RRR	SCE_GXM_TEXTURE_BASE_FORMAT_U8 | SCE_GXM_TEXTURE_SWIZZLE1_1RRR	The U8 value is swizzled to 1RRR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_U8_R000	SCE_GXM_TEXTURE_BASE_FORMAT_U8 | SCE_GXM_TEXTURE_SWIZZLE1_R000	The U8 value is swizzled to R000 (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_U8_R111	SCE_GXM_TEXTURE_BASE_FORMAT_U8 | SCE_GXM_TEXTURE_SWIZZLE1_R111	The U8 value is swizzled to R111 (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_U8_R	SCE_GXM_TEXTURE_BASE_FORMAT_U8 | SCE_GXM_TEXTURE_SWIZZLE1_R	The U8 value is returned as a single component result.
SCE_GXM_TEXTURE_FORMAT_S8_000R	SCE_GXM_TEXTURE_BASE_FORMAT_S8 | SCE_GXM_TEXTURE_SWIZZLE1_000R	The S8 value is swizzled to 000R (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_S8_111R	SCE_GXM_TEXTURE_BASE_FORMAT_S8 | SCE_GXM_TEXTURE_SWIZZLE1_111R	The S8 value is swizzled to 111R (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_S8_RRRR	SCE_GXM_TEXTURE_BASE_FORMAT_S8 | SCE_GXM_TEXTURE_SWIZZLE1_RRRR	The S8 value is swizzled to RRRR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_S8_0RRR	SCE_GXM_TEXTURE_BASE_FORMAT_S8 | SCE_GXM_TEXTURE_SWIZZLE1_0RRR	The S8 value is swizzled to 0RRR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_S8_1RRR	SCE_GXM_TEXTURE_BASE_FORMAT_S8 | SCE_GXM_TEXTURE_SWIZZLE1_1RRR	The S8 value is swizzled to 1RRR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_S8_R000	SCE_GXM_TEXTURE_BASE_FORMAT_S8 | SCE_GXM_TEXTURE_SWIZZLE1_R000	The S8 value is swizzled to R000 (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_S8_R111	SCE_GXM_TEXTURE_BASE_FORMAT_S8 | SCE_GXM_TEXTURE_SWIZZLE1_R111	The S8 value is swizzled to R111 (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_S8_R	SCE_GXM_TEXTURE_BASE_FORMAT_S8 | SCE_GXM_TEXTURE_SWIZZLE1_R	The S8 value is returned as a single component result.

Macro	Value	Description
SCE_GXM_TEXTURE_FORMAT_U4U4U4U4_ABGR	SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR	The U4U4U4U4 data is read in ABGR order from memory.
SCE_GXM_TEXTURE_FORMAT_U4U4U4U4_ARGB	SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB	The U4U4U4U4 data is read in ARGB order from memory.
SCE_GXM_TEXTURE_FORMAT_U4U4U4U4_RGBA	SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA	The U4U4U4U4 data is read in RGBA order from memory.
SCE_GXM_TEXTURE_FORMAT_U4U4U4U4_BGRA	SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA	The U4U4U4U4 data is read in BGRA order from memory.
SCE_GXM_TEXTURE_FORMAT_X4U4U4U4_1BGR	SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR	The U4U4U4U4 data is read in ABGR order from memory, A is replaced with 0xf.
SCE_GXM_TEXTURE_FORMAT_X4U4U4U4_1RGB	SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB	The U4U4U4U4 data is read in ARGB order from memory, A is replaced with 0xf.
SCE_GXM_TEXTURE_FORMAT_U4U4U4X4_RGB1	SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1	The U4U4U4U4 data is read in RGBA order from memory, A is replaced with 0xf.
SCE_GXM_TEXTURE_FORMAT_U4U4U4X4_BGR1	SCE_GXM_TEXTURE_BASE_FORMAT_U4U4U4U4 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1	The U4U4U4U4 data is read in BGRA order from memory, A is replaced with 0xf.
SCE_GXM_TEXTURE_FORMAT_U8U3U3U2_ARGB	SCE_GXM_TEXTURE_BASE_FORMAT_U8U3U3U2	The U8U3U3U2 data is read in ARGB order from memory.
SCE_GXM_TEXTURE_FORMAT_U1U5U5U5_ABGR	SCE_GXM_TEXTURE_BASE_FORMAT_U1U5U5U5 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR	The U1U5U5U5 data is read in ABGR order from memory.
SCE_GXM_TEXTURE_FORMAT_U1U5U5U5_ARGB	SCE_GXM_TEXTURE_BASE_FORMAT_U1U5U5U5 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB	The U1U5U5U5 data is read in ARGB order from memory.
SCE_GXM_TEXTURE_FORMAT_U5U5U5U1_RGBA	SCE_GXM_TEXTURE_BASE_FORMAT_U1U5U5U5 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA	The U5U5U5U1 data is read in RGBA order from memory.

Macro	Value	Description
SCE_GXM_TEXTURE_ FORMAT_U5U5U5U1_ BGRA	SCE_GXM_TEXTURE_ BASE_FORMAT_ U1U5U5U5 | SCE_GXM_TEXTURE_ SWIZZLE4_BGRA	The U5U5U5U1 data is read in BGRA order from memory.
SCE_GXM_TEXTURE_ FORMAT_X1U5U5U5_ 1BGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ U1U5U5U5 | SCE_GXM_TEXTURE_ SWIZZLE4_1BGR	The U1U5U5U5 data is read in ABGR order from memory, A is replaced with 1.
SCE_GXM_TEXTURE_ FORMAT_X1U5U5U5_ 1RGB	SCE_GXM_TEXTURE_ BASE_FORMAT_ U1U5U5U5 | SCE_GXM_TEXTURE_ SWIZZLE4_1RGB	The U1U5U5U5 data is read in ARGB order from memory, A is replaced with 1.
SCE_GXM_TEXTURE_ FORMAT_U5U5U5X1_ RGB1	SCE_GXM_TEXTURE_ BASE_FORMAT_ U1U5U5U5 | SCE_GXM_TEXTURE_ SWIZZLE4_RGB1	The U5U5U5U1 data is read in RGBA order from memory, A is replaced with 1.
SCE_GXM_TEXTURE_ FORMAT_U5U5U5X1_ BGR1	SCE_GXM_TEXTURE_ BASE_FORMAT_ U1U5U5U5 | SCE_GXM_TEXTURE_ SWIZZLE4_BGR1	The U5U5U5U1 data is read in BGRA order from memory, A is replaced with 1.
SCE_GXM_TEXTURE_ FORMAT_U5U6U5_BGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ U5U6U5 | SCE_GXM_TEXTURE_ SWIZZLE3_BGR	The U5U6U5 data is read in BGR order from memory, A is implicit and assigned 1.
SCE_GXM_TEXTURE_ FORMAT_U5U6U5_RGB	SCE_GXM_TEXTURE_ BASE_FORMAT_ U5U6U5 | SCE_GXM_TEXTURE_ SWIZZLE3_RGB	The U5U6U5 data is read in RGB order from memory, A is implicit and assigned 1.
SCE_GXM_TEXTURE_ FORMAT_U6S5S5_BGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ S5S5U6 | SCE_GXM_TEXTURE_ SWIZZLE3_BGR	The U6S5S5 data is read in BGR order from memory, A is implicit and assigned 1.
SCE_GXM_TEXTURE_ FORMAT_S5S5U6_RGB	SCE_GXM_TEXTURE_ BASE_FORMAT_ S5S5U6 | SCE_GXM_TEXTURE_ SWIZZLE3_RGB	The S5S5U6 data is read in RGB order from memory, A is implicit and assigned 1.
SCE_GXM_TEXTURE_ FORMAT_U8U8_00GR	SCE_GXM_TEXTURE_ BASE_FORMAT_U8U8 | SCE_GXM_TEXTURE_ SWIZZLE2_00GR	The U8U8 (GR) value is swizzled to 00GR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U8U8_GRRR	SCE_GXM_TEXTURE_ BASE_FORMAT_U8U8 | SCE_GXM_TEXTURE_ SWIZZLE2_GRRR	The U8U8 (GR) value is swizzled to GRRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U8U8_RGGG	SCE_GXM_TEXTURE_ BASE_FORMAT_U8U8 | SCE_GXM_TEXTURE_ SWIZZLE2_RGGG	The U8U8 (GR) value is swizzled to RGGG (in ABGR notation).

Macro	Value	Description
SCE_GXM_TEXTURE_ FORMAT_U8U8_GRGR	SCE_GXM_TEXTURE_ BASE_FORMAT_U8U8 | SCE_GXM_TEXTURE_ SWIZZLE2_GRGR	The U8U8 (GR) value is swizzled to GRGR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U8U8_00RG	SCE_GXM_TEXTURE_ BASE_FORMAT_U8U8 | SCE_GXM_TEXTURE_ SWIZZLE2_00RG	The U8U8 (GR) value is swizzled to 00RG (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U8U8_GR	SCE_GXM_TEXTURE_ BASE_FORMAT_U8U8 | SCE_GXM_TEXTURE_ SWIZZLE2_GR	The U8U8 (GR) value is returned as a 2-component result.
SCE_GXM_TEXTURE_ FORMAT_S8S8_00GR	SCE_GXM_TEXTURE_ BASE_FORMAT_S8S8 | SCE_GXM_TEXTURE_ SWIZZLE2_00GR	The S8S8 (GR) value is swizzled to 00GR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S8S8_GRRR	SCE_GXM_TEXTURE_ BASE_FORMAT_S8S8 | SCE_GXM_TEXTURE_ SWIZZLE2_GRRR	The S8S8 (GR) value is swizzled to GRRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S8S8_RGGG	SCE_GXM_TEXTURE_ BASE_FORMAT_S8S8 | SCE_GXM_TEXTURE_ SWIZZLE2_RGGG	The S8S8 (GR) value is swizzled to RGGG (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S8S8_GRGR	SCE_GXM_TEXTURE_ BASE_FORMAT_S8S8 | SCE_GXM_TEXTURE_ SWIZZLE2_GRGR	The S8S8 (GR) value is swizzled to GRGR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S8S8_00RG	SCE_GXM_TEXTURE_ BASE_FORMAT_S8S8 | SCE_GXM_TEXTURE_ SWIZZLE2_00RG	The S8S8 (GR) value is swizzled to 00RG (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S8S8_GR	SCE_GXM_TEXTURE_ BASE_FORMAT_S8S8 | SCE_GXM_TEXTURE_ SWIZZLE2_GR	The S8S8 (GR) value is returned as a 2-component result.
SCE_GXM_TEXTURE_ FORMAT_U16_000R	SCE_GXM_TEXTURE_ BASE_FORMAT_U16 | SCE_GXM_TEXTURE_ SWIZZLE1_000R	The U16 value is swizzled to 000R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U16_111R	SCE_GXM_TEXTURE_ BASE_FORMAT_U16 | SCE_GXM_TEXTURE_ SWIZZLE1_111R	The U16 value is swizzled to 111R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U16_RRRR	SCE_GXM_TEXTURE_ BASE_FORMAT_U16 | SCE_GXM_TEXTURE_ SWIZZLE1_RRRR	The U16 value is swizzled to RRRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U16_0RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_U16 | SCE_GXM_TEXTURE_ SWIZZLE1_0RRR	The U16 value is swizzled to 0RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U16_1RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_U16 | SCE_GXM_TEXTURE_ SWIZZLE1_1RRR	The U16 value is swizzled to 1RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U16_R000	SCE_GXM_TEXTURE_ BASE_FORMAT_U16 | SCE_GXM_TEXTURE_ SWIZZLE1_R000	The U16 value is swizzled to R000 (in ABGR notation).

Macro	Value	Description
SCE_GXM_TEXTURE_ FORMAT_U16_R111	SCE_GXM_TEXTURE_ BASE_FORMAT_U16 | SCE_GXM_TEXTURE_ SWIZZLE1_R111	The U16 value is swizzled to R111 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U16_R	SCE_GXM_TEXTURE_ BASE_FORMAT_U16 | SCE_GXM_TEXTURE_ SWIZZLE1_R	The U16 value is returned as a single component result.
SCE_GXM_TEXTURE_ FORMAT_S16_000R	SCE_GXM_TEXTURE_ BASE_FORMAT_S16 | SCE_GXM_TEXTURE_ SWIZZLE1_000R	The S16 value is swizzled to 000R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S16_111R	SCE_GXM_TEXTURE_ BASE_FORMAT_S16 | SCE_GXM_TEXTURE_ SWIZZLE1_111R	The S16 value is swizzled to 111R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S16_RRRR	SCE_GXM_TEXTURE_ BASE_FORMAT_S16 | SCE_GXM_TEXTURE_ SWIZZLE1_RRRR	The S16 value is swizzled to RRRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S16_0RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_S16 | SCE_GXM_TEXTURE_ SWIZZLE1_0RRR	The S16 value is swizzled to 0RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S16_1RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_S16 | SCE_GXM_TEXTURE_ SWIZZLE1_1RRR	The S16 value is swizzled to 1RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S16_R000	SCE_GXM_TEXTURE_ BASE_FORMAT_S16 | SCE_GXM_TEXTURE_ SWIZZLE1_R000	The S16 value is swizzled to R000 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S16_R111	SCE_GXM_TEXTURE_ BASE_FORMAT_S16 | SCE_GXM_TEXTURE_ SWIZZLE1_R111	The S16 value is swizzled to R111 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S16_R	SCE_GXM_TEXTURE_ BASE_FORMAT_S16 | SCE_GXM_TEXTURE_ SWIZZLE1_R	The S16 value is returned as a single component result.
SCE_GXM_TEXTURE_ FORMAT_F16_000R	SCE_GXM_TEXTURE_ BASE_FORMAT_F16 | SCE_GXM_TEXTURE_ SWIZZLE1_000R	The F16 value is swizzled to 000R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F16_111R	SCE_GXM_TEXTURE_ BASE_FORMAT_F16 | SCE_GXM_TEXTURE_ SWIZZLE1_111R	The F16 value is swizzled to 111R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F16_RRRR	SCE_GXM_TEXTURE_ BASE_FORMAT_F16 | SCE_GXM_TEXTURE_ SWIZZLE1_RRRR	The F16 value is swizzled to RRRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F16_0RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_F16 | SCE_GXM_TEXTURE_ SWIZZLE1_0RRR	The F16 value is swizzled to 0RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F16_1RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_F16 | SCE_GXM_TEXTURE_ SWIZZLE1_1RRR	The F16 value is swizzled to 1RRR (in ABGR notation).

Macro	Value	Description
SCE_GXM_TEXTURE_ FORMAT_F16_R000	SCE_GXM_TEXTURE_ BASE_FORMAT_F16 | SCE_GXM_TEXTURE_ SWIZZLE1_R000	The F16 value is swizzled to R000 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F16_R111	SCE_GXM_TEXTURE_ BASE_FORMAT_F16 | SCE_GXM_TEXTURE_ SWIZZLE1_R111	The F16 value is swizzled to R111 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F16_R	SCE_GXM_TEXTURE_ BASE_FORMAT_F16 | SCE_GXM_TEXTURE_ SWIZZLE1_R	The F16 value is returned as a single component result.
SCE_GXM_TEXTURE_ FORMAT_U8U8U8U8_ ABGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ U8U8U8U8 | SCE_GXM_TEXTURE_ SWIZZLE4_ABGR	The U8U8U8U8 data is read in ABGR order from memory.
SCE_GXM_TEXTURE_ FORMAT_U8U8U8U8_ ARGB	SCE_GXM_TEXTURE_ BASE_FORMAT_ U8U8U8U8 | SCE_GXM_TEXTURE_ SWIZZLE4_ARGB	The U8U8U8U8 data is read in ARGB order from memory.
SCE_GXM_TEXTURE_ FORMAT_U8U8U8U8_ RGBA	SCE_GXM_TEXTURE_ BASE_FORMAT_ U8U8U8U8 | SCE_GXM_TEXTURE_ SWIZZLE4_RGBA	The U8U8U8U8 data is read in RGBA order from memory.
SCE_GXM_TEXTURE_ FORMAT_U8U8U8U8_ BGRA	SCE_GXM_TEXTURE_ BASE_FORMAT_ U8U8U8U8 | SCE_GXM_TEXTURE_ SWIZZLE4_BGRA	The U8U8U8U8 data is read in BGRA order from memory.
SCE_GXM_TEXTURE_ FORMAT_X8U8U8U8_ 1BGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ U8U8U8U8 | SCE_GXM_TEXTURE_ SWIZZLE4_1BGR	The U8U8U8U8 data is read in ABGR order from memory, A is replaced with 0xff.
SCE_GXM_TEXTURE_ FORMAT_X8U8U8U8_ 1RGB	SCE_GXM_TEXTURE_ BASE_FORMAT_ U8U8U8U8 | SCE_GXM_TEXTURE_ SWIZZLE4_1RGB	The U8U8U8U8 data is read in ARGB order from memory, A is replaced with 0xff.
SCE_GXM_TEXTURE_ FORMAT_U8U8U8X8_ RGB1	SCE_GXM_TEXTURE_ BASE_FORMAT_ U8U8U8U8 | SCE_GXM_TEXTURE_ SWIZZLE4_RGB1	The U8U8U8U8 data is read in RGBA order from memory, A is replaced with 0xff.
SCE_GXM_TEXTURE_ FORMAT_U8U8U8X8_ BGR1	SCE_GXM_TEXTURE_ BASE_FORMAT_ U8U8U8U8 | SCE_GXM_TEXTURE_ SWIZZLE4_BGR1	The U8U8U8U8 data is read in BGRA order from memory, A is replaced with 0xff.
SCE_GXM_TEXTURE_ FORMAT_S8S8S8S8_ ABGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ S8S8S8S8 | SCE_GXM_TEXTURE_ SWIZZLE4_ABGR	The S8S8S8S8 data is read in ABGR order from memory.

Macro	Value	Description
SCE_GXM_TEXTURE_FORMAT_S8S8S8S8_ARGB	SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB	The S8S8S8S8 data is read in ARGB order from memory.
SCE_GXM_TEXTURE_FORMAT_S8S8S8S8_RGBA	SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA	The S8S8S8S8 data is read in RGBA order from memory.
SCE_GXM_TEXTURE_FORMAT_S8S8S8S8_BGRA	SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA	The S8S8S8S8 data is read in BGRA order from memory.
SCE_GXM_TEXTURE_FORMAT_X8S8S8S8_1BGR	SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR	The S8S8S8S8 data is read in ABGR order from memory, A is replaced with 0x7f.
SCE_GXM_TEXTURE_FORMAT_X8S8S8S8_1RGB	SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB	The S8S8S8S8 data is read in ARGB order from memory, A is replaced with 0x7f.
SCE_GXM_TEXTURE_FORMAT_S8S8S8X8_RGB1	SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1	The S8S8S8S8 data is read in RGBA order from memory, A is replaced with 0x7f.
SCE_GXM_TEXTURE_FORMAT_S8S8S8X8_BGR1	SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1	The S8S8S8S8 data is read in BGRA order from memory, A is replaced with 0x7f.
SCE_GXM_TEXTURE_FORMAT_U2U10U10U10_ABGR	SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR	The U2U10U10U10 data is read in ABGR order from memory.
SCE_GXM_TEXTURE_FORMAT_U2U10U10U10_ARGB	SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB	The U2U10U10U10 data is read in ARGB order from memory.
SCE_GXM_TEXTURE_FORMAT_U10U10U10U2_RGBA	SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA	The U10U10U10U2 data is read in RGBA order from memory.
SCE_GXM_TEXTURE_FORMAT_U10U10U10U2_BGRA	SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA	The U10U10U10U2 data is read in BGRA order from memory.
SCE_GXM_TEXTURE_FORMAT_X2U10U10U10_1BGR	SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR	The U2U10U10U10 data is read in ABGR order from memory, A is replaced with 0x3.

Macro	Value	Description
SCE_GXM_TEXTURE_FORMAT_X2U10U10U10_1RGB	SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB	The U2U10U10U10 data is read in ARGB order from memory, A is replaced with 0x3.
SCE_GXM_TEXTURE_FORMAT_U10U10U10X2_RGB1	SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1	The U10U10U10U2 data is read in RGBA order from memory, A is replaced with 0x3.
SCE_GXM_TEXTURE_FORMAT_U10U10U10X2_BGR1	SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1	The U10U10U10U2 data is read in BGRA order from memory, A is replaced with 0x3.
SCE_GXM_TEXTURE_FORMAT_U16U16_00GR	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16 | SCE_GXM_TEXTURE_SWIZZLE2_00GR	The U16U16 (GR) value is swizzled to 00GR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_U16U16_GRRR	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16 | SCE_GXM_TEXTURE_SWIZZLE2_GRRR	The U16U16 (GR) value is swizzled to GRRR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_U16U16_RGGG	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16 | SCE_GXM_TEXTURE_SWIZZLE2_RGGG	The U16U16 (GR) value is swizzled to RGGG (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_U16U16_GRGR	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16 | SCE_GXM_TEXTURE_SWIZZLE2_GRGR	The U16U16 (GR) value is swizzled to GRGR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_U16U16_00RG	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16 | SCE_GXM_TEXTURE_SWIZZLE2_00RG	The U16U16 (GR) value is swizzled to 00RG (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_U16U16_GR	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16 | SCE_GXM_TEXTURE_SWIZZLE2_GR	The U16U16 (GR) value is returned as a 2-component result.
SCE_GXM_TEXTURE_FORMAT_S16S16_00GR	SCE_GXM_TEXTURE_BASE_FORMAT_S16S16 | SCE_GXM_TEXTURE_SWIZZLE2_00GR	The S16S16 (GR) value is swizzled to 00GR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_S16S16_GRRR	SCE_GXM_TEXTURE_BASE_FORMAT_S16S16 | SCE_GXM_TEXTURE_SWIZZLE2_GRRR	The S16S16 (GR) value is swizzled to GRRR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_S16S16_RGGG	SCE_GXM_TEXTURE_BASE_FORMAT_S16S16 | SCE_GXM_TEXTURE_SWIZZLE2_RGGG	The S16S16 (GR) value is swizzled to RGGG (in ABGR notation).

Macro	Value	Description
`SCE_GXM_TEXTURE_FORMAT_S16S16_GRGR`	`SCE_GXM_TEXTURE_BASE_FORMAT_S16S16 | SCE_GXM_TEXTURE_SWIZZLE2_GRGR`	The S16S16 (GR) value is swizzled to GRGR (in ABGR notation).
`SCE_GXM_TEXTURE_FORMAT_S16S16_00RG`	`SCE_GXM_TEXTURE_BASE_FORMAT_S16S16 | SCE_GXM_TEXTURE_SWIZZLE2_00RG`	The S16S16 (GR) value is swizzled to 00RG (in ABGR notation).
`SCE_GXM_TEXTURE_FORMAT_S16S16_GR`	`SCE_GXM_TEXTURE_BASE_FORMAT_S16S16 | SCE_GXM_TEXTURE_SWIZZLE2_GR`	The S16S16 (GR) value is returned as a 2-component result.
`SCE_GXM_TEXTURE_FORMAT_F16F16_00GR`	`SCE_GXM_TEXTURE_BASE_FORMAT_F16F16 | SCE_GXM_TEXTURE_SWIZZLE2_00GR`	The F16F16 (GR) value is swizzled to 00GR (in ABGR notation).
`SCE_GXM_TEXTURE_FORMAT_F16F16_GRRR`	`SCE_GXM_TEXTURE_BASE_FORMAT_F16F16 | SCE_GXM_TEXTURE_SWIZZLE2_GRRR`	The F16F16 (GR) value is swizzled to GRRR (in ABGR notation).
`SCE_GXM_TEXTURE_FORMAT_F16F16_RGGG`	`SCE_GXM_TEXTURE_BASE_FORMAT_F16F16 | SCE_GXM_TEXTURE_SWIZZLE2_RGGG`	The F16F16 (GR) value is swizzled to RGGG (in ABGR notation).
`SCE_GXM_TEXTURE_FORMAT_F16F16_GRGR`	`SCE_GXM_TEXTURE_BASE_FORMAT_F16F16 | SCE_GXM_TEXTURE_SWIZZLE2_GRGR`	The F16F16 (GR) value is swizzled to GRGR (in ABGR notation).
`SCE_GXM_TEXTURE_FORMAT_F16F16_00RG`	`SCE_GXM_TEXTURE_BASE_FORMAT_F16F16 | SCE_GXM_TEXTURE_SWIZZLE2_00RG`	The F16F16 (GR) value is swizzled to 00RG (in ABGR notation).
`SCE_GXM_TEXTURE_FORMAT_F16F16_GR`	`SCE_GXM_TEXTURE_BASE_FORMAT_F16F16 | SCE_GXM_TEXTURE_SWIZZLE2_GR`	The F16F16 (GR) value is returned as a 2-component result.
`SCE_GXM_TEXTURE_FORMAT_F32_000R`	`SCE_GXM_TEXTURE_BASE_FORMAT_F32 | SCE_GXM_TEXTURE_SWIZZLE1_000R`	The F32 value is swizzled to 000R (in ABGR notation).
`SCE_GXM_TEXTURE_FORMAT_F32_111R`	`SCE_GXM_TEXTURE_BASE_FORMAT_F32 | SCE_GXM_TEXTURE_SWIZZLE1_111R`	The F32 value is swizzled to 111R (in ABGR notation).
`SCE_GXM_TEXTURE_FORMAT_F32_RRRR`	`SCE_GXM_TEXTURE_BASE_FORMAT_F32 | SCE_GXM_TEXTURE_SWIZZLE1_RRRR`	The F32 value is swizzled to RRRR (in ABGR notation).

Macro	Value	Description
SCE_GXM_TEXTURE_ FORMAT_F32_0RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_F32 | SCE_GXM_TEXTURE_ SWIZZLE1_0RRR	The F32 value is swizzled to 0RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F32_1RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_F32 | SCE_GXM_TEXTURE_ SWIZZLE1_1RRR	The F32 value is swizzled to 1RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F32_R000	SCE_GXM_TEXTURE_ BASE_FORMAT_F32 | SCE_GXM_TEXTURE_ SWIZZLE1_R000	The F32 value is swizzled to R000 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F32_R111	SCE_GXM_TEXTURE_ BASE_FORMAT_F32 | SCE_GXM_TEXTURE_ SWIZZLE1_R111	The F32 value is swizzled to R111 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F32_R	SCE_GXM_TEXTURE_ BASE_FORMAT_F32 | SCE_GXM_TEXTURE_ SWIZZLE1_R	The F32 value is returned as a single component result.
SCE_GXM_TEXTURE_ FORMAT_F32M_000R	SCE_GXM_TEXTURE_ BASE_FORMAT_F32M | SCE_GXM_TEXTURE_ SWIZZLE1_000R	The F32M value is swizzled to 000R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F32M_111R	SCE_GXM_TEXTURE_ BASE_FORMAT_F32M | SCE_GXM_TEXTURE_ SWIZZLE1_111R	The F32M value is swizzled to 111R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F32M_RRRR	SCE_GXM_TEXTURE_ BASE_FORMAT_F32M | SCE_GXM_TEXTURE_ SWIZZLE1_RRRR	The F32M value is swizzled to RRRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F32M_0RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_F32M | SCE_GXM_TEXTURE_ SWIZZLE1_0RRR	The F32M value is swizzled to 0RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F32M_1RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_F32M | SCE_GXM_TEXTURE_ SWIZZLE1_1RRR	The F32M value is swizzled to 1RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F32M_R000	SCE_GXM_TEXTURE_ BASE_FORMAT_F32M | SCE_GXM_TEXTURE_ SWIZZLE1_R000	The F32M value is swizzled to R000 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F32M_R111	SCE_GXM_TEXTURE_ BASE_FORMAT_F32M | SCE_GXM_TEXTURE_ SWIZZLE1_R111	The F32M value is swizzled to R111 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F32M_R	SCE_GXM_TEXTURE_ BASE_FORMAT_F32M | SCE_GXM_TEXTURE_ SWIZZLE1_R	The F32M value is returned as a single component result.
SCE_GXM_TEXTURE_ FORMAT_X8S8S8U8_ 1BGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ X8S8S8U8 | SCE_GXM_TEXTURE_ SWIZZLE3_BGR	The U8S8S8U8 data is read in ABGR order from memory, A is replaced with 0xff.

Macro	Value	Description
SCE_GXM_TEXTURE_ FORMAT_X8U8S8S8_ 1RGB	SCE_GXM_TEXTURE_ BASE_FORMAT_ X8S8S8U8 | SCE_GXM_TEXTURE_ SWIZZLE3_RGB	The U8U8S8S8 data is read in ARGB order from memory, A is replaced with 0xff.
SCE_GXM_TEXTURE_ FORMAT_X8U24_SD	SCE_GXM_TEXTURE_ BASE_FORMAT_ X8U24 | SCE_GXM_TEXTURE_ SWIZZLE2_SD	The U8U24 data is read in SD order, D is returned as a single component result.
SCE_GXM_TEXTURE_ FORMAT_U24X8_DS	SCE_GXM_TEXTURE_ BASE_FORMAT_ X8U24 | SCE_GXM_TEXTURE_ SWIZZLE2_DS	The U24U8 data is read in DS order, D is returned as a single component result.
SCE_GXM_TEXTURE_ FORMAT_U32_000R	SCE_GXM_TEXTURE_ BASE_FORMAT_U32 | SCE_GXM_TEXTURE_ SWIZZLE1_000R	The U32 value is swizzled to 000R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U32_111R	SCE_GXM_TEXTURE_ BASE_FORMAT_U32 | SCE_GXM_TEXTURE_ SWIZZLE1_111R	The U32 value is swizzled to 111R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U32_RRRR	SCE_GXM_TEXTURE_ BASE_FORMAT_U32 | SCE_GXM_TEXTURE_ SWIZZLE1_RRRR	The U32 value is swizzled to RRRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U32_0RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_U32 | SCE_GXM_TEXTURE_ SWIZZLE1_0RRR	The U32 value is swizzled to 0RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U32_1RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_U32 | SCE_GXM_TEXTURE_ SWIZZLE1_1RRR	The U32 value is swizzled to 1RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U32_R000	SCE_GXM_TEXTURE_ BASE_FORMAT_U32 | SCE_GXM_TEXTURE_ SWIZZLE1_R000	The U32 value is swizzled to R000 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U32_R111	SCE_GXM_TEXTURE_ BASE_FORMAT_U32 | SCE_GXM_TEXTURE_ SWIZZLE1_R111	The U32 value is swizzled to R111 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U32_R	SCE_GXM_TEXTURE_ BASE_FORMAT_U32 | SCE_GXM_TEXTURE_ SWIZZLE1_R	The U32 value is returned as a single component result.
SCE_GXM_TEXTURE_ FORMAT_S32_000R	SCE_GXM_TEXTURE_ BASE_FORMAT_S32 | SCE_GXM_TEXTURE_ SWIZZLE1_000R	The S32 value is swizzled to 000R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S32_111R	SCE_GXM_TEXTURE_ BASE_FORMAT_S32 | SCE_GXM_TEXTURE_ SWIZZLE1_111R	The S32 value is swizzled to 111R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S32_RRRR	SCE_GXM_TEXTURE_ BASE_FORMAT_S32 | SCE_GXM_TEXTURE_ SWIZZLE1_RRRR	The S32 value is swizzled to RRRR (in ABGR notation).

Macro	Value	Description
SCE_GXM_TEXTURE_ FORMAT_S32_0RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_S32 | SCE_GXM_TEXTURE_ SWIZZLE1_0RRR	The S32 value is swizzled to 0RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S32_1RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_S32 | SCE_GXM_TEXTURE_ SWIZZLE1_1RRR	The S32 value is swizzled to 1RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S32_R000	SCE_GXM_TEXTURE_ BASE_FORMAT_S32 | SCE_GXM_TEXTURE_ SWIZZLE1_R000	The S32 value is swizzled to R000 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S32_R111	SCE_GXM_TEXTURE_ BASE_FORMAT_S32 | SCE_GXM_TEXTURE_ SWIZZLE1_R111	The S32 value is swizzled to R111 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_S32_R	SCE_GXM_TEXTURE_ BASE_FORMAT_S32 | SCE_GXM_TEXTURE_ SWIZZLE1_R	The S32 value is returned as a single component result.
SCE_GXM_TEXTURE_ FORMAT_SE5M9M9M9_ BGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ SE5M9M9M9 | SCE_GXM_TEXTURE_ SWIZZLE3_BGR	The SE5M9M9M9 data is read in EBGR order from memory, A is implicit and assigned 1.
SCE_GXM_TEXTURE_ FORMAT_SE5M9M9M9_ RGB	SCE_GXM_TEXTURE_ BASE_FORMAT_ SE5M9M9M9 | SCE_GXM_TEXTURE_ SWIZZLE3_RGB	The SE5M9M9M9 data is read in ERGB order from memory, A is implicit and assigned 1.
SCE_GXM_TEXTURE_ FORMAT_F10F11F11_ BGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ F11F11F10 | SCE_GXM_TEXTURE_ SWIZZLE3_BGR	The F10F11F11 data is read in BGR order from memory, A is implicit and assigned 1.
SCE_GXM_TEXTURE_ FORMAT_F11F11F10_ RGB	SCE_GXM_TEXTURE_ BASE_FORMAT_ F11F11F10 | SCE_GXM_TEXTURE_ SWIZZLE3_RGB	The F11F11F10 data is read in RGB order from memory, A is implicit and assigned 1.
SCE_GXM_TEXTURE_ FORMAT_ F16F16F16F16_ ABGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ F16F16F16F16 | SCE_GXM_TEXTURE_ SWIZZLE4_ABGR	The F16F16F16F16 data is read in ABGR order from memory.
SCE_GXM_TEXTURE_ FORMAT_ F16F16F16F16_ ARGB	SCE_GXM_TEXTURE_ BASE_FORMAT_ F16F16F16F16 | SCE_GXM_TEXTURE_ SWIZZLE4_ARGB	The F16F16F16F16 data is read in ARGB order from memory.
SCE_GXM_TEXTURE_ FORMAT_ F16F16F16F16_ RGBA	SCE_GXM_TEXTURE_ BASE_FORMAT_ F16F16F16F16 | SCE_GXM_TEXTURE_ SWIZZLE4_RGBA	The F16F16F16F16 data is read in RGBA order from memory.
SCE_GXM_TEXTURE_ FORMAT_ F16F16F16F16_ BGRA	SCE_GXM_TEXTURE_ BASE_FORMAT_ F16F16F16F16 | SCE_GXM_TEXTURE_ SWIZZLE4_BGRA	The F16F16F16F16 data is read in BGRA order from memory.

Macro	Value	Description
SCE_GXM_TEXTURE_FORMAT_X16F16F16F16_1BGR	SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR	The F16F16F16F16 data is read in ABGR order from memory, A is replaced with 0x3c00.
SCE_GXM_TEXTURE_FORMAT_X16F16F16F16_1RGB	SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB	The F16F16F16F16 data is read in ARGB order from memory, A is replaced with 0x3c00.
SCE_GXM_TEXTURE_FORMAT_F16F16F16X16_RGB1	SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1	The F16F16F16F16 data is read in RGBA order from memory, A is replaced with 0x3c00.
SCE_GXM_TEXTURE_FORMAT_F16F16F16X16_BGR1	SCE_GXM_TEXTURE_BASE_FORMAT_F16F16F16F16 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1	The F16F16F16F16 data is read in BGRA order from memory, A is replaced with 0x3c00.
SCE_GXM_TEXTURE_FORMAT_U16U16U16U16_ABGR	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR	The U16U16U16U16 data is read in ABGR order from memory.
SCE_GXM_TEXTURE_FORMAT_U16U16U16U16_ARGB	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB	The U16U16U16U16 data is read in ARGB order from memory.
SCE_GXM_TEXTURE_FORMAT_U16U16U16U16_RGBA	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA	The U16U16U16U16 data is read in RGBA order from memory.
SCE_GXM_TEXTURE_FORMAT_U16U16U16U16_BGRA	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA	The U16U16U16U16 data is read in BGRA order from memory.
SCE_GXM_TEXTURE_FORMAT_X16U16U16U16_1BGR	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR	The U16U16U16U16 data is read in ABGR order from memory, A is replaced with 0xffff.
SCE_GXM_TEXTURE_FORMAT_X16U16U16U16_1RGB	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB	The U16U16U16U16 data is read in ARGB order from memory, A is replaced with 0xffff.
SCE_GXM_TEXTURE_FORMAT_U16U16U16X16_RGB1	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1	The U16U16U16U16 data is read in RGBA order from memory, A is replaced with 0xffff.
SCE_GXM_TEXTURE_FORMAT_U16U16U16X16_BGR1	SCE_GXM_TEXTURE_BASE_FORMAT_U16U16U16U16 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1	The U16U16U16U16 data is read in BGRA order from memory, A is replaced with 0xffff.

Macro	Value	Description
SCE_GXM_TEXTURE_FORMAT_S16S16S16S16_ABGR	SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR	The S16S16S16S16 data is read in ABGR order from memory.
SCE_GXM_TEXTURE_FORMAT_S16S16S16S16_ARGB	SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB	The S16S16S16S16 data is read in ARGB order from memory.
SCE_GXM_TEXTURE_FORMAT_S16S16S16S16_RGBA	SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA	The S16S16S16S16 data is read in RGBA order from memory.
SCE_GXM_TEXTURE_FORMAT_S16S16S16S16_BGRA	SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA	The S16S16S16S16 data is read in BGRA order from memory.
SCE_GXM_TEXTURE_FORMAT_X16S16S16S16_1BGR	SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR	The S16S16S16S16 data is read in ABGR order from memory, A is replaced with 0x7fff.
SCE_GXM_TEXTURE_FORMAT_X16S16S16S16_1RGB	SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB	The S16S16S16S16 data is read in ARGB order from memory, A is replaced with 0x7fff.
SCE_GXM_TEXTURE_FORMAT_S16S16S16X16_RGB1	SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1	The S16S16S16S16 data is read in RGBA order from memory, A is replaced with 0x7fff.
SCE_GXM_TEXTURE_FORMAT_S16S16S16X16_BGR1	SCE_GXM_TEXTURE_BASE_FORMAT_S16S16S16S16 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1	The S16S16S16S16 data is read in BGRA order from memory, A is replaced with 0x7fff.
SCE_GXM_TEXTURE_FORMAT_F32F32_00GR	SCE_GXM_TEXTURE_BASE_FORMAT_F32F32 | SCE_GXM_TEXTURE_SWIZZLE2_00GR	The F32F32 (GR) value is swizzled to 00GR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_F32F32_GRRR	SCE_GXM_TEXTURE_BASE_FORMAT_F32F32 | SCE_GXM_TEXTURE_SWIZZLE2_GRRR	The F32F32 (GR) value is swizzled to GRRR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_F32F32_RGGG	SCE_GXM_TEXTURE_BASE_FORMAT_F32F32 | SCE_GXM_TEXTURE_SWIZZLE2_RGGG	The F32F32 (GR) value is swizzled to RGGG (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_F32F32_GRGR	SCE_GXM_TEXTURE_BASE_FORMAT_F32F32 | SCE_GXM_TEXTURE_SWIZZLE2_GRGR	The F32F32 (GR) value is swizzled to GRGR (in ABGR notation).

Macro	Value	Description
SCE_GXM_TEXTURE_ FORMAT_F32F32_ 00RG	SCE_GXM_TEXTURE_ BASE_FORMAT_ F32F32 | SCE_GXM_ TEXTURE_ SWIZZLE2_00RG	The F32F32 (GR) value is swizzled to 00RG (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_F32F32_GR	SCE_GXM_TEXTURE_ BASE_FORMAT_ F32F32 | SCE_GXM_ TEXTURE_ SWIZZLE2_GR	The F32F32 (GR) value is returned as a 2-component result.
SCE_GXM_TEXTURE_ FORMAT_U32U32_ 00GR	SCE_GXM_TEXTURE_ BASE_FORMAT_ U32U32 | SCE_GXM_ TEXTURE_ SWIZZLE2_00GR	The U32U32 (GR) value is swizzled to 00GR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U32U32_ GRRR	SCE_GXM_TEXTURE_ BASE_FORMAT_ U32U32 | SCE_GXM_ TEXTURE_ SWIZZLE2_GRRR	The U32U32 (GR) value is swizzled to GRRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U32U32_ RGGG	SCE_GXM_TEXTURE_ BASE_FORMAT_ U32U32 | SCE_GXM_ TEXTURE_ SWIZZLE2_RGGG	The U32U32 (GR) value is swizzled to RGGG (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U32U32_ GRGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ U32U32 | SCE_GXM_ TEXTURE_ SWIZZLE2_GRGR	The U32U32 (GR) value is swizzled to GRGR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U32U32_ 00RG	SCE_GXM_TEXTURE_ BASE_FORMAT_ U32U32 | SCE_GXM_TEXTURE_ SWIZZLE2_00RG	The U32U32 (GR) value is swizzled to 00RG (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_U32U32_GR	SCE_GXM_TEXTURE_ BASE_FORMAT_ U32U32 | SCE_GXM_ TEXTURE_ SWIZZLE2_GR	The U32U32 (GR) value is returned as a 2-component result.
SCE_GXM_TEXTURE_ FORMAT_PVRT2BPP_ ABGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ PVRT2BPP | SCE_GXM_TEXTURE_ SWIZZLE4_ABGR	The PVRT2BPP data is decoded into ABGR.
SCE_GXM_TEXTURE_ FORMAT_PVRT2BPP_ 1BGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ PVRT2BPP | SCE_GXM_TEXTURE_ SWIZZLE4_1BGR	The PVRT2BPP data is decoded into ABGR, A is replaced with 0xff.
SCE_GXM_TEXTURE_ FORMAT_PVRT4BPP_ ABGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ PVRT4BPP | SCE_GXM_TEXTURE_ SWIZZLE4_ABGR	The PVRT4BPP data is decoded into ABGR.
SCE_GXM_TEXTURE_ FORMAT_PVRT4BPP_ 1BGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ PVRT4BPP | SCE_GXM_TEXTURE_ SWIZZLE4_1BGR	The PVRT4BPP data is decoded into ABGR, A is replaced with 0xff.

Macro	Value	Description
SCE_GXM_TEXTURE_FORMAT_PVRTII2BPP_ABGR	SCE_GXM_TEXTURE_BASE_FORMAT_PVRTII2BPP | SCE_GXM_TEXTURE_SWIZZLE4_ABGR	The PVRTII2BPP data is decoded into ABGR.
SCE_GXM_TEXTURE_FORMAT_PVRTII2BPP_1BGR	SCE_GXM_TEXTURE_BASE_FORMAT_PVRTII2BPP | SCE_GXM_TEXTURE_SWIZZLE4_1BGR	The PVRTII2BPP data is decoded into ABGR, A is replaced with 0xff.
SCE_GXM_TEXTURE_FORMAT_PVRTII4BPP_ABGR	SCE_GXM_TEXTURE_BASE_FORMAT_PVRTII4BPP | SCE_GXM_TEXTURE_SWIZZLE4_ABGR	The PVRTII4BPP data is decoded into ABGR.
SCE_GXM_TEXTURE_FORMAT_PVRTII4BPP_1BGR	SCE_GXM_TEXTURE_BASE_FORMAT_PVRTII4BPP | SCE_GXM_TEXTURE_SWIZZLE4_1BGR	The PVRTII4BPP data is decoded into ABGR, A is replaced with 0xff.
SCE_GXM_TEXTURE_FORMAT_UBC1_ABGR	SCE_GXM_TEXTURE_BASE_FORMAT_UBC1 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR	The UBC1 data is decoded into ABGR.
SCE_GXM_TEXTURE_FORMAT_UBC1_1BGR	SCE_GXM_TEXTURE_BASE_FORMAT_UBC1 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR	The UBC1 data is decoded into ABGR, A is replaced with 0xff.
SCE_GXM_TEXTURE_FORMAT_UBC2_ABGR	SCE_GXM_TEXTURE_BASE_FORMAT_UBC2 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR	The UBC2 data is decoded into ABGR.
SCE_GXM_TEXTURE_FORMAT_UBC2_1BGR	SCE_GXM_TEXTURE_BASE_FORMAT_UBC2 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR	The UBC2 data is decoded into ABGR, A is replaced with 0xff.
SCE_GXM_TEXTURE_FORMAT_UBC3_ABGR	SCE_GXM_TEXTURE_BASE_FORMAT_UBC3 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR	The UBC3 data is decoded into ABGR.
SCE_GXM_TEXTURE_FORMAT_UBC3_1BGR	SCE_GXM_TEXTURE_BASE_FORMAT_UBC3 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR	The UBC3 data is decoded into ABGR, A is replaced with 0xff.
SCE_GXM_TEXTURE_FORMAT_UBC4_000R	SCE_GXM_TEXTURE_BASE_FORMAT_UBC4 | SCE_GXM_TEXTURE_SWIZZLE1_000R	The decoded UBC4 (R) value is swizzled to 000R (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_UBC4_111R	SCE_GXM_TEXTURE_BASE_FORMAT_UBC4 | SCE_GXM_TEXTURE_SWIZZLE1_111R	The decoded UBC4 (R) value is swizzled to 111R (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_UBC4_RRRR	SCE_GXM_TEXTURE_BASE_FORMAT_UBC4 | SCE_GXM_TEXTURE_SWIZZLE1_RRRR	The decoded UBC4 (R) value is swizzled to RRRR (in ABGR notation).
SCE_GXM_TEXTURE_FORMAT_UBC4_0RRR	SCE_GXM_TEXTURE_BASE_FORMAT_UBC4 | SCE_GXM_TEXTURE_SWIZZLE1_0RRR	The decoded UBC4 (R) value is swizzled to 0RRR (in ABGR notation).

Macro	Value	Description
SCE_GXM_TEXTURE_ FORMAT_UBC4_1RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_UBC4 | SCE_GXM_TEXTURE_ SWIZZLE1_1RRR	The decoded UBC4 (R) value is swizzled to 1RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_UBC4_R000	SCE_GXM_TEXTURE_ BASE_FORMAT_UBC4 | SCE_GXM_TEXTURE_ SWIZZLE1_R000	The decoded UBC4 (R) value is swizzled to R000 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_UBC4_R111	SCE_GXM_TEXTURE_ BASE_FORMAT_UBC4 | SCE_GXM_TEXTURE_ SWIZZLE1_R111	The decoded UBC4 (R) value is swizzled to R111 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_UBC4_R	SCE_GXM_TEXTURE_ BASE_FORMAT_UBC4 | SCE_GXM_TEXTURE_ SWIZZLE1_R	The decoded UBC4 (R) value is returned as a single component result.
SCE_GXM_TEXTURE_ FORMAT_SBC4_000R	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC4 | SCE_GXM_TEXTURE_ SWIZZLE1_000R	The decoded SBC4 (R) value is swizzled to 000R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_SBC4_111R	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC4 | SCE_GXM_TEXTURE_ SWIZZLE1_111R	The decoded SBC4 (R) value is swizzled to 111R (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_SBC4_RRRR	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC4 | SCE_GXM_TEXTURE_ SWIZZLE1_RRRR	The decoded SBC4 (R) value is swizzled to RRRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_SBC4_0RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC4 | SCE_GXM_TEXTURE_ SWIZZLE1_0RRR	The decoded SBC4 (R) value is swizzled to 0RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_SBC4_1RRR	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC4 | SCE_GXM_TEXTURE_ SWIZZLE1_1RRR	The decoded SBC4 (R) value is swizzled to 1RRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_SBC4_R000	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC4 | SCE_GXM_TEXTURE_ SWIZZLE1_R000	The decoded SBC4 (R) value is swizzled to R000 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_SBC4_R111	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC4 | SCE_GXM_TEXTURE_ SWIZZLE1_R111	The decoded SBC4 (R) value is swizzled to R111 (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_SBC4_R	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC4 | SCE_GXM_TEXTURE_ SWIZZLE1_R	The decoded SBC4 (R) value is returned as a single component result.
SCE_GXM_TEXTURE_ FORMAT_UBC5_00GR	SCE_GXM_TEXTURE_ BASE_FORMAT_UBC5 | SCE_GXM_TEXTURE_ SWIZZLE2_00GR	The decoded UBC5 (GR) value is swizzled to 00GR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_UBC5_GRRR	SCE_GXM_TEXTURE_ BASE_FORMAT_UBC5 | SCE_GXM_TEXTURE_ SWIZZLE2_GRRR	The decoded UBC5 (GR) value is swizzled to GRRR (in ABGR notation).

Macro	Value	Description
SCE_GXM_TEXTURE_ FORMAT_UBC5_RGGG	SCE_GXM_TEXTURE_ BASE_FORMAT_ UBC5 | SCE_GXM_ TEXTURE_ SWIZZLE2_RGGG	The decoded UBC5 (GR) value is swizzled to RGGG (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_UBC5_GRGR	SCE_GXM_TEXTURE_ BASE_FORMAT_ UBC5 | SCE_GXM_ TEXTURE_ SWIZZLE2_GRGR	The decoded UBC5 (GR) value is swizzled to GRGR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_UBC5_00RG	SCE_GXM_TEXTURE_ BASE_FORMAT_UBC5 | SCE_GXM_TEXTURE_ SWIZZLE2_00RG	The decoded UBC5 (GR) value is swizzled to 00RG (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_UBC5_GR	SCE_GXM_TEXTURE_ BASE_FORMAT_UBC5 | SCE_GXM_TEXTURE_ SWIZZLE2_GR	The decoded UBC5 (GR) value is returned as a 2-component result.
SCE_GXM_TEXTURE_ FORMAT_SBC5_00GR	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC5 | SCE_GXM_TEXTURE_ SWIZZLE2_00GR	The decoded SBC5 (GR) value is swizzled to 00GR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_SBC5_GRRR	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC5 | SCE_GXM_TEXTURE_ SWIZZLE2_GRRR	The decoded SBC5 (GR) value is swizzled to GRRR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_SBC5_RGGG	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC5 | SCE_GXM_TEXTURE_ SWIZZLE2_RGGG	The decoded SBC5 (GR) value is swizzled to RGGG (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_SBC5_GRGR	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC5 | SCE_GXM_TEXTURE_ SWIZZLE2_GRGR	The decoded SBC5 (GR) value is swizzled to GRGR (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_SBC5_00RG	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC5 | SCE_GXM_TEXTURE_ SWIZZLE2_00RG	The decoded SBC5 (GR) value is swizzled to 00RG (in ABGR notation).
SCE_GXM_TEXTURE_ FORMAT_SBC5_GR	SCE_GXM_TEXTURE_ BASE_FORMAT_SBC5 | SCE_GXM_TEXTURE_ SWIZZLE2_GR	The decoded SBC5 (GR) value is returned as a 2-component result.
SCE_GXM_TEXTURE_ FORMAT_YUV420P2_ CSC0	SCE_GXM_TEXTURE_ BASE_FORMAT_ YUV420P2 | SCE_GXM_TEXTURE_ SWIZZLE_YUV_CSC0	The Y plane and UV plane is converted to ABGR using CSC matrix 0, A is implicit and assigned 1.
SCE_GXM_TEXTURE_ FORMAT_YVU420P2_ CSC0	SCE_GXM_TEXTURE_ BASE_FORMAT_ YUV420P2 | SCE_GXM_TEXTURE_ SWIZZLE_YVU_CSC0	The Y plane and VU plane is converted to ABGR using CSC matrix 0, A is implicit and assigned 1.
SCE_GXM_TEXTURE_ FORMAT_YUV420P2_ CSC1	SCE_GXM_TEXTURE_ BASE_FORMAT_ YUV420P2 | SCE_GXM_TEXTURE_ SWIZZLE_YUV_CSC1	The Y plane and UV plane is converted to ABGR using CSC matrix 1, A is implicit and assigned 1.

Macro	Value	Description
SCE_GXM_TEXTURE_FORMAT_YVU420P2_CSC1	SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P2 | SCE_GXM_TEXTURE_SWIZZLE_YVU_CSC1	The Y plane and VU plane is converted to ABGR using CSC matrix 1, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_YUV420P3_CSC0	SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P3 | SCE_GXM_TEXTURE_SWIZZLE_YUV_CSC0	The Y plane, U plane and V plane is converted to ABGR using CSC matrix 0, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_YVU420P3_CSC0	SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P3 | SCE_GXM_TEXTURE_SWIZZLE_YVU_CSC0	The Y plane, V plane and U plane is converted to ABGR using CSC matrix 0, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_YUV420P3_CSC1	SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P3 | SCE_GXM_TEXTURE_SWIZZLE_YUV_CSC1	The Y plane, U plane and V plane is converted to ABGR using CSC matrix 1, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_YVU420P3_CSC1	SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P3 | SCE_GXM_TEXTURE_SWIZZLE_YVU_CSC1	The Y plane, V plane and U plane is converted to ABGR using CSC matrix 1, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_YUYV422_CSC0	SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 | SCE_GXM_TEXTURE_SWIZZLE_YUYV_CSC0	Packed YUYV 2-pixel blocks are converted to ABGR using CSC matrix 0, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_YVYU422_CSC0	SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 | SCE_GXM_TEXTURE_SWIZZLE_YVYU_CSC0	Packed YVYU 2-pixel blocks are converted to ABGR using CSC matrix 0, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_UYVY422_CSC0	SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 | SCE_GXM_TEXTURE_SWIZZLE_UYVY_CSC0	Packed UYVY 2-pixel blocks are converted to ABGR using CSC matrix 0, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_VYUY422_CSC0	SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 | SCE_GXM_TEXTURE_SWIZZLE_VYUY_CSC0	Packed VYUY 2-pixel blocks are converted to ABGR using CSC matrix 0, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_YUYV422_CSC1	SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 | SCE_GXM_TEXTURE_SWIZZLE_YUYV_CSC1	Packed YUYV 2-pixel blocks are converted to ABGR using CSC matrix 1, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_YVYU422_CSC1	SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 | SCE_GXM_TEXTURE_SWIZZLE_YVYU_CSC1	Packed YVYU 2-pixel blocks are converted to ABGR using CSC matrix 1, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_UYVY422_CSC1	SCE_GXM_TEXTURE_BASE_FORMAT_YUV422 | SCE_GXM_TEXTURE_SWIZZLE_UYVY_CSC1	Packed UYVY 2-pixel blocks are converted to ABGR using CSC matrix 1, A is implicit and assigned 1.

Macro	Value	Description
SCE_GXM_TEXTURE_ FORMAT_VYUY422_ CSC1	SCE_GXM_TEXTURE_ BASE_FORMAT_ YUV422 | SCE_GXM_ TEXTURE_ SWIZZLE_VYUY_CSC1	Packed VYUY 2-pixel blocks are converted to ABGR using CSC matrix 1, A is implicit and assigned 1.
SCE_GXM_TEXTURE_ FORMAT_P4_ABGR	SCE_GXM_TEXTURE_ BASE_FORMAT_P4 | SCE_GXM_TEXTURE_ SWIZZLE4_ABGR	The U4 index looks up into a palette of U8U8U8U8 data read in ABGR order from memory.
SCE_GXM_TEXTURE_ FORMAT_P4_ARGB	SCE_GXM_TEXTURE_ BASE_FORMAT_P4 | SCE_GXM_TEXTURE_ SWIZZLE4_ARGB	The U4 index looks up into a palette of U8U8U8U8 data read in ARGB order from memory.
SCE_GXM_TEXTURE_ FORMAT_P4_RGBA	SCE_GXM_TEXTURE_ BASE_FORMAT_P4 | SCE_GXM_TEXTURE_ SWIZZLE4_RGBA	The U4 index looks up into a palette of U8U8U8U8 data read in RGBA order from memory.
SCE_GXM_TEXTURE_ FORMAT_P4_BGRA	SCE_GXM_TEXTURE_ BASE_FORMAT_P4 | SCE_GXM_TEXTURE_ SWIZZLE4_BGRA	The U4 index looks up into a palette of U8U8U8U8 data read in BGRA order from memory.
SCE_GXM_TEXTURE_ FORMAT_P4_1BGR	SCE_GXM_TEXTURE_ BASE_FORMAT_P4 | SCE_GXM_TEXTURE_ SWIZZLE4_1BGR	The U4 index looks up into a palette of U8U8U8U8 data read in ABGR order from memory, A is replaced with 0xff.
SCE_GXM_TEXTURE_ FORMAT_P4_1RGB	SCE_GXM_TEXTURE_ BASE_FORMAT_P4 | SCE_GXM_TEXTURE_ SWIZZLE4_1RGB	The U4 index looks up into a palette of U8U8U8U8 data read in ARGB order from memory, A is replaced with 0xff.
SCE_GXM_TEXTURE_ FORMAT_P4_RGB1	SCE_GXM_TEXTURE_ BASE_FORMAT_P4 | SCE_GXM_TEXTURE_ SWIZZLE4_RGB1	The U4 index looks up into a palette of U8U8U8U8 data read in RGBA order from memory, A is replaced with 0xff.
SCE_GXM_TEXTURE_ FORMAT_P4_BGR1	SCE_GXM_TEXTURE_ BASE_FORMAT_P4 | SCE_GXM_TEXTURE_ SWIZZLE4_BGR1	The U4 index looks up into a palette of U8U8U8U8 data read in BGRA order from memory, A is replaced with 0xff.
SCE_GXM_TEXTURE_ FORMAT_P8_ABGR	SCE_GXM_TEXTURE_ BASE_FORMAT_P8 | SCE_GXM_TEXTURE_ SWIZZLE4_ABGR	The U8 index looks up into a palette of U8U8U8U8 data read in ABGR order from memory.
SCE_GXM_TEXTURE_ FORMAT_P8_ARGB	SCE_GXM_TEXTURE_ BASE_FORMAT_P8 | SCE_GXM_TEXTURE_ SWIZZLE4_ARGB	The U8 index looks up into a palette of U8U8U8U8 data read in ARGB order from memory.
SCE_GXM_TEXTURE_ FORMAT_P8_RGBA	SCE_GXM_TEXTURE_ BASE_FORMAT_P8 | SCE_GXM_TEXTURE_ SWIZZLE4_RGBA	The U8 index looks up into a palette of U8U8U8U8 data read in RGBA order from memory.
SCE_GXM_TEXTURE_ FORMAT_P8_BGRA	SCE_GXM_TEXTURE_ BASE_FORMAT_P8 | SCE_GXM_TEXTURE_ SWIZZLE4_BGRA	The U8 index looks up into a palette of U8U8U8U8 data read in BGRA order from memory.
SCE_GXM_TEXTURE_ FORMAT_P8_1BGR	SCE_GXM_TEXTURE_ BASE_FORMAT_P8 | SCE_GXM_TEXTURE_ SWIZZLE4_1BGR	The U8 index looks up into a palette of U8U8U8U8 data read in ABGR order from memory, A is replaced with 0xff.

Macro	Value	Description
SCE_GXM_TEXTURE_FORMAT_P8_1RGB	SCE_GXM_TEXTURE_BASE_FORMAT_P8 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB	The U8 index looks up into a palette of U8U8U8U8 data read in ARGB order from memory, A is replaced with 0xff.
SCE_GXM_TEXTURE_FORMAT_P8_RGB1	SCE_GXM_TEXTURE_BASE_FORMAT_P8 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1	The U8 index looks up into a palette of U8U8U8U8 data read in RGBA order from memory, A is replaced with 0xff.
SCE_GXM_TEXTURE_FORMAT_P8_BGR1	SCE_GXM_TEXTURE_BASE_FORMAT_P8 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1	The U8 index looks up into a palette of U8U8U8U8 data read in BGRA order from memory, A is replaced with 0xff.
SCE_GXM_TEXTURE_FORMAT_U8U8U8_BGR	SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8 | SCE_GXM_TEXTURE_SWIZZLE3_BGR	The packed 24-bit U8U8U8 data is read in BGR order from memory, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_U8U8U8_RGB	SCE_GXM_TEXTURE_BASE_FORMAT_U8U8U8 | SCE_GXM_TEXTURE_SWIZZLE3_RGB	The packed 24-bit U8U8U8 data is read in RGB order from memory, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_S8S8S8_BGR	SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE3_BGR	The packed 24-bit S8S8S8 data is read in BGR order from memory, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_S8S8S8_RGB	SCE_GXM_TEXTURE_BASE_FORMAT_S8S8S8 | SCE_GXM_TEXTURE_SWIZZLE3_RGB	The packed 24-bit S8S8S8 data is read in RGB order from memory, A is implicit and assigned 1.
SCE_GXM_TEXTURE_FORMAT_U2F10F10F10_ABGR	SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_ABGR	The U2F10F10F10 data is read in ABGR order from memory.
SCE_GXM_TEXTURE_FORMAT_U2F10F10F10_ARGB	SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_ARGB	The U2F10F10F10 data is read in ARGB order from memory.
SCE_GXM_TEXTURE_FORMAT_F10F10F10U2_RGBA	SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_RGBA	The F10F10F10U2 data is read in RGBA order from memory.
SCE_GXM_TEXTURE_FORMAT_F10F10F10U2_BGRA	SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_BGRA	The F10F10F10U2 data is read in BGRA order from memory.
SCE_GXM_TEXTURE_FORMAT_X2F10F10F10_1BGR	SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_1BGR	The U2F10F10F10 data is read in ABGR order from memory, A is replaced with 0x3.

Macro	Value	Description
SCE_GXM_TEXTURE_FORMAT_X2F10F10F10_1RGB	SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_1RGB	The U2F10F10F10 data is read in ARGB order from memory, A is replaced with 0x3.
SCE_GXM_TEXTURE_FORMAT_F10F10F10X2_RGB1	SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_RGB1	The F10F10F10U2 data is read in RGBA order from memory, A is replaced with 0x3.
SCE_GXM_TEXTURE_FORMAT_F10F10F10X2_BGR1	SCE_GXM_TEXTURE_BASE_FORMAT_U2F10F10F10 | SCE_GXM_TEXTURE_SWIZZLE4_BGR1	The F10F10F10U2 data is read in BGRA order from memory, A is replaced with 0x3.
SCE_GXM_TEXTURE_FORMAT_L8	SCE_GXM_TEXTURE_FORMAT_U8_1RRR	Legacy name for SCE_GXM_TEXTURE_FORMAT_U8_1RRR.
SCE_GXM_TEXTURE_FORMAT_A8	SCE_GXM_TEXTURE_FORMAT_U8_R000	Legacy name for SCE_GXM_TEXTURE_FORMAT_U8_R000.
SCE_GXM_TEXTURE_FORMAT_R8	SCE_GXM_TEXTURE_FORMAT_U8_000R	Legacy name for SCE_GXM_TEXTURE_FORMAT_U8_000R.
SCE_GXM_TEXTURE_FORMAT_A4R4G4B4	SCE_GXM_TEXTURE_FORMAT_U4U4U4U4_ARGB	Legacy name for SCE_GXM_TEXTURE_FORMAT_U4U4U4U4_ARGB.
SCE_GXM_TEXTURE_FORMAT_A1R5G5B5	SCE_GXM_TEXTURE_FORMAT_U1U5U5U5_ARGB	Legacy name for SCE_GXM_TEXTURE_FORMAT_U1U5U5U5_ARGB.
SCE_GXM_TEXTURE_FORMAT_R5G6B5	SCE_GXM_TEXTURE_FORMAT_U5U6U5_RGB	Legacy name for SCE_GXM_TEXTURE_FORMAT_U5U6U5_RGB.
SCE_GXM_TEXTURE_FORMAT_A8L8	SCE_GXM_TEXTURE_FORMAT_U8U8_GRRR	Legacy name for SCE_GXM_TEXTURE_FORMAT_U8U8_GRRR.
SCE_GXM_TEXTURE_FORMAT_L8A8	SCE_GXM_TEXTURE_FORMAT_U8U8_RGGG	Legacy name for SCE_GXM_TEXTURE_FORMAT_U8U8_RGGG.
SCE_GXM_TEXTURE_FORMAT_G8R8	SCE_GXM_TEXTURE_FORMAT_U8U8_00GR	Legacy name for SCE_GXM_TEXTURE_FORMAT_U8U8_00GR.
SCE_GXM_TEXTURE_FORMAT_L16	SCE_GXM_TEXTURE_FORMAT_U16_1RRR	Legacy name for SCE_GXM_TEXTURE_FORMAT_U16_1RRR.
SCE_GXM_TEXTURE_FORMAT_A16	SCE_GXM_TEXTURE_FORMAT_U16_R000	Legacy name for SCE_GXM_TEXTURE_FORMAT_U16_R000.
SCE_GXM_TEXTURE_FORMAT_R16	SCE_GXM_TEXTURE_FORMAT_U16_000R	Legacy name for SCE_GXM_TEXTURE_FORMAT_U16_000R.
SCE_GXM_TEXTURE_FORMAT_D16	SCE_GXM_TEXTURE_FORMAT_U16_R	Legacy name for SCE_GXM_TEXTURE_FORMAT_U16_R.
SCE_GXM_TEXTURE_FORMAT_LF16	SCE_GXM_TEXTURE_FORMAT_F16_1RRR	Legacy name for SCE_GXM_TEXTURE_FORMAT_F16_1RRR.
SCE_GXM_TEXTURE_FORMAT_AF16	SCE_GXM_TEXTURE_FORMAT_F16_R000	Legacy name for SCE_GXM_TEXTURE_FORMAT_F16_R000.
SCE_GXM_TEXTURE_FORMAT_RF16	SCE_GXM_TEXTURE_FORMAT_F16_000R	Legacy name for SCE_GXM_TEXTURE_FORMAT_F16_000R.
SCE_GXM_TEXTURE_FORMAT_A8R8G8B8	SCE_GXM_TEXTURE_FORMAT_U8U8U8U8_ARGB	Legacy name for SCE_GXM_TEXTURE_FORMAT_U8U8U8U8_ARGB.
SCE_GXM_TEXTURE_FORMAT_A8B8G8R8	SCE_GXM_TEXTURE_FORMAT_U8U8U8U8_ABGR	Legacy name for SCE_GXM_TEXTURE_FORMAT_U8U8U8U8_ABGR.

Macro	Value	Description
SCE_GXM_TEXTURE_FORMAT_AF16LF16	SCE_GXM_TEXTURE_FORMAT_F16F16_GRRR	Legacy name for SCE_GXM_TEXTURE_FORMAT_F16F16_GRRR.
SCE_GXM_TEXTURE_FORMAT_LF16AF16	SCE_GXM_TEXTURE_FORMAT_F16F16_RGGG	Legacy name for SCE_GXM_TEXTURE_FORMAT_F16F16_RGGG.
SCE_GXM_TEXTURE_FORMAT_GF16RF16	SCE_GXM_TEXTURE_FORMAT_F16F16_00GR	Legacy name for SCE_GXM_TEXTURE_FORMAT_F16F16_00GR.
SCE_GXM_TEXTURE_FORMAT_LF32M	SCE_GXM_TEXTURE_FORMAT_F32M_1RRR	Legacy name for SCE_GXM_TEXTURE_FORMAT_F32M_1RRR.
SCE_GXM_TEXTURE_FORMAT_AF32M	SCE_GXM_TEXTURE_FORMAT_F32M_R000	Legacy name for SCE_GXM_TEXTURE_FORMAT_F32M_R000.
SCE_GXM_TEXTURE_FORMAT_RF32M	SCE_GXM_TEXTURE_FORMAT_F32M_000R	Legacy name for SCE_GXM_TEXTURE_FORMAT_F32M_000R.
SCE_GXM_TEXTURE_FORMAT_DF32M	SCE_GXM_TEXTURE_FORMAT_F32M_R	Legacy name for SCE_GXM_TEXTURE_FORMAT_F32M_R.
SCE_GXM_TEXTURE_FORMAT_VYUY	SCE_GXM_TEXTURE_FORMAT_VYUY422_CSC0	Legacy name for SCE_GXM_TEXTURE_FORMAT_VYUY422_CSC0.
SCE_GXM_TEXTURE_FORMAT_YVYU	SCE_GXM_TEXTURE_FORMAT_YVYU422_CSC0	Legacy name for SCE_GXM_TEXTURE_FORMAT_YVYU422_CSC0.
SCE_GXM_TEXTURE_FORMAT_UBC1	SCE_GXM_TEXTURE_FORMAT_UBC1_ABGR	Legacy name for SCE_GXM_TEXTURE_FORMAT_UBC1_ABGR.
SCE_GXM_TEXTURE_FORMAT_UBC2	SCE_GXM_TEXTURE_FORMAT_UBC2_ABGR	Legacy name for SCE_GXM_TEXTURE_FORMAT_UBC2_ABGR.
SCE_GXM_TEXTURE_FORMAT_UBC3	SCE_GXM_TEXTURE_FORMAT_UBC3_ABGR	Legacy name for SCE_GXM_TEXTURE_FORMAT_UBC3_ABGR.
SCE_GXM_TEXTURE_FORMAT_PVRT2BPP	SCE_GXM_TEXTURE_FORMAT_PVRT2BPP_ABGR	Legacy name for SCE_GXM_TEXTURE_FORMAT_PVRT2BPP_ABGR.
SCE_GXM_TEXTURE_FORMAT_PVRT4BPP	SCE_GXM_TEXTURE_FORMAT_PVRT4BPP_ABGR	Legacy name for SCE_GXM_TEXTURE_FORMAT_PVRT4BPP_ABGR.
SCE_GXM_TEXTURE_FORMAT_PVRTII2BPP	SCE_GXM_TEXTURE_FORMAT_PVRTII2BPP_ABGR	Legacy name for SCE_GXM_TEXTURE_FORMAT_PVRTII2BPP_ABGR.
SCE_GXM_TEXTURE_FORMAT_PVRTII4BPP	SCE_GXM_TEXTURE_FORMAT_PVRTII4BPP_ABGR	Legacy name for SCE_GXM_TEXTURE_FORMAT_PVRTII4BPP_ABGR.

Description

The texture formats. These are split into two sections: the full list of all texture formats supported by the hardware, followed by some legacy defines for convenience. The full list uses a standard syntax of `FORMAT_SWIZZLE`.

The format part of the name is written for high-to-low bit ordering assuming the value is in a register. Note that registers are stored in memory in a little-endian format.

For 4 and 3-component formats in memory, the swizzle part of the name is the component ordering in the value loaded from memory. For example, a texel of format SCE_GXM_TEXTURE_FORMAT_U4U4U4U4_ABGR would have A in the high 4 bits and R in the low 4 bits if the 16-bit value was loaded into a register.

For 2 and 1-component formats in memory, the format in memory is always GR or R and the swizzle represents the mapping to an ABGR (or WZYX) result in the shader code. For example, the format

SCE_GXM_TEXTURE_FORMAT_U8_000R would return zero in the ABG components and the U8 value in the R component.

The depth/stencil format X8U24 is an exception and behaves like the 4 and 3-component formats: the swizzle part of the name is the component ordering in memory, and the result is always returned with D in the x component. Because of this, the format can only be used with single-component query formats.

For a full table of all texture base formats, swizzles and supported query formats please refer to the *GPU User's Guide*, Appendix A.

SceGxmTextureGammaMode

The texture gamma mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureGammaMode {
    SCE_GXM_TEXTURE_GAMMA_NONE = 0x00000000U,
    SCE_GXM_TEXTURE_GAMMA_R = 0x08000000U,
    SCE_GXM_TEXTURE_GAMMA_GR = 0x18000000U,
    SCE_GXM_TEXTURE_GAMMA_BGR = 0x08000000U
} SceGxmTextureGammaMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TEXTURE_GAMMA_NONE`	`0x00000000U`	No gamma correction on texture read.
`SCE_GXM_TEXTURE_GAMMA_R`	`0x08000000U`	Gamma correction is performed for the R component on texture read.
`SCE_GXM_TEXTURE_GAMMA_GR`	`0x18000000U`	Gamma correction is performed for the G and R components on texture read.
`SCE_GXM_TEXTURE_GAMMA_BGR`	`0x08000000U`	Gamma correction is performed for the B, G, and R components on texture read.

Description

The texture gamma mode.

Notes

`SCE_GXM_TEXTURE_GAMMA_R` and `SCE_GXM_TEXTURE_GAMMA_BGR` enumerations intentionally share the same value. The implied meaning of the value changes depending on the texture format being used. Please see the *GPU User's Guide* for details.

SceGxmTextureMipFilter

The mipmap filter mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureMipFilter {
    SCE_GXM_TEXTURE_MIP_FILTER_DISABLED = 0x00000000U,
    SCE_GXM_TEXTURE_MIP_FILTER_ENABLED = 0x00000200U
} SceGxmTextureMipFilter;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_TEXTURE_MIP_FILTER_DISABLED	0x00000000U	Do not filter between mipmaps.
SCE_GXM_TEXTURE_MIP_FILTER_ENABLED	0x00000200U	Filter between mipmaps.

Description

The mipmap filter mode.

SceGxmTextureNormalizeMode

The texture normalize mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureNormalizeMode {
    SCE_GXM_TEXTURE_NORMALIZE_DISABLED = 0x00000000U,
    SCE_GXM_TEXTURE_NORMALIZE_ENABLED = 0x80000000U
} SceGxmTextureNormalizeMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TEXTURE_NORMALIZE_DISABLED`	`0x00000000U`	No normalization of values during integer-to-float conversion on texture read.
`SCE_GXM_TEXTURE_NORMALIZE_ENABLED`	`0x80000000U`	Normalize values during integer-to-float conversion on texture read, producing results in the range [0.0, 1.0] for unsigned data and [-1.0, 1.0] for signed data.

Description

The texture normalize mode.

SceGxmTextureSwizzle1Mode

Defines the result layout of 1-component texture formats which can be swizzled.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureSwizzle1Mode {
    SCE_GXM_TEXTURE_SWIZZLE1_R = 0x00000000U,
    SCE_GXM_TEXTURE_SWIZZLE1_000R = 0x00001000U,
    SCE_GXM_TEXTURE_SWIZZLE1_111R = 0x00002000U,
    SCE_GXM_TEXTURE_SWIZZLE1_RRRR = 0x00003000U,
    SCE_GXM_TEXTURE_SWIZZLE1_0RRR = 0x00004000U,
    SCE_GXM_TEXTURE_SWIZZLE1_1RRR = 0x00005000U,
    SCE_GXM_TEXTURE_SWIZZLE1_R000 = 0x00006000U,
    SCE_GXM_TEXTURE_SWIZZLE1_R111 = 0x00007000U
} SceGxmTextureSwizzle1Mode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TEXTURE_SWIZZLE1_R`	`0x00000000U`	Texture format is swizzled to ABGR form as XXXR (where X is undefined)
`SCE_GXM_TEXTURE_SWIZZLE1_000R`	`0x00001000U`	Texture format is swizzled to ABGR form as 000R.
`SCE_GXM_TEXTURE_SWIZZLE1_111R`	`0x00002000U`	Texture format is swizzled to ABGR form as 111R.
`SCE_GXM_TEXTURE_SWIZZLE1_RRRR`	`0x00003000U`	Texture format is swizzled to ABGR form as RRRR.
`SCE_GXM_TEXTURE_SWIZZLE1_0RRR`	`0x00004000U`	Texture format is swizzled to ABGR form as 0RRR.
`SCE_GXM_TEXTURE_SWIZZLE1_1RRR`	`0x00005000U`	Texture format is swizzled to ABGR form as 1RRR.
`SCE_GXM_TEXTURE_SWIZZLE1_R000`	`0x00006000U`	Texture format is swizzled to ABGR form as R000.
`SCE_GXM_TEXTURE_SWIZZLE1_R111`	`0x00007000U`	Texture format is swizzled to ABGR form as R111.

Description

Defines the result layout of 1-component texture formats which can be swizzled.

SceGxmTextureSwizzle2Mode

Defines the result layout of 2-component texture formats which can be swizzled.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureSwizzle2Mode {
    SCE_GXM_TEXTURE_SWIZZLE2_GR = 0x00000000U,
    SCE_GXM_TEXTURE_SWIZZLE2_00GR = 0x00001000U,
    SCE_GXM_TEXTURE_SWIZZLE2_GRRR = 0x00002000U,
    SCE_GXM_TEXTURE_SWIZZLE2_RGGG = 0x00003000U,
    SCE_GXM_TEXTURE_SWIZZLE2_GRGR = 0x00004000U,
    SCE_GXM_TEXTURE_SWIZZLE2_00RG = 0x00005000U
} SceGxmTextureSwizzle2Mode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TEXTURE_SWIZZLE2_GR`	`0x00000000U`	The texture format is swizzled to ABGR form as XXGR (where X is undefined)
`SCE_GXM_TEXTURE_SWIZZLE2_00GR`	`0x00001000U`	The texture format is swizzled to ABGR form as 00GR.
`SCE_GXM_TEXTURE_SWIZZLE2_GRRR`	`0x00002000U`	The texture format is swizzled to ABGR form as GRRR.
`SCE_GXM_TEXTURE_SWIZZLE2_RGGG`	`0x00003000U`	The texture format is swizzled to ABGR form as RGGG.
`SCE_GXM_TEXTURE_SWIZZLE2_GRGR`	`0x00004000U`	The texture format is swizzled to ABGR form as GRGR.
`SCE_GXM_TEXTURE_SWIZZLE2_00RG`	`0x00005000U`	The texture format is swizzled to ABGR form as 00RG.

Description

Defines the result layout of 2-component texture formats which can be swizzled.

SceGxmTextureSwizzle2ModeAlt

Defines the result layout of 2-component texture formats which can be swizzled.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureSwizzle2ModeAlt {
    SCE_GXM_TEXTURE_SWIZZLE2_SD = 0x00000000U,
    SCE_GXM_TEXTURE_SWIZZLE2_DS = 0x00001000U
} SceGxmTextureSwizzle2ModeAlt;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TEXTURE_SWIZZLE2_SD`	`0x00000000U`	The depth/stencil texture format read in SD order.
`SCE_GXM_TEXTURE_SWIZZLE2_DS`	`0x00001000U`	The depth/stencil texture format read in DS order.

Description

Defines the result layout of 2-component texture formats which can be swizzled.

SceGxmTextureSwizzle3Mode

Defines the layout of 3-component texture formats which can be swizzled.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureSwizzle3Mode {
    SCE_GXM_TEXTURE_SWIZZLE3_BGR = 0x00000000U,
    SCE_GXM_TEXTURE_SWIZZLE3_RGB = 0x00001000U
} SceGxmTextureSwizzle3Mode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TEXTURE_SWIZZLE3_BGR`	`0x00000000U`	Texture format read in BGR order.
`SCE_GXM_TEXTURE_SWIZZLE3_RGB`	`0x00001000U`	Texture format read in RGB order.

Description

Defines the layout of 3-component texture formats which can be swizzled.

SceGxmTextureSwizzle4Mode

Defines the layout of 4-component texture formats which can be swizzled.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureSwizzle4Mode {
    SCE_GXM_TEXTURE_SWIZZLE4_ABGR = 0x00000000U,
    SCE_GXM_TEXTURE_SWIZZLE4_ARGB = 0x00001000U,
    SCE_GXM_TEXTURE_SWIZZLE4_RGBA = 0x00002000U,
    SCE_GXM_TEXTURE_SWIZZLE4_BGRA = 0x00003000U,
    SCE_GXM_TEXTURE_SWIZZLE4_1BGR = 0x00004000U,
    SCE_GXM_TEXTURE_SWIZZLE4_1RGB = 0x00005000U,
    SCE_GXM_TEXTURE_SWIZZLE4_RGB1 = 0x00006000U,
    SCE_GXM_TEXTURE_SWIZZLE4_BGR1 = 0x00007000U
} SceGxmTextureSwizzle4Mode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TEXTURE_SWIZZLE4_ABGR`	`0x00000000U`	Texture format read in ABGR order.
`SCE_GXM_TEXTURE_SWIZZLE4_ARGB`	`0x00001000U`	Texture format read in ARGB order.
`SCE_GXM_TEXTURE_SWIZZLE4_RGBA`	`0x00002000U`	Texture format read in RGBA order.
`SCE_GXM_TEXTURE_SWIZZLE4_BGRA`	`0x00003000U`	Texture format read in BGRA order.
`SCE_GXM_TEXTURE_SWIZZLE4_1BGR`	`0x00004000U`	Texture format read in ABGR order, and A is forced to 1.0.
`SCE_GXM_TEXTURE_SWIZZLE4_1RGB`	`0x00005000U`	Texture format read in ARGB order, and A is forced to 1.0.
`SCE_GXM_TEXTURE_SWIZZLE4_RGB1`	`0x00006000U`	Texture format read in RGBA order, and A is forced to 1.0.
`SCE_GXM_TEXTURE_SWIZZLE4_BGR1`	`0x00007000U`	Texture format read in BGRA order, and A is forced to 1.0.

Description

Defines the layout of 4-component texture formats which can be swizzled.

SceGxmTextureSwizzleYUV420Mode

Defines the memory layout of YUV420 texture formats.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureSwizzleYUV420Mode {
    SCE_GXM_TEXTURE_SWIZZLE_YUV_CSC0 = 0x00000000U,
    SCE_GXM_TEXTURE_SWIZZLE_YVU_CSC0 = 0x00001000U,
    SCE_GXM_TEXTURE_SWIZZLE_YUV_CSC1 = 0x00002000U,
    SCE_GXM_TEXTURE_SWIZZLE_YVU_CSC1 = 0x00003000U
} SceGxmTextureSwizzleYUV420Mode;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_TEXTURE_SWIZZLE_YUV_CSC0	0x00000000U	Texture planes read in YUV order, using CSC matrix 0.
SCE_GXM_TEXTURE_SWIZZLE_YVU_CSC0	0x00001000U	Texture planes read in YVU order, using CSC matrix 0.
SCE_GXM_TEXTURE_SWIZZLE_YUV_CSC1	0x00002000U	Texture planes read in YUV order, using CSC matrix 1.
SCE_GXM_TEXTURE_SWIZZLE_YVU_CSC1	0x00003000U	Texture planes read in YVU order, using CSC matrix 1.

Description

Defines the memory layout of YUV420 texture formats.

SceGxmTextureSwizzleYUV422Mode

Defines the memory layout of YUV422 texture formats.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureSwizzleYUV422Mode {
    SCE_GXM_TEXTURE_SWIZZLE_YUYV_CSC0 = 0x00000000U,
    SCE_GXM_TEXTURE_SWIZZLE_YVYU_CSC0 = 0x00001000U,
    SCE_GXM_TEXTURE_SWIZZLE_UYVY_CSC0 = 0x00002000U,
    SCE_GXM_TEXTURE_SWIZZLE_VYUY_CSC0 = 0x00003000U,
    SCE_GXM_TEXTURE_SWIZZLE_YUYV_CSC1 = 0x00004000U,
    SCE_GXM_TEXTURE_SWIZZLE_YVYU_CSC1 = 0x00005000U,
    SCE_GXM_TEXTURE_SWIZZLE_UYVY_CSC1 = 0x00006000U,
    SCE_GXM_TEXTURE_SWIZZLE_VYUY_CSC1 = 0x00007000U
} SceGxmTextureSwizzleYUV422Mode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TEXTURE_SWIZZLE_YUYV_CSC0`	`0x00000000U`	Texture format read in Y1UY0V order, using CSC matrix 0.
`SCE_GXM_TEXTURE_SWIZZLE_YVYU_CSC0`	`0x00001000U`	Texture format read in Y1VY0U order, using CSC matrix 0.
`SCE_GXM_TEXTURE_SWIZZLE_UYVY_CSC0`	`0x00002000U`	Texture format read in UY1VY0 order, using CSC matrix 0.
`SCE_GXM_TEXTURE_SWIZZLE_VYUY_CSC0`	`0x00003000U`	Texture format read in VY1UY0 order, using CSC matrix 0.
`SCE_GXM_TEXTURE_SWIZZLE_YUYV_CSC1`	`0x00004000U`	Texture format read in Y1UY0V order, using CSC matrix 1.
`SCE_GXM_TEXTURE_SWIZZLE_YVYU_CSC1`	`0x00005000U`	Texture format read in Y1VY0U order, using CSC matrix 1.
`SCE_GXM_TEXTURE_SWIZZLE_UYVY_CSC1`	`0x00006000U`	Texture format read in UY1VY0 order, using CSC matrix 1.
`SCE_GXM_TEXTURE_SWIZZLE_VYUY_CSC1`	`0x00007000U`	Texture format read in VY1UY0 order, using CSC matrix 1.

Description

Defines the memory layout of YUV422 texture formats.

SceGxmTextureType

The texture type.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTextureType {
    SCE_GXM_TEXTURE_SWIZZLED = 0x00000000U,
    SCE_GXM_TEXTURE_CUBE = 0x40000000U,
    SCE_GXM_TEXTURE_LINEAR = 0x60000000U,
    SCE_GXM_TEXTURE_TILED = 0x80000000U,
    SCE_GXM_TEXTURE_SWIZZLED_ARBITRARY = 0xa0000000U,
    SCE_GXM_TEXTURE_LINEAR_STRIDED = 0xc0000000U,
    SCE_GXM_TEXTURE_CUBE_ARBITRARY = 0xe0000000U
} SceGxmTextureType;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TEXTURE_SWIZZLED`	`0x00000000U`	The texture uses a swizzled memory layout.
`SCE_GXM_TEXTURE_CUBE`	`0x40000000U`	The texture uses a (implicitly swizzled) cube memory layout.
`SCE_GXM_TEXTURE_LINEAR`	`0x60000000U`	The texture uses a linear memory layout with implicit stride.
`SCE_GXM_TEXTURE_TILED`	`0x80000000U`	The texture uses a tiled memory layout.
`SCE_GXM_TEXTURE_SWIZZLED_ARBITRARY`	`0xa0000000U`	The texture uses a swizzled memory layout with arbitrary width and height.
`SCE_GXM_TEXTURE_LINEAR_STRIDED`	`0xc0000000U`	The texture uses a linear memory layout with an explicit stride value.
`SCE_GXM_TEXTURE_CUBE_ARBITRARY`	`0xe0000000U`	The texture uses a cube memory layout (implicitly swizzled) with arbitrary width and height.

Description

The texture type.

SceGxmTransferColorKeyMode

The color key modes.

Definition

```
#include <gxm/transfer.h>
typedef enum SceGxmTransferColorKeyMode {
    SCE_GXM_TRANSFER_COLORKEY_NONE,
    SCE_GXM_TRANSFER_COLORKEY_PASS,
    SCE_GXM_TRANSFER_COLORKEY_REJECT
} SceGxmTransferColorKeyMode;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_TRANSFER_COLORKEY_NONE	N/A	No color keying on source read.
SCE_GXM_TRANSFER_COLORKEY_PASS	N/A	Matching colors are passed.
SCE_GXM_TRANSFER_COLORKEY_REJECT	N/A	Matching colors are rejected.

Description

The color key modes.

SceGxmTransferFlags

Vertex and fragment synchronization flags for transfers.

Definition

```
#include <gxm/transfer.h>
typedef enum SceGxmTransferFlags {
    SCE_GXM_TRANSFER_FRAGMENT_SYNC = 0x00000001U,
    SCE_GXM_TRANSFER_VERTEX_SYNC = 0x00000002U
} SceGxmTransferFlags;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TRANSFER_FRAGMENT_SYNC`	`0x00000001U`	The transfer should wait for fragment processing to complete.
`SCE_GXM_TRANSFER_VERTEX_SYNC`	`0x00000002U`	The transfer should wait for vertex processing to complete.

Description

Vertex and fragment synchronization flags for transfers.

SceGxmTransferFormat

The formats available for transfers.

Definition

```
#include <gxm/transfer.h>
typedef enum SceGxmTransferFormat {
    SCE_GXM_TRANSFER_FORMAT_U8_R = 0x00000000U,
    SCE_GXM_TRANSFER_FORMAT_U4U4U4U4_ABGR = 0x00010000U,
    SCE_GXM_TRANSFER_FORMAT_U1U5U5U5_ABGR = 0x00020000U,
    SCE_GXM_TRANSFER_FORMAT_U5U6U5_BGR = 0x00030000U,
    SCE_GXM_TRANSFER_FORMAT_U8U8_GR = 0x00040000U,
    SCE_GXM_TRANSFER_FORMAT_U8U8U8_BGR = 0x00050000U,
    SCE_GXM_TRANSFER_FORMAT_U8U8U8U8_ABGR = 0x00060000U,
    SCE_GXM_TRANSFER_FORMAT_VYUY422 = 0x00070000U,
    SCE_GXM_TRANSFER_FORMAT_YVYU422 = 0x00080000U,
    SCE_GXM_TRANSFER_FORMAT_UYVY422 = 0x00090000U,
    SCE_GXM_TRANSFER_FORMAT_YUYV422 = 0x000a0000U,
    SCE_GXM_TRANSFER_FORMAT_U2U10U10U10_ABGR = 0x000d0000U,
    SCE_GXM_TRANSFER_FORMAT_RAW16 = 0x000f0000U,
    SCE_GXM_TRANSFER_FORMAT_RAW32 = 0x00110000U,
    SCE_GXM_TRANSFER_FORMAT_RAW64 = 0x00120000U,
    SCE_GXM_TRANSFER_FORMAT_RAW128 = 0x00130000U
} SceGxmTransferFormat;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TRANSFER_FORMAT_U8_R`	`0x00000000U`	8-bit format, 8-bit unsigned integer.
`SCE_GXM_TRANSFER_FORMAT_U4U4U4U4_ABGR`	`0x00010000U`	16-bit format, 4x 4-bit unsigned integer, read in ABGR order from memory.
`SCE_GXM_TRANSFER_FORMAT_U1U5U5U5_ABGR`	`0x00020000U`	16-bit format, 1-bit unsigned and 3x 5-bit unsigned integer, read in ABGR order from memory.
`SCE_GXM_TRANSFER_FORMAT_U5U6U5_BGR`	`0x00030000U`	16-bit format, 5-bit unsigned, 6-bit unsigned and 5-bit unsigned integer, read in BGR order from memory.
`SCE_GXM_TRANSFER_FORMAT_U8U8_GR`	`0x00040000U`	16-bit format, 2x 8-bit unsigned integer, read in GR order from memory.
`SCE_GXM_TRANSFER_FORMAT_U8U8U8_BGR`	`0x00050000U`	24-bit packed format, 3x 8-bit unsigned integer, read in BGR order from memory.
`SCE_GXM_TRANSFER_FORMAT_U8U8U8U8_ABGR`	`0x00060000U`	32-bit format, 4x 8-bit unsigned integer, read in ABGR order from memory.
`SCE_GXM_TRANSFER_FORMAT_VYUY422`	`0x00070000U`	Interleaved YUV, VYUY 2-pixel blocks.
`SCE_GXM_TRANSFER_FORMAT_YVYU422`	`0x00080000U`	Interleaved YUV, YVYU 2-pixel blocks.

Macro	Value	Description
`SCE_GXM_TRANSFER_FORMAT_UYVY422`	`0x00090000U`	Interleaved YUV, UYVY 2-pixel blocks.
`SCE_GXM_TRANSFER_FORMAT_YUYV422`	`0x000a0000U`	Interleaved YUV, YUYV 2-pixel blocks.
`SCE_GXM_TRANSFER_FORMAT_U2U10U10U10_ABGR`	`0x000d0000U`	32-bit format, 2-bit unsigned and 3x 10-bit unsigned integer, read in ABGR order from memory.
`SCE_GXM_TRANSFER_FORMAT_RAW16`	`0x000f0000U`	16-bit format, raw data.
`SCE_GXM_TRANSFER_FORMAT_RAW32`	`0x00110000U`	32-bit format, raw data.
`SCE_GXM_TRANSFER_FORMAT_RAW64`	`0x00120000U`	62-bit format, raw data.
`SCE_GXM_TRANSFER_FORMAT_RAW128`	`0x00130000U`	128-bit format, raw data.

Description

The formats available for transfers.

SceGxmTransferType

The transfer memory layout types.

Definition

```
#include <gxm/transfer.h>
typedef enum SceGxmTransferType {
    SCE_GXM_TRANSFER_LINEAR = 0x00000000U,
    SCE_GXM_TRANSFER_TILED = 0x00400000U,
    SCE_GXM_TRANSFER_SWIZZLED = 0x00800000U
} SceGxmTransferType;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TRANSFER_LINEAR`	`0x00000000U`	The transfer area uses a linear memory layout.
`SCE_GXM_TRANSFER_TILED`	`0x00400000U`	The transfer area uses a tiled memory layout.
`SCE_GXM_TRANSFER_SWIZZLED`	`0x00800000U`	The transfer area uses a swizzled memory layout.

Description

The transfer memory layout types.

SceGxmTwoSidedMode

Two-sided mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmTwoSidedMode {
    SCE_GXM_TWO_SIDED_DISABLED = 0x00000000U,
    SCE_GXM_TWO_SIDED_ENABLED = 0x00000800U
} SceGxmTwoSidedMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_TWO_SIDED_DISABLED`	`0x00000000U`	Front setting used for both front and back.
`SCE_GXM_TWO_SIDED_ENABLED`	`0x00000800U`	Independent front and back setting.

Description

Two-sided mode. When two sided mode is enabled, most state can be set independently for front and back polygons. When two sided mode is disabled, the front setting is used for both front and back, and setting the back state is unnecessary (and should be avoided).

SceGxmValidRegion

Represents a rectangular region from `0,0` to *`xMax`*, *`yMax`* (inclusive).

Definition

```
#include <gxm/structs.h>
typedef struct SceGxmValidRegion {
    uint32_t xMax;
    uint32_t yMax;
} SceGxmValidRegion;
```

Members

xMax	The maximum x value in pixels.
yMax	The maximum y value in pixels.

Description

Represents a rectangular region from `0,0` to *`xMax`*, *`yMax`* (inclusive). This structure is for use as the optional *`validRegion`* parameter to `sceGxmBeginScene()`.

Although the region is provided in pixels, this will be internally aligned to tile granularity for the hardware.

SceGxmVertexAttribute

A vertex attribute descriptor.

Definition

```
#include <gxm/structs.h>
typedef struct SceGxmVertexAttribute {
    uint16_t streamIndex;
    uint16_t offset;
    uint8_t format;
    uint8_t componentCount;
    uint16_t regIndex;
} SceGxmVertexAttribute;
```

Members

streamIndex	The index within the stream array.
offset	The byte offset from the start of each vertex.
format	The data type for each scalar from `SceGxmAttributeFormat`.
componentCount	The number of components of typed data or the number of 32-bit words of untyped data.
regIndex	The PA register start index.

Description

A vertex attribute descriptor. A vertex attribute is either typed or untyped. Typed attributes are unpacked to float4 format for the shader code, and must specify between 1 and 4 scalar components in the *componentCount* field. Untyped attributes may only be used with attributes that are unpacked in shader code explicitly (using the `__regformat` keyword), and may specify between 1 and 255 32-bit words of untyped data in the *componentCount* field.

Vertex attributes are fetched in units of 32-bit words even if the address of the vertex attribute data is not aligned to 4 bytes. Because of this, additional data may be fetched after the attribute data in order to align the fetched size up to a whole number of 32-bit words. To ensure that this overfetch does not result in unexpected GPU page faults, vertex data should not be placed in the last `SCE_GXM_MAX_ATTRIBUTE_OVERFETCH` bytes of GPU mapped memory.

SceGxmVertexProgram

The data structure for vertex programs.

Definition

```
#include <gxm/vertex_program.h>
typedef struct SceGxmVertexProgram;
```

Description

The data structure for vertex programs. This structure is currently opaque, filled out internally by the shader patcher.

SceGxmVertexStream

A vertex stream descriptor.

Definition

```
#include <gxm/structs.h>
typedef struct SceGxmVertexStream {
    uint16_t stride;
    uint16_t indexSource;
} SceGxmVertexStream;
```

Members

stride	The byte stride between each vertex. A zero stride can be used to specify constant streams.
indexSource	Defines how the stream is indexed from `SceGxmIndexSource`.

Description

A vertex stream descriptor.

SceGxmViewportMode

Viewport transform mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmViewportMode {
    SCE_GXM_VIEWPORT_DISABLED = 0x00010000U,
    SCE_GXM_VIEWPORT_ENABLED = 0x00000000U
} SceGxmViewportMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_VIEWPORT_DISABLED`	`0x00010000U`	No viewport transform.
`SCE_GXM_VIEWPORT_ENABLED`	`0x00000000U`	Use the viewport transform.

Description

Viewport transform mode. When the viewport transform is disabled, vertices are assumed to be output directly in screen space coordinates within a range of -1024 to 7167 for each axis. Use of coordinates outside of this range may result in incorrect rasterization. For a description of the usual transform see `sceGxmSetViewport()`.

SceGxmVisibilityTestMode

Visibility test mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmVisibilityTestMode {
    SCE_GXM_VISIBILITY_TEST_DISABLED = 0x00000000U,
    SCE_GXM_VISIBILITY_TEST_ENABLED = 0x00004000U
} SceGxmVisibilityTestMode;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_VISIBILITY_TEST_DISABLED	0x00000000U	Visibility test disabled.
SCE_GXM_VISIBILITY_TEST_ENABLED	0x00004000U	Visibility test enabled.

Description

Visibility test mode.

SceGxmVisibilityTestOp

Operation to perform during visibility test.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmVisibilityTestOp {
    SCE_GXM_VISIBILITY_TEST_OP_INCREMENT = 0x00000000U,
    SCE_GXM_VISIBILITY_TEST_OP_SET = 0x00040000U
} SceGxmVisibilityTestOp;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_VISIBILITY_TEST_OP_INCREMENT`	`0x00000000U`	Increment the count for each visible pixel.
`SCE_GXM_VISIBILITY_TEST_OP_SET`	`0x00040000U`	Set the count to a non-zero value if any pixel is visible.

Description

Operation to perform during visibility test.

SceGxmWarning

Warnings that can be selectively configured using sceGxmSetWarningEnabled().

Definition

```
#include <gxm/init.h>
typedef enum SceGxmWarning {
    SCE_GXM_WARNING_SCENE_SPLIT,
    SCE_GXM_WARNING_VERTEX_DEFAULT_UNIFORM_BUFFER_RECYCLED,
    SCE_GXM_WARNING_FRAGMENT_DEFAULT_UNIFORM_BUFFER_RECYCLED,
    SCE_GXM_WARNING_STREAMS_PROVIDED_WITH_ZERO_COUNT,
    SCE_GXM_WARNING_ATTRIBUTES_PROVIDED_WITH_ZERO_COUNT,
    SCE_GXM_WARNING_PROGRAM_REGISTERED_WITH_SHADER_PATCHER,
    SCE_GXM_WARNING_BLEND_INFO_IGNORED_FOR_NATIVECOLOR,
    SCE_GXM_WARNING_USING_INTERPOLANT_NOT_WRITTEN_BY_VERTEX_PROGRAM,
    SCE_GXM_WARNING_DEPTH_STENCIL_SURFACE_SETTING_IGNORED,
    SCE_GXM_WARNING_DEFERRED_CONTEXT_MISSING_VIEWPORT,
    SCE_GXM_WARNING_DEFERRED_CONTEXT_MISSING_REGION_CLIP,
    SCE_GXM_WARNING_USING_INAPPROPRIATE_MEMORY_CACHE_CONFIGURATION,
    SCE_GXM_WARNING_PRECOMPUTING_DISABLED_FRAGMENT_STATE
} SceGxmWarning;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_WARNING_SCENE_SPLIT	N/A	Ring buffer high-water marks have been passed, resulting in the scene being split into multiple jobs.
SCE_GXM_WARNING_VERTEX_DEFAULT_UNIFORM_BUFFER_RECYCLED	N/A	The previous vertex default uniform buffer has not been used, resulting in it being recycled.
SCE_GXM_WARNING_FRAGMENT_DEFAULT_UNIFORM_BUFFER_RECYCLED	N/A	The previous fragment default uniform buffer has not been used, resulting in it being recycled.
SCE_GXM_WARNING_STREAMS_PROVIDED_WITH_ZERO_COUNT	N/A	Streams provided to sceGxmShaderPatcherCreateVertexProgram(), but the provided stream count is 0.
SCE_GXM_WARNING_ATTRIBUTES_PROVIDED_WITH_ZERO_COUNT	N/A	Attributes provided to sceGxmShaderPatcherCreateVertexProgram(), but the provided attribute count is 0.
SCE_GXM_WARNING_PROGRAM_REGISTERED_WITH_SHADER_PATCHER	N/A	Program is being registered multiple times with the shader patcher.
SCE_GXM_WARNING_BLEND_INFO_IGNORED_FOR_NATIVECOLOR	N/A	Ignoring SceGxmBlendInfo structure provided to sceGxmShaderPatcherCreateFragmentProgram() as program uses __nativecolor.
SCE_GXM_WARNING_USING_INTERPOLANT_NOT_WRITTEN_BY_VERTEX_PROGRAM	N/A	An interpolant is being used by the fragment program, without being written by the associated vertex program.
SCE_GXM_WARNING_DEPTH_STENCIL_SURFACE_SETTING_IGNORED	N/A	The depth stencil surface has a setting which is ignored.

Macro	Value	Description
SCE_GXM_WARNING_DEFERRED_CONTEXT_MISSING_VIEWPORT	N/A	A deferred context has encountered a draw call, but it has not had a valid viewport set.
SCE_GXM_WARNING_DEFERRED_CONTEXT_MISSING_REGION_CLIP	N/A	A deferred context has encountered a draw call, but it has not had a valid region clip set.
SCE_GXM_WARNING_USING_INAPPROPRIATE_MEMORY_CACHE_CONFIGURATION	N/A	Supplied memory is using an inappropriate cache configuration.
SCE_GXM_WARNING_PRECOMPUTING_DISABLED_FRAGMENT_STATE	N/A	Precomputed fragment state is being created for a fragment program with shading disabled.

Description

Warnings that can be selectively configured using sceGxmSetWarningEnabled().

SceGxmWBufferMode

W buffering mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmWBufferMode {
    SCE_GXM_WBUFFER_DISABLED = 0x00000000U,
    SCE_GXM_WBUFFER_ENABLED = 0x00004000U
} SceGxmWBufferMode;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_WBUFFER_DISABLED	0x00000000U	Do not use W buffer, use Z buffering.
SCE_GXM_WBUFFER_ENABLED	0x00004000U	Use W buffering.

Description

W buffering mode. Enabled or disable using the W value when building the screen space coordinates during the viewport transform. See `sceGxmSetViewport()` for more details.

SceGxmWClampMode

W clamp mode.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmWClampMode {
    SCE_GXM_WCLAMP_MODE_DISABLED = 0x00000000U,
    SCE_GXM_WCLAMP_MODE_ENABLED = 0x00008000U
} SceGxmWClampMode;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_WCLAMP_MODE_DISABLED`	`0x00000000U`	No W clamping in the viewport transform.
`SCE_GXM_WCLAMP_MODE_ENABLED`	`0x00008000U`	W clamping in the viewport transform.

Description

W clamp mode. See `sceGxmSetViewport()` for a description of how this interacts with the viewport transform.

SceGxmYuvProfile

YUV profile.

Definition

```
#include <gxm/constants.h>
typedef enum SceGxmYuvProfile {
    SCE_GXM_YUV_PROFILE_BT601_STANDARD,
    SCE_GXM_YUV_PROFILE_BT709_STANDARD,
    SCE_GXM_YUV_PROFILE_BT601_FULL_RANGE,
    SCE_GXM_YUV_PROFILE_BT709_FULL_RANGE
} SceGxmYuvProfile;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_YUV_PROFILE_BT601_STANDARD`	N/A	BT.601 with brightness range from 16-235 and color difference values from 16-240.
`SCE_GXM_YUV_PROFILE_BT709_STANDARD`	N/A	BT.709 with brightness range from 16-235 and color difference values from 16-240.
`SCE_GXM_YUV_PROFILE_BT601_FULL_RANGE`	N/A	BT.601 with brightness range from 0-255 and color difference values from 0-255.
`SCE_GXM_YUV_PROFILE_BT709_FULL_RANGE`	N/A	BT.709 with brightness range from 0-255 and color difference values from 0-255.

Description

YUV profile.

Functions

sceGxmAddRazorGpuCaptureBuffer

Register a capture buffer with Razor.

Definition

```
#include <gxm/memory.h>
SceGxmErrorCode sceGxmAddRazorGpuCaptureBuffer(
    void *base,
    uint32_t size
);
```

Arguments

[in] *base*	The base address of the GPU capture buffer.
[in] *size*	The size of the capture buffer in bytes.

Return Values

Value	Description
SCE_OK	The operation was completed successfully.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a pointer was invalid.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because the *size* parameter was 0.
SCE_GXM_ERROR_RAZOR	The operation failed because of an error in the Razor layer.

Description

Register a capture buffer with Razor. This API is required only when performing a Razor GPU Capture.

Registering a buffer with Razor is only necessary if the size of a user declared uniform buffer declared in a vertex or fragment program can be exceeded through dynamic indexing at runtime. One example is an array of matrices dynamically indexed by a vertex program performing skinning.

When a uniform buffer start address is set, by calling `sceGxmSetVertexUniformBuffer()` or `sceGxmSetFragmentUniformBuffer()`, for a vertex or fragment program which dynamically indexes the buffer, Razor will try to capture the whole registered capture buffer whose range includes this start address. Therefore, one call to this function can register one single capture buffer, sub-ranges of which can be used as user-declared uniform buffers for individual draw calls.

Currently up to 1024 buffers can be registered through this API at any one time.

sceGxmBeginCommandList

Begins rendering to a command list.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmBeginCommandList(
    SceGxmContext *deferredContext
);
```

Arguments

[in,out] *deferredContext*	A pointer to a deferred context.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because the pointer to the deferred context was NULL.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because the context is not a deferred context.
SCE_GXM_ERROR_WITHIN_COMMAND_LIST	The operation failed because the deferred context is already within a command list.
SCE_GXM_ERROR_RESERVE_FAILED	The operation failed because the VDM buffer callback function failed to provide the minimum amount of memory required for a command list.

Description

Begins rendering to a command list. This function is only supported on a deferred context, and it will return the SCE_GXM_ERROR_INVALID_VALUE error code if called using the immediate context.

A command list represents a group of draw calls with a full GPU state reset before and after the group. Draw calls within the command list use the current state set on the deferred context. This is independent from the state of any other deferred context and also the immediate context, which will ultimately execute the command list.

Region clip and viewport are not set automatically on the deferred context when starting a command list, so this state, if necessary, should be set before the first draw call. At a minimum, it should be set once after the deferred context has been created.

Once all draw calls have been submitted, the command list can be ended using sceGxmEndCommandList(), and the resulting SceGxmCommandList may be executed on the immediate context.

sceGxmBeginScene

Starts a scene with the given render target and optional color and depth/stencil surfaces.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmBeginScene(
    SceGxmContext *immediateContext,
    uint32_t flags,
    const SceGxmRenderTarget *renderTarget,
    const SceGxmValidRegion *validRegion,
    SceGxmSyncObject *vertexSyncObject,
    SceGxmSyncObject *fragmentSyncObject,
    const SceGxmColorSurface *colorSurface,
    const SceGxmDepthStencilSurface *depthStencilSurface
);
```

Arguments

[in,out] *immediateContext*	A pointer to the immediate context.
[in] *flags*	Bitwise combined flags from `SceGxmSceneFlags`.
[in] *renderTarget*	The render target dimensions and firmware job info. The render target must not be destroyed until the scene has been ended using `sceGxmEndScene()`.
[in] *validRegion*	An optional valid region of the render target. This structure does not need to persist after the call.
[in] *vertexSyncObject*	Reserved for future use. This must be `NULL`.
[in] *fragmentSyncObject*	An optional sync object to synchronize against fragment processing.
[in] *colorSurface*	An optional pointer to the color surface. The structure is copied during this function and therefore does not need to persist after the call. The GPU data pointed to by the structure must persist until fragment processing for the current scene has been completed.
[in] *depthStencilSurface*	An optional pointer to the depth/stencil surface. Passing `NULL` results in behaviour that is the same as providing a depth/stencil surface that has been initialized with `SceGxmDepthStencilSurfaceInitDisabled()`. The structure is copied during this function and therefore does not need to persist after the call. The GPU data pointed to by the structure must persist until fragment processing for the current scene has been completed.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because both color and depth surfaces were disabled or `NULL`, or because the context is not an immediate context.
`SCE_GXM_ERROR_WITHIN_SCENE`	The operation failed because the function was called within a scene.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation failed due to an unsupported combination of MSAA mode and color surface downscale.
`SCE_GXM_ERROR_DRIVER`	The operation failed due to a driver error.

Description

Starts a scene with the given render target and optional color and depth/stencil surfaces. For a complete description of this function, please refer to the documentation for `sceGxmBeginSceneEx()`. `sceGxmBeginScene()` behaves identically to `sceGxmBeginSceneEx()` when using the same depth stencil surface for both load and store.

sceGxmBeginSceneEx

Starts a scene with the given render target and optional color and load/store depth/stencil surfaces.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmBeginSceneEx(
    SceGxmContext *immediateContext,
    uint32_t flags,
    const SceGxmRenderTarget *renderTarget,
    const SceGxmValidRegion *validRegion,
    SceGxmSyncObject *vertexSyncObject,
    SceGxmSyncObject *fragmentSyncObject,
    const SceGxmColorSurface *colorSurface,
    const SceGxmDepthStencilSurface *loadDepthStencilSurface,
    const SceGxmDepthStencilSurface *storeDepthStencilSurface
);
```

Arguments

[in,out] `immediateContext`	A pointer to the immediate context.
[in] `flags`	Bitwise combined flags from `SceGxmSceneFlags`.
[in] `renderTarget`	The render target dimensions and firmware job info. The render target must not be destroyed until the scene has been ended using `sceGxmEndScene()`.
[in] `validRegion`	An optional valid region of the render target. This structure does not need to persist after the call.
[in] `vertexSyncObject`	Reserved for future use. Must be `NULL`.
[in] `fragmentSyncObject`	An optional sync object to synchronize against fragment processing.
[in] `colorSurface`	An optional pointer to the color surface. The structure is copied during this function and therefore does not need to persist after the call. The GPU data pointed to by the structure must persist until fragment processing for the current scene has been completed.
[in] `loadDepthStencilSurface`	An optional pointer to the depth/stencil surface used for load. The structure is copied during this function and therefore does not need to persist after the call. The GPU data pointed to by the structure must persist until fragment processing for the current scene has been completed.
[in] `storeDepthStencilSurface`	An optional pointer to the depth/stencil surface used for load. The structure is copied during this function and therefore does not need to persist after the call. The GPU data pointed to by the structure must persist until fragment processing for the current scene has been completed.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.

Value	Description
`SCE_GXM_ERROR_WITHIN_SCENE`	The operation failed because the function was called within a scene.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation failed due to an unsupported combination of MSAA mode and color surface downscale.
`SCE_GXM_ERROR_DRIVER`	The operation failed due to a driver error.

Description

Starts a scene with the given render target and optional color and load/store depth/stencil surfaces. This function is only supported on the immediate context, and it will return the `SCE_GXM_ERROR_INVALID_VALUE` error code if called using a deferred context.

The *`flags`* argument contains information about scene dependencies and synchronization. See the documentation for `SceGxmSceneFlags` for more details.

The render target defines the how the surfaces are tiled for fragment processing, and provides resources for the firmware layer to schedule the job(s) for this scene. Within the render target, the set of valid tiles can be restricted to a subset of the render target using the *`validRegion`* parameter. If this parameter is not supplied, then the whole render target is considered to be valid. Only valid tiles are executed by the fragment pipeline. Geometry can be discarded at tile granularity by calling `sceGxmSetRegionClip()` before drawing. Although the valid region must be static for the whole scene, region clip can change between draw calls. Geometry, in tiles that pass region clip, is serialized into the parameter buffer for fragment processing.

If the *`colorSurface`* is not `NULL` and is not disabled, the region in which color values are written to memory can be clipped at pixel granularity by calling `sceGxmColorSurfaceSetClip()` on the surface before `sceGxmBeginSceneEx()`. The *`colorSurface`* parameter does not need to persist in memory after the call to `sceGxmBeginSceneEx()` returns.

If depth tests other than NEVER or ALWAYS are required within the scene, a store depth/stencil surface must be provided that contains depth data as part of its format; this ensures correct behavior during a partial render. Similarly, if stencil tests other than NEVER or ALWAYS are required within the scene, a store depth/stencil buffer must be provided that contains stencil data as part of its format. Attempting to use a depth test other than NEVER or ALWAYS with either a disabled store depth/stencil surface or a store depth/stencil surface with no depth data in its format will cause the draw call to return `SCE_GXM_ERROR_INVALID_DEPTH_STENCIL_CONFIGURATION`. Attempting to use a stencil test other than NEVER or ALWAYS with either a disabled store depth/stencil surface or a store depth/stencil surface with no stencil data in its format will also cause the draw call to return `SCE_GXM_ERROR_INVALID_DEPTH_STENCIL_CONFIGURATION`. If the depth and stencil tests will be NEVER or ALWAYS for all draw calls in a scene, the store depth/stencil surface may be `NULL` or disabled. If a mask update fragment program is required within the scene, a store depth/stencil surface must be provided that contains mask data as part of its format; this ensures correct behavior during a partial render. Attempting to issue a draw call that uses the mask update fragment program when the store depth/stencil surface does not contain mask data as part of its format will cause the draw call to return `SCE_GXM_ERROR_INVALID_DEPTH_STENCIL_CONFIGURATION`.

In order to have correct behavior during a partial render, the depth/stencil/mask components supported by the load surface must be a subset of the depth/stencil/mask components supported by the store surface. After a partial render, all depth/stencil/mask data is loaded and stored through the store surface only.

The implementation of `sceGxmBeginSceneEx()` also sets the region clip to clip all geometry outside of the render target, and sets the viewport to cover the valid region. This is equivalent to calling:

```
uint32_t xMax = validRegion ? validRegion->xMax : (
renderTargetWidthInPixels - 1);
uint32_t yMax = validRegion ? validRegion->yMax : (
renderTargetHeightInPixels - 1);
sceGxmSetDefaultRegionClipAndViewport(context, xMax, yMax);
```

sceGxmColorSurfaceGetClip

Gets the color surface clip parameters.

Definition

```
#include <gxm/surface.h>
void sceGxmColorSurfaceGetClip(
    const SceGxmColorSurface *surface,
    uint32_t *xMin,
    uint32_t *yMin,
    uint32_t *xMax,
    uint32_t *yMax
);
```

Arguments

[in] *surface*	A pointer to the surface.
[out] *xMin*	A pointer to storage for the minimum x value in pixels.
[out] *yMin*	A pointer to storage for the minimum y value in pixels.
[out] *xMax*	A pointer to storage for the maximum x value in pixels.
[out] *yMax*	A pointer to storage for the maximum y value in pixels.

Return Values

None

Description

Gets the color surface clip parameters. The min and max values are inclusive.

sceGxmColorSurfaceGetData

Gets a pointer to the surface data for the given color surface.

Definition

```
#include <gxm/surface.h>
void *sceGxmColorSurfaceGetData(
    const SceGxmColorSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the surface.

Return Values

A pointer to the surface data.

Description

Gets a pointer to the surface data for the given color surface.

sceGxmColorSurfaceGetDitherMode

Gets the color surface dither mode.

Definition

```
#include <gxm/surface.h>
SceGxmColorSurfaceDitherMode sceGxmColorSurfaceGetDitherMode(
    const SceGxmColorSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the surface.

Return Values

The dither mode currently used for the surface.

Description

Gets the color surface dither mode.

sceGxmColorSurfaceGetFormat

Gets the color format for the given color surface.

Definition

```
#include <gxm/surface.h>
SceGxmColorFormat sceGxmColorSurfaceGetFormat(
    const SceGxmColorSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the surface.

Return Values

The color format of the surface.

Description

Gets the color format for the given color surface.

sceGxmColorSurfaceGetGammaMode

Gets the color surface gamma mode.

Definition

```
#include <gxm/surface.h>
SceGxmColorSurfaceGammaMode sceGxmColorSurfaceGetGammaMode(
    const SceGxmColorSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the surface.

Return Values

The gamma mode currently used for the surface.

Description

Gets the color surface gamma mode.

sceGxmColorSurfaceGetScaleMode

Gets the color surface scale mode.

Definition

```
#include <gxm/surface.h>
SceGxmColorSurfaceScaleMode sceGxmColorSurfaceGetScaleMode(
    const SceGxmColorSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the surface.

Return Values

The scale mode currently used for the surface.

Description

Gets the color surface scale mode.

sceGxmColorSurfaceGetStrideInPixels

Gets the stride in pixels for the given color surface.

Definition

```
#include <gxm/surface.h>
uint32_t sceGxmColorSurfaceGetStrideInPixels(
    const SceGxmColorSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the surface.

Return Values

The stride in pixels of the surface data.

Description

Gets the stride in pixels for the given color surface.

sceGxmColorSurfaceGetType

Gets the memory layout type for the given color surface.

Definition

```
#include <gxm/surface.h>
SceGxmColorSurfaceType sceGxmColorSurfaceGetType(
    const SceGxmColorSurface *surface
);
```

Arguments

[in] `surface`	A pointer to the surface.

Return Values

The memory layout of the surface.

Description

Gets the memory layout type for the given color surface.

sceGxmColorSurfaceInit

Creates a surface with the given buffer parameters.

Definition

```
#include <gxm/surface.h>
SceGxmErrorCode sceGxmColorSurfaceInit(
    SceGxmColorSurface *surface,
    SceGxmColorFormat colorFormat,
    SceGxmColorSurfaceType surfaceType,
    SceGxmColorSurfaceScaleMode scaleMode,
    SceGxmOutputRegisterSize outputRegisterSize,
    uint32_t width,
    uint32_t height,
    uint32_t strideInPixels,
    void *data
);
```

Arguments

[out] *surface*	A pointer to surface to be initialized.
[in] *colorFormat*	The color format for the surface.
[in] *surfaceType*	The memory layout type.
[in] *scaleMode*	The scaling mode to use before storing pixels.
[in] *outputRegisterSize*	The output register size.
[in] *width*	The width of the surface.
[in] *height*	The height of the surface.
[in] *strideInPixels*	The stride of the surface in pixels.
[in] *data*	A pointer to the surface data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a pointer was unexpectedly NULL.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because a parameter was invalid.
SCE_GXM_ERROR_UNSUPPORTED	The operation failed because an unsupported output register size was specified.

Description

Creates a surface with the given buffer parameters. The surface is set up by default to emit to all pixels in the buffer. To clip the output to a subset of the buffer, call sceGxmColorSurfaceSetClip().

The *data* parameter must be aligned to SCE_GXM_COLOR_SURFACE_ALIGNMENT bytes. The *strideInPixels* parameter must be aligned to 2 pixels.

sceGxmColorSurfaceInitDisabled

Creates a color surface for depth-only rendering.

Definition

```
#include <gxm/surface.h>
SceGxmErrorCode sceGxmColorSurfaceInitDisabled(
    SceGxmColorSurface *surface
);
```

Arguments

[out] *surface*	A pointer to the surface.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because `SceGxmColorSurface` cannot be a `NULL` pointer.

Description

Creates a color surface for depth-only rendering.

sceGxmColorSurfaceIsEnabled

Returns true if the given color surface emits pixels.

Definition

```
#include <gxm/surface.h>
bool sceGxmColorSurfaceIsEnabled(
    const SceGxmColorSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the depth/stencil surface.

Return Values

Value	Description
`true`	The color surface is enabled (emits pixels).
`false`	The color surface is disabled (depth only rendering).

Description

Returns true if the given color surface emits pixels.

sceGxmColorSurfaceSetClip

Sets the surface clip parameters to only emit to a subset of the pixels in the buffer.

Definition

```
#include <gxm/surface.h>
void sceGxmColorSurfaceSetClip(
    SceGxmColorSurface *surface,
    uint32_t xMin,
    uint32_t yMin,
    uint32_t xMax,
    uint32_t yMax
);
```

Arguments

[in,out] *surface*	A pointer to the surface.
[in] *xMin*	The minimum x value in pixels.
[in] *yMin*	The minimum y value in pixels.
[in] *xMax*	The maximum x value in pixels.
[in] *yMax*	The maximum y value in pixels.

Return Values

None

Description

Sets the surface clip parameters to only emit to a subset of the pixels in the buffer. The min and max values are inclusive.

sceGxmColorSurfaceSetData

Sets the data pointer for the given color surface.

Definition

```
#include <gxm/surface.h>
SceGxmErrorCode sceGxmColorSurfaceSetData(
    SceGxmColorSurface *surface,
    void *data
);
```

Arguments

[in,out] *surface*	A pointer to the surface.
[in] *data*	A pointer to the surface data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a pointer was unexpectedly NULL.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because the surface was not aligned correctly.
SCE_GXM_ERROR_UNSUPPORTED	The operation failed because it is not supported.

Description

Sets the data pointer for the given color surface. The *data* parameter must be aligned to SCE_GXM_COLOR_SURFACE_ALIGNMENT bytes.

Notes

Not supported when the surface was initialized using sceGxmColorSurfaceInitDisabled().

sceGxmColorSurfaceSetDitherMode

Sets the color surface dither mode.

Definition

```
#include <gxm/surface.h>
SceGxmErrorCode sceGxmColorSurfaceSetDitherMode(
    SceGxmColorSurface *surface,
    SceGxmColorSurfaceDitherMode ditherMode
);
```

Arguments

[in,out] *surface*	A pointer to the surface.
[in] *ditherMode*	The dither mode to apply on pixel write.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a pointer was unexpectedly NULL.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because a parameter was invalid.

Description

Sets the color surface dither mode.

sceGxmColorSurfaceSetFormat

Sets the color format for the given color surface.

Definition

```
#include <gxm/surface.h>
SceGxmErrorCode sceGxmColorSurfaceSetFormat(
    SceGxmColorSurface *surface,
    SceGxmColorFormat format
);
```

Arguments

[in,out] *surface*	A pointer to the surface.
[in] *format*	The color format to set.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a pointer was unexpectedly NULL.
SCE_GXM_ERROR_UNSUPPORTED	The operation failed because it is not supported.

Description

Sets the color format for the given color surface.

Notes

Not supported when the surface was initialized using sceGxmColorSurfaceInitDisabled().

The number of bits per pixel for the format needs to match the number of bits per pixel for the format used during initialization.

The supplied format must be able to support the use of the scale mode currently set on the color surface.

sceGxmColorSurfaceSetGammaMode

Sets the color surface gamma mode.

Definition

```
#include <gxm/surface.h>
SceGxmErrorCode sceGxmColorSurfaceSetGammaMode(
    SceGxmColorSurface *surface,
    SceGxmColorSurfaceGammaMode gammaMode
);
```

Arguments

[in,out] *surface*	A pointer to the surface.
[in] *gammaMode*	The gamma mode to apply on pixel write.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a pointer was unexpectedly NULL.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because a parameter was invalid.

Description

Sets the color surface gamma mode.

sceGxmColorSurfaceSetScaleMode

Sets the color surface scale mode.

Definition

```
#include <gxm/surface.h>
void sceGxmColorSurfaceSetScaleMode(
    SceGxmColorSurface *surface,
    SceGxmColorSurfaceScaleMode scaleMode
);
```

Arguments

[in,out] *surface*	A pointer to the surface.
[in] *scaleMode*	The scaling mode to use before storing pixels.

Return Values

None

Description

Sets the color surface scale mode.

sceGxmCreateContext

Creates the immediate context.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmCreateContext(
    const SceGxmContextParams *params,
    SceGxmContext **immediateContext
);
```

Arguments

[in] *params*	A pointer to the initialization parameters. This structure does not need to persist after the call.
[out] *immediateContext*	A pointer to storage for an `SceGxmContext` pointer.

Return Values

Value	Description
`SCE_OK`	The operation was successful and the rendering context is now populated.
`SCE_GXM_ERROR_UNINITIALIZED`	The operation failed because the library was not initialized.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed due to invalid buffer alignment.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.

Description

Creates the immediate context. The rendering context uses the memory pointed to by the *hostMem* member of `SceGxmContextParams` for state tracking between draw calls and between GPU jobs. The remaining memory regions described by `SceGxmContextParams` are used for dynamically generated GPU data structures.

After the rendering context has been created, the following states will be set to `NULL`:

- Vertex program
- Fragment program
- Vertex streams
- Vertex and fragment uniform buffers
- Vertex and fragment textures
- Precomputed vertex and fragment state
- Visibility buffer

The following default states are set during creation:

```
sceGxmSetYuvProfile(context, 0, SCE_GXM_YUV_PROFILE_BT601_STANDARD);
sceGxmSetYuvProfile(context, 1, SCE_GXM_YUV_PROFILE_BT709_STANDARD);
sceGxmSetFrontDepthFunc(context, SCE_GXM_DEPTH_FUNC_LESS_EQUAL);
sceGxmSetBackDepthFunc(context, SCE_GXM_DEPTH_FUNC_LESS_EQUAL);
sceGxmSetFrontFragmentProgramEnable(context,
    SCE_GXM_FRAGMENT_PROGRAM_ENABLED);
sceGxmSetBackFragmentProgramEnable(context,
    SCE_GXM_FRAGMENT_PROGRAM_ENABLED);
sceGxmSetFrontDepthWriteEnable(context, SCE_GXM_DEPTH_WRITE_ENABLED);
sceGxmSetBackDepthWriteEnable(context, SCE_GXM_DEPTH_WRITE_ENABLED);
```

```
sceGxmSetFrontLineFillLastPixelEnable(context,
SCE_GXM_LINE_FILL_LAST_PIXEL_DISABLED);
sceGxmSetBackLineFillLastPixelEnable(context,
    SCE_GXM_LINE_FILL_LAST_PIXEL_DISABLED);
sceGxmSetFrontStencilRef(context, 0);
sceGxmSetBackStencilRef(context, 0);
sceGxmSetFrontPointLineWidth(context, 1);
sceGxmSetBackPointLineWidth(context, 1);
sceGxmSetFrontPolygonMode(context, SCE_GXM_POLYGON_MODE_TRIANGLE_FILL);
sceGxmSetBackPolygonMode(context, SCE_GXM_POLYGON_MODE_TRIANGLE_FILL);
sceGxmSetFrontStencilFunc(
    context,
    SCE_GXM_STENCIL_FUNC_ALWAYS,
    SCE_GXM_STENCIL_OP_KEEP,
    SCE_GXM_STENCIL_OP_KEEP,
    SCE_GXM_STENCIL_OP_KEEP,
    0,
0);
sceGxmSetBackStencilFunc(
    context,
    SCE_GXM_STENCIL_FUNC_ALWAYS,
    SCE_GXM_STENCIL_OP_KEEP,
    SCE_GXM_STENCIL_OP_KEEP,
    SCE_GXM_STENCIL_OP_KEEP,
    0,
0);
sceGxmSetFrontDepthBias(context, 0, 0);
sceGxmSetBackDepthBias(context, 0, 0);
sceGxmSetTwoSidedEnable(context, SCE_GXM_TWO_SIDED_DISABLED);
sceGxmSetViewport(context, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f);
sceGxmSetWClampValue(context, 0.00001f);
sceGxmSetWClampEnable(context, SCE_GXM_WCLAMP_MODE_ENABLED);
sceGxmSetRegionClip(context, SCE_GXM_REGION_CLIP_NONE, 0, 0, 0, 0);
sceGxmSetCullMode(context, SCE_GXM_CULL_NONE);
sceGxmSetViewportEnable(context, SCE_GXM_VIEWPORT_ENABLED);
sceGxmSetWBufferEnable(context, SCE_GXM_WBUFFER_DISABLED);
sceGxmSetFrontVisibilityTestIndex(context, 0);
sceGxmSetBackVisibilityTestIndex(context, 0);
sceGxmSetFrontVisibilityTestOp(context,
    SCE_GXM_VISIBILITY_TEST_OP_INCREMENT);
sceGxmSetBackVisibilityTestOp(context,
    SCE_GXM_VISIBILITY_TEST_OP_INCREMENT);
sceGxmSetFrontVisibilityTestEnable(context,
    SCE_GXM_VISIBILITY_TEST_DISABLED);
sceGxmSetBackVisibilityTestEnable(context,
    SCE_GXM_VISIBILITY_TEST_DISABLED);
```

Note that the region clip and viewport are reset during `sceGxmBeginScene()`.

When the rendering context is no longer needed, it should be destroyed, by calling `sceGxmDestroyContext()`, after the GPU has finished executing all jobs issued from the context.

sceGxmCreateDeferredContext

Creates a deferred context.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmCreateDeferredContext(
    SceGxmDeferredContextParams *params,
    SceGxmContext **deferredContext
);
```

Arguments

[in] `params`	A pointer to the initialization parameters. This structure does not need to persist after the call.
[out] `deferredContext`	A pointer to storage for an `SceGxmContext` pointer.

Return Values

Value	Description
`SCE_OK`	The operation was successful and `deferredContext` is now populated.
`SCE_GXM_ERROR_UNINITIALIZED`	The operation failed because the library was not initialized.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed due to invalid buffer alignment.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.

Description

Creates a deferred context. This context will use the memory pointed to by the `hostMem` member of `SceGxmDeferredContextParams` for state tracking between draw calls. As the context is used to create a command list, the callbacks and optional initial remaining memory regions described by `SceGxmDeferredContextParams` are used for dynamically generated GPU data structures.

Note that unlike an immediate context, a deferred context only tracks state for draw calls and not GPU jobs. State used at the job level, such as YUV profiles or the visibility buffer base address, may only be set on an immediate context.

After the context has been created, the following states will be set to `NULL`:

- Vertex program
- Fragment program
- Vertex streams
- Vertex and fragment uniform buffers
- Vertex and fragment textures
- Precomputed vertex and fragment state

The following default states are set during creation:

```
sceGxmSetFrontDepthFunc(context, SCE_GXM_DEPTH_FUNC_LESS_EQUAL);
sceGxmSetBackDepthFunc(context, SCE_GXM_DEPTH_FUNC_LESS_EQUAL);
sceGxmSetFrontFragmentProgramEnable(context,
SCE_GXM_FRAGMENT_PROGRAM_ENABLED);
sceGxmSetBackFragmentProgramEnable(context,
    SCE_GXM_FRAGMENT_PROGRAM_ENABLED);
sceGxmSetFrontDepthWriteEnable(context, SCE_GXM_DEPTH_WRITE_ENABLED);
```

```
sceGxmSetBackDepthWriteEnable(context, SCE_GXM_DEPTH_WRITE_ENABLED);
sceGxmSetFrontLineFillLastPixelEnable(context,
    SCE_GXM_LINE_FILL_LAST_PIXEL_DISABLED);
sceGxmSetBackLineFillLastPixelEnable(context,
    SCE_GXM_LINE_FILL_LAST_PIXEL_DISABLED);
sceGxmSetFrontStencilRef(context, 0);
sceGxmSetBackStencilRef(context, 0);
sceGxmSetFrontPointLineWidth(context, 1);
sceGxmSetBackPointLineWidth(context, 1);
sceGxmSetFrontPolygonMode(context, SCE_GXM_POLYGON_MODE_TRIANGLE_FILL);
sceGxmSetBackPolygonMode(context, SCE_GXM_POLYGON_MODE_TRIANGLE_FILL);
sceGxmSetFrontStencilFunc(
    context,
    SCE_GXM_STENCIL_FUNC_ALWAYS,
    SCE_GXM_STENCIL_OP_KEEP,
    SCE_GXM_STENCIL_OP_KEEP,
    SCE_GXM_STENCIL_OP_KEEP,
    0,
0);
sceGxmSetBackStencilFunc(
    context,
    SCE_GXM_STENCIL_FUNC_ALWAYS,
    SCE_GXM_STENCIL_OP_KEEP,
    SCE_GXM_STENCIL_OP_KEEP,
    SCE_GXM_STENCIL_OP_KEEP,
    0,
0);
sceGxmSetFrontDepthBias(context, 0, 0);
sceGxmSetBackDepthBias(context, 0, 0);
sceGxmSetTwoSidedEnable(context, SCE_GXM_TWO_SIDED_DISABLED);
sceGxmSetViewport(context, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f);
sceGxmSetWClampValue(context, 0.00001f);
sceGxmSetWClampEnable(context, SCE_GXM_WCLAMP_MODE_ENABLED);
sceGxmSetRegionClip(context, SCE_GXM_REGION_CLIP_NONE, 0, 0, 0, 0);
sceGxmSetCullMode(context, SCE_GXM_CULL_NONE);
sceGxmSetViewportEnable(context, SCE_GXM_VIEWPORT_ENABLED);
sceGxmSetWBufferEnable(context, SCE_GXM_WBUFFER_DISABLED);
sceGxmSetFrontVisibilityTestIndex(context, 0);
sceGxmSetBackVisibilityTestIndex(context, 0);
sceGxmSetFrontVisibilityTestOp(context,
    SCE_GXM_VISIBILITY_TEST_OP_INCREMENT);
sceGxmSetBackVisibilityTestOp(context,
    SCE_GXM_VISIBILITY_TEST_OP_INCREMENT);
sceGxmSetFrontVisibilityTestEnable(context,
    SCE_GXM_VISIBILITY_TEST_DISABLED);
sceGxmSetBackVisibilityTestEnable(context,
    SCE_GXM_VISIBILITY_TEST_DISABLED);
```

The state set on a deferred context is self-contained and independent from all other contexts. Because of this, before adding draw calls to a command list, it is necessary to manually set at least the region clip and viewport by calling, for example, `sceGxmSetDefaultRegionClipAndViewport()`. The debug version of libgxm will warn when draw calls are performed on a deferred context that has not yet had region clip or viewport set up.

When the context is no longer needed, it should be destroyed by calling `sceGxmDestroyDeferredContext()`. A deferred context does not need to persist in order to use command lists created by that particular deferred context, so it may be safely destroyed once all the command lists have been created.

sceGxmCreateRenderTarget

Creates a render target object.

Definition

```
#include <gxm/render_target.h>
SceGxmErrorCode sceGxmCreateRenderTarget(
    const SceGxmRenderTargetParams *params,
    SceGxmRenderTarget **renderTarget
);
```

Arguments

[in] *params*	The creation parameters for the render target.
[out] *renderTarget*	A pointer to storage for the render target pointer.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_UNINITIALIZED`	The operation failed as libgxm is not initialized.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid parameter value.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to a `NULL` pointer.
`SCE_GXM_ERROR_OUT_OF_RENDER_TARGETS`	The operation failed because the maximum number of render targets have already been created.
`SCE_GXM_ERROR_DRIVER`	The operation failed due to a driver error.

Description

Creates a render target object. A render target defines the layout for tiled rendering and is needed to start a scene and draw geometry. Render targets should ideally be created at load time, since creating them requires resources from the OS.

Once the render target is no longer needed, call `sceGxmDestroyRenderTarget()` to destroy it.

sceGxmDepthStencilSurfaceGetBackgroundDepth

Gets the current background depth value to use if a tile does not read depth values from memory.

Definition

```
#include <gxm/surface.h>
float sceGxmDepthStencilSurfaceGetBackgroundDepth(
   const SceGxmDepthStencilSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the depth/stencil surface.

Return Values

The floating-point background depth value.

Description

Gets the current background depth value to use if a tile does not read depth values from memory.

sceGxmDepthStencilSurfaceGetBackgroundMask

Gets the current background mask value to use if a tile does not read mask values from memory.

Definition

```
#include <gxm/surface.h>
bool sceGxmDepthStencilSurfaceGetBackgroundMask(
    const SceGxmDepthStencilSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the depth/stencil surface.

Return Values

A boolean representing the background mask value.

Description

Gets the current background mask value to use if a tile does not read mask values from memory.

sceGxmDepthStencilSurfaceGetBackgroundStencil

Gets the current background stencil value to use if a tile does not read stencil values from memory.

Definition

```
#include <gxm/surface.h>
uint8_t sceGxmDepthStencilSurfaceGetBackgroundStencil(
   const SceGxmDepthStencilSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the depth/stencil surface.

Return Values

The background stencil value.

Description

Gets the current background stencil value to use if a tile does not read stencil values from memory.

sceGxmDepthStencilSurfaceGetForceLoadMode

Tests if forced loading of depth/stencil values is enabled for a surface.

Definition

```
#include <gxm/surface.h>
SceGxmDepthStencilForceLoadMode sceGxmDepthStencilSurfaceGetForceLoadMode(
    const SceGxmDepthStencilSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the surface.

Return Values

A flag that indicates whether forced loading is enabled or disabled.

Description

Tests if forced loading of depth/stencil values is enabled for a surface.

sceGxmDepthStencilSurfaceGetForceStoreMode

Tests if forced storing of depth/stencil values is enabled for a surface.

Definition

```
#include <gxm/surface.h>
SceGxmDepthStencilForceStoreMode sceGxmDepthStencilSurfaceGetForceStoreMode(
    const SceGxmDepthStencilSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the surface.

Return Values

A flag that indicates whether forced storing is enabled or disabled.

Description

Tests if forced storing of depth/stencil values is enabled for a surface.

sceGxmDepthStencilSurfaceGetFormat

Gets the depth/stencil format of the given surface.

Definition

```
#include <gxm/surface.h>
SceGxmDepthStencilFormat sceGxmDepthStencilSurfaceGetFormat(
    const SceGxmDepthStencilSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the surface.

Return Values

The depth/stencil format of the surface.

Description

Gets the depth/stencil format of the given surface.

sceGxmDepthStencilSurfaceGetStrideInSamples

Gets the stride of the given surface in samples.

Definition

```
#include <gxm/surface.h>
uint32_t sceGxmDepthStencilSurfaceGetStrideInSamples(
    const SceGxmDepthStencilSurface *surface
);
```

Arguments

[in] *surface*	A pointer to the surface.

Return Values

The stride of the surface in samples.

Description

Gets the stride of the given surface in samples. Each sample is of the format returned by `sceGxmDepthStencilSurfaceGetFormat()`.

sceGxmDepthStencilSurfaceInit

Sets up a surface to store depth/stencil data to memory.

Definition

```
#include <gxm/surface.h>
SceGxmErrorCode sceGxmDepthStencilSurfaceInit(
    SceGxmDepthStencilSurface *surface,
    SceGxmDepthStencilFormat depthStencilFormat,
    SceGxmDepthStencilSurfaceType surfaceType,
    uint32_t strideInSamples,
    void *depthData,
    void *stencilData
);
```

Arguments

[out] `surface`	A pointer to the depth/stencil surface.
[in] `depthStencilFormat`	The depth/stencil format.
[in] `surfaceType`	The memory layout type.
[in] `strideInSamples`	The stride of the surface in samples.
[in] `depthData`	A pointer to the depth data or `NULL`.
[in] `stencilData`	A pointer to the stencil data or `NULL`.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to a `NULL` pointer parameter.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed due to an invalid alignment of an input parameter.

Description

Sets up a surface to store depth/stencil data to memory. Note that depth/stencil surfaces store values at sample resolution when using MSAA and may be of linear or tiled layout only. The *`depthData`* and *`stencilData`* parameters must be aligned to `SCE_GXM_DEPTHSTENCIL_SURFACE_ALIGNMENT` bytes. The *`strideInSamples`* parameter must be a multiple of `SCE_GXM_TILE_SIZEX`.

The depth/stencil surface is set up by default not to load or save its values at the beginning or end of a scene respectively. Call `sceGxmDepthStencilSurfaceSetForceLoadMode()` or `sceGxmDepthStencilSurfaceSetForceStoreMode()` to change this behavior.

sceGxmDepthStencilSurfaceInitDisabled

Creates a depth/stencil surface that does not store to memory.

Definition

```
#include <gxm/surface.h>
SceGxmErrorCode sceGxmDepthStencilSurfaceInitDisabled(
    SceGxmDepthStencilSurface *surface
);
```

Arguments

[out] `surface`   A pointer to the depth/stencil surface.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to a `NULL` pointer parameter.

Description

Creates a depth/stencil surface that does not store to memory.

sceGxmDepthStencilSurfaceIsEnabled

Returns true if the given depth/stencil surface is enabled.

Definition

```
#include <gxm/surface.h>
bool sceGxmDepthStencilSurfaceIsEnabled(
    const SceGxmDepthStencilSurface *surface
);
```

Arguments

[in] *surface* A pointer to the depth/stencil surface.

Return Values

Value	Description
`true`	The depth/stencil surface is enabled. The data will be stored at least during partial renders.
`false`	The depth/stencil surface is disabled and no data will ever be stored.

Description

Returns true if the given depth/stencil surface is enabled.

sceGxmDepthStencilSurfaceSetBackgroundDepth

Sets the current background depth value to use if a tile does not read depth values from memory.

Definition

```
#include <gxm/surface.h>
void sceGxmDepthStencilSurfaceSetBackgroundDepth(
    SceGxmDepthStencilSurface *surface,
    float backgroundDepth
);
```

Arguments

[in,out] *surface*	A pointer to the depth/stencil surface.
[in] *backgroundDepth*	The floating-point background depth value.

Return Values

None

Description

Sets the current background depth value to use if a tile does not read depth values from memory. The default value for the background depth is 1.0f.

sceGxmDepthStencilSurfaceSetBackgroundMask

Sets the current background mask value to use if a tile does not read mask values from memory.

Definition

```
#include <gxm/surface.h>
void sceGxmDepthStencilSurfaceSetBackgroundMask(
    SceGxmDepthStencilSurface *surface,
    bool backgroundMask
);
```

Arguments

[in,out] *surface*	A pointer to the depth/stencil surface.
[in] *backgroundMask*	The boolean representing the background mask value.

Return Values

None

Description

Sets the current background mask value to use if a tile does not read mask values from memory. The default value for the background mask is `true`.

sceGxmDepthStencilSurfaceSetBackgroundStencil

Sets the current background stencil value to use if a tile does not read stencil values from memory.

Definition

```
#include <gxm/surface.h>
void sceGxmDepthStencilSurfaceSetBackgroundStencil(
    SceGxmDepthStencilSurface *surface,
    uint8_t backgroundStencil
);
```

Arguments

[in,out] *surface*	A pointer to the depth/stencil surface.
[in] *backgroundStencil*	The background stencil value.

Return Values

None

Description

Sets the current background stencil value to use if a tile does not read stencil values from memory. The default value for the background stencil is `0x00`.

sceGxmDepthStencilSurfaceSetForceLoadMode

Controls forced loading of depth/stencil values at the start of a scene.

Definition

```
#include <gxm/surface.h>
void sceGxmDepthStencilSurfaceSetForceLoadMode(
    SceGxmDepthStencilSurface *surface,
    SceGxmDepthStencilForceLoadMode forceLoad
);
```

Arguments

[in,out] `surface`	A pointer to the surface.
[in] `forceLoad`	A flag that specifies whether to enable or disable forced loading.

Return Values

None

Description

Controls forced loading of depth/stencil values at the start of a scene. You will need to force a load if you wish to load depth or stencil values from a previous scene. This previous scene must have forced a store of its depth/stencil values.

sceGxmDepthStencilSurfaceSetForceStoreMode

Controls forced storing of depth/stencil values at the end of a scene.

Definition

```
#include <gxm/surface.h>
void sceGxmDepthStencilSurfaceSetForceStoreMode(
    SceGxmDepthStencilSurface *surface,
    SceGxmDepthStencilForceStoreMode forceStore
);
```

Arguments

[in,out] `surface`	A pointer to the surface.
[in] `forceStore`	A flag that specifies whether to enable or disable forced storing.

Return Values

None

Description

Controls forced storing of depth/stencil values at the end of a scene. You will need to force a store if you wish to use the depth or stencil values from this scene in a future scene (for example, as a shadow map).

sceGxmDestroyContext

Destroys the given immediate context.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmDestroyContext(
    SceGxmContext *immediateContext
);
```

Arguments

[in,out] *immediateContext*	A pointer to an immediate context.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_UNINITIALIZED`	The operation failed because the library was not initialized.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because the context was not an immediate context.
`SCE_GXM_ERROR_INVALID_POINTER`	The pointer to the context was `NULL`.

Description

Destroys the given immediate context. This function may only be called for contexts created with `sceGxmCreateContext()`. Contexts created with `sceGxmCreateDeferredContext()` should be destroyed by calling `sceGxmDestroyDeferredContext()`.

sceGxmDestroyDeferredContext

Destroys a deferred context.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmDestroyDeferredContext(
    SceGxmContext *deferredContext
);
```

Arguments

[in,out] *deferredContext*	A pointer to a SceGxmContext structure.

Return Values

Value	Description
SCE_OK	The operation was successful and *deferredContext* is now destroyed.
SCE_GXM_ERROR_UNINITIALIZED	The operation failed because the library was not initialized.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to an invalid input pointer.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because the context was not a deferred context.

Description

Destroys a deferred context. This function may only be called for contexts created with sceGxmCreateDeferredContext(). Immediate contexts created with sceGxmCreateContext() should be destroyed by calling sceGxmDestroyContext().

sceGxmDestroyRenderTarget

Destroys a render target object.

Definition

```
#include <gxm/render_target.h>
SceGxmErrorCode sceGxmDestroyRenderTarget(
    SceGxmRenderTarget *renderTarget
);
```

Arguments

[in,out] *renderTarget*	A pointer to the render target to destroy.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_UNINITIALIZED	The operation failed as libgxm is not initialized.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to a NULL pointer.
SCE_GXM_ERROR_DRIVER	The operation failed due to a driver error.

Description

Destroys a render target object.

sceGxmDisplayQueueAddEntry

Adds an entry to the display queue.

Definition

```
#include <gxm/display_queue.h>
SceGxmErrorCode sceGxmDisplayQueueAddEntry(
    SceGxmSyncObject *oldBuffer,
    SceGxmSyncObject *newBuffer,
    const void *callbackData
);
```

Arguments

[in] *oldBuffer*	A pointer to a sync object that is associated with the currently displayed buffer at the time this entry is processed.
[in] *newBuffer*	A pointer to a sync object associated with the buffer that will be displayed by the callback.
[in] *callbackData*	A pointer to the data to be passed to the display callback. This data is copied to internal storage during this call.

Return Values

Value	Description
SCE_OK	The operation was completed successfully.
SCE_GXM_ERROR_UNINITIALIZED	No display queue callback was provided to sceGxmInitialize() as part of the SceGxmInitializeParams structure.
SCE_GXM_ERROR_INVALID_POINTER	A pointer parameter was NULL.
SCE_GXM_ERROR_DRIVER	The operation failed due to a driver error.

Description

Adds an entry to the display queue. The parameters pointed to by *callbackData* are copied to internal storage during this call. Because this copy is passed to the callback there is no need for the parameters to persist once this call has completed.

The display callback will be called from a thread once all queued operations to both the old and new buffers have completed.

sceGxmDisplayQueueFinish

Blocks until all pending display queue entries have completed.

Definition

```
#include <gxm/display_queue.h>
SceGxmErrorCode sceGxmDisplayQueueFinish(void);
```

Arguments

None

Return Values

Value	Description
SCE_OK	The operation was completed successfully.
SCE_GXM_ERROR_UNINITIALIZED	No display queue callback was provided to sceGxmInitialize() as part of the SceGxmInitializeParams structure.

Description

Blocks until all pending display queue entries have completed. This function can be called to ensure that there will be no further calls to the display callback.

sceGxmDraw

Draws indexed geometry.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmDraw(
    SceGxmContext *context,
    SceGxmPrimitiveType primType,
    SceGxmIndexFormat indexType,
    const void *indexData,
    uint32_t indexCount
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *primType*	The type of primitive.
[in] *indexType*	The type of the indices.
[in] *indexData*	A pointer to the index data. The GPU data pointed to must persist until vertex processing for the current scene has completed.
[in] *indexCount*	The number of indices.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a parameter was unexpectedly NULL.
SCE_GXM_ERROR_NOT_WITHIN_SCENE	The operation failed since the call is not between calls to sceGxmBeginScene() and sceGxmEndScene(). This error will only be returned from an immediate context.
SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST	The operation failed since the call is not between calls to sceGxmBeginCommandList() and sceGxmEndCommandList(). This error will only be returned from a deferred context.
SCE_GXM_ERROR_NULL_PROGRAM	The operation failed because a vertex and/or fragment program was not set.
SCE_GXM_ERROR_UNIFORM_BUFFER_NOT_RESERVED	The operation failed because a required default uniform buffer has not been reserved.
SCE_GXM_ERROR_INVALID_DEPTH_STENCIL_CONFIGURATION	The operation failed because of an invalid depth/stencil configuration.

Value	Description
SCE_GXM_ERROR_RESERVE_FAILED	The operation failed due to a buffer callback function failing to provide sufficient memory for the GPU data structures required by this draw call. This error will only be returned from a deferred context. Immediate context reservations always succeed, but they potentially have high latency if scenes are split to recycle memory.
SCE_GXM_ERROR_INVALID_INDEX_COUNT	The operation failed because an invalid index count was supplied. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_POLYGON_MODE	The operation failed because an invalid polygon mode was supplied for the selected primitive type. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_PRIMITIVE_TYPE	The operation failed because a primitive type of SCE_GXM_PRIMITIVE_POINTS was supplied but the current vertex program does not output PSIZE. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_ PRECISION	The operation failed because one of the textures does not support the precision of a query format used in shader code. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_ COMPONENT_COUNT	The operation failed because one of the textures is using a swizzle that produces an incorrect number of components for the query formats used in the shader code. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_SAMPLER_FILTER_MODE	The operation failed because one of the textures has a filtering mode that is incompatible with the query used in the shader code. This error is only returned when running the debug version of libgxm.

Value	Description
`SCE_GXM_ERROR_INVALID_TEXTURE`	The operation failed because a texture was invalid. This can occur if the format or dimensionality of the texture is not compatible with its usage in the shader code, or the texture control words themselves are malformed. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_TEXTURE_DATA_POINTER`	The operation failed because a texture has a `NULL` data pointer. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_TEXTURE_PALETTE_POINTER`	The operation failed because a palettized texture has a `NULL` palette pointer. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_OUTPUT_REGISTER_SIZE`	The operation failed because there is a mismatch in output register size between the fragment program and the color surface. This error is only returned when using the immediate context while running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_FRAGMENT_MSAA_MODE`	The operation failed because there is a mismatch in MSAA usage between the fragment program and the render target. This error can only occur for fragment programs that read the output register to perform blending operations, and it is only returned when using the immediate context while running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_VISIBILITY_BUFFER_POINTER`	The operation failed because visibility testing is enabled but the visibility buffer pointer is `NULL`. This error is only returned when using the immediate context while running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_VISIBILITY_INDEX`	The operation failed because visibility testing is enabled but the visibility index is invalid. This error is only returned when using the immediate context while running the debug version of libgxm.

Value	Description
`SCE_GXM_ERROR_BUFFER_OVERRUN`	The operation failed because the memory directly after the default uniform buffer reservation was modified between the reserve call and the draw call. This indicates a memory overrun when writing the default uniform buffer contents. This error is only returned when running the debug version of libgxm.

Description

Draws indexed geometry. This function may only be called inside a scene or command list. When using the immediate context, calling this function outside of a scene will result in the `SCE_GXM_ERROR_NOT_WITHIN_SCENE` error code being returned. When using a deferred context, calling this function outside of a command list will result in the `SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST` error code being returned.

Drawing geometry is only valid when both the current vertex program and current fragment program are non-`NULL`. If either are `NULL`, this function will return the error code `SCE_GXM_ERROR_NULL_PROGRAM` and nothing will be drawn.

Drawing geometry flushes any pending state changes to the GPU, if any state has changed since the last draw call. To reduce most of the CPU overhead involved in flushing state, precomputed vertex or fragment state can be used via the `SceGxmPrecomputedVertexState` and `SceGxmPrecomputedFragmentState` objects. While using these objects has CPU cost benefits, using precomputed state can mean that you have a larger memory footprint than flushing state dynamically. To further reduce overheads, the draw call itself can be precomputed by using a `SceGxmPrecomputedDraw` object with `sceGxmDrawPrecomputed()`.

If the current vertex or fragment program has a non-empty default uniform buffer, and if you are not using precomputed state for that pipeline, the default uniform buffer needs to have been reserved or set. This is done by calling `sceGxmReserveVertexDefaultUniformBuffer()` or `sceGxmSetVertexDefaultUniformBuffer()` for the vertex pipeline, and by calling `sceGxmReserveFragmentDefaultUniformBuffer()` or `sceGxmSetFragmentDefaultUniformBuffer()` for the fragment pipeline. Failing to reserve or set a (non-empty) default uniform buffer before a draw call will result in the `SCE_GXM_ERROR_UNIFORM_BUFFER_NOT_RESERVED` error code being returned, and nothing will be drawn. Once a default uniform buffer is reserved successfully, it persists until either the scene is ended or a different program is set on that pipeline. When using the same program for consecutive draw calls, it is not necessary to reserve it again before the next draw call unless different uniform values need to be written.

When drawing using the immediate context, the data for all GPU resources (textures, vertex streams and uniform buffers) must have been fully written to memory before the draw function is called. This is because the immediate context may need to split a scene into multiple jobs internally to free up ring buffer space, which potentially causes the GPU to begin processing the draw call immediately after the function returns.

When drawing using a deferred context, the writing of data for GPU resources (textures, vertex streams and uniform buffers) may be deferred until the command list containing the draw call is used on the immediate context. This is done by calling `sceGxmExecuteCommandList()`.

If, when drawing using a deferred context, additional buffer space is required to generate GPU data structures, one or more of the `SceGxmDeferredContextCallback` functions will be called to acquire more space. Should the callback function fail to provide enough memory, the draw call will return an `SCE_GXM_ERROR_RESERVE_FAILED` error, and it will not be present in the current command list. Future draw calls on the deferred context will continue to request memory through the buffer callbacks and fail in the same way if the callback continues to fail. It is always possible to end the command list using `sceGxmEndCommandList()`, and the resulting command list will still be valid to execute on the immediate context using `sceGxmExecuteCommandList()`. However, only draw calls that succeeded without error will be present in the command list.

sceGxmDrawInstanced

Draws multiple instances of indexed geometry.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmDrawInstanced(
    SceGxmContext *context,
    SceGxmPrimitiveType primType,
    SceGxmIndexFormat indexType,
    const void *indexData,
    uint32_t indexCount,
    uint32_t indexWrap
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *primType*	The type of primitive.
[in] *indexType*	The type of the indices.
[in] *indexData*	A pointer to the index data. The GPU data pointed to must persist until vertex processing for the current scene has completed.
[in] *indexCount*	The total number of indices to draw.
[in] *indexWrap*	The number of indices to draw for each instance.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.
`SCE_GXM_ERROR_NOT_WITHIN_SCENE`	The operation failed since the call is not between calls to `sceGxmBeginScene()` and `sceGxmEndScene()`. This error will only be returned from an immediate context.
`SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST`	The operation failed since the call is not between calls to `sceGxmBeginCommandList()` and `sceGxmEndCommandList()`. This error will only be returned from a deferred context.
`SCE_GXM_ERROR_NULL_PROGRAM`	The operation failed because a vertex and/or fragment program was not set.
`SCE_GXM_ERROR_UNIFORM_BUFFER_NOT_RESERVED`	The operation failed because a required default uniform buffer has not been reserved.
`SCE_GXM_ERROR_INVALID_DEPTH_STENCIL_CONFIGURATION`	The operation failed because of an invalid depth/stencil configuration.

Value	Description
SCE_GXM_ERROR_RESERVE_FAILED	The operation failed due to a buffer callback function failing to provide sufficient memory for the GPU data structures required by this draw call. This error will only be returned from a deferred context. Immediate context reservations always succeed, but they potentially have high latency if scenes are split to recycle memory.
SCE_GXM_ERROR_INVALID_INDEX_COUNT	The operation failed because an invalid index count or index wrap was supplied. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_POLYGON_MODE	The operation failed because an invalid polygon mode was supplied for the selected primitive type. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_PRIMITIVE_TYPE	The operation failed because a primitive type of SCE_GXM_PRIMITIVE_POINTS was supplied but the current vertex program does not output PSIZE. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_PRECISION	The operation failed because one of the textures does not support the precision of a query format used in shader code. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_COMPONENT_COUNT	The operation failed because one of the textures is using a swizzle that produces an incorrect number of components for the query formats used in the shader code. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_SAMPLER_FILTER_MODE	The operation failed because one of the textures has a filtering mode that is incompatible with the query used in the shader code. This error is only returned when running the debug version of libgxm.

Value	Description
SCE_GXM_ERROR_INVALID_TEXTURE	The operation failed because a texture was invalid. This can occur if the format or dimensionality of the texture is not compatible with its usage in the shader code, or the texture control words themselves are malformed. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_TEXTURE_DATA_POINTER	The operation failed because a texture has a `NULL` data pointer. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_TEXTURE_PALETTE_POINTER	The operation failed because a palettized texture has a `NULL` palette pointer. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_OUTPUT_REGISTER_SIZE	The operation failed because there is a mismatch in output register size between the fragment program and the color surface. This error is only returned when using the immediate context while running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_FRAGMENT_MSAA_MODE	The operation failed because there is a mismatch in MSAA usage between the fragment program and the render target. This error can only occur for fragment programs that read the output register to perform blending operations, and it is only returned when using the immediate context while running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_VISIBILITY_BUFFER_POINTER	The operation failed because visibility testing is enabled but the visibility buffer pointer is `NULL`. This error is only returned when using the immediate context while running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_VISIBILITY_INDEX	The operation failed because visibility testing is enabled but the visibility index is invalid. This error is only returned when using the immediate context while running the debug version of libgxm.

Value	Description
SCE_GXM_ERROR_BUFFER_OVERRUN	The operation failed because the memory directly after the default uniform buffer reservation was modified between the reserve call and the draw call. This indicates a memory overrun when writing the default uniform buffer contents. This error is only returned when running the debug version of libgxm.

Description

Draws multiple instances of indexed geometry. The position within the index buffer is reset each time *indexWrap* indices are processed until a total of *indexCount* indices have been processed. Each instance must be a whole number of primitives. Triangle strips and triangle fans are reset for the start of each instance.

This function may only be called inside a scene or command list. When using the immediate context, calling this function outside of a scene will result in the SCE_GXM_ERROR_NOT_WITHIN_SCENE error code being returned. When using a deferred context, calling this function outside of a command list will result in the SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST error code being returned.

Drawing geometry is only valid when both the current vertex program and current fragment program are non-NULL. If either are NULL, this function will return the error code SCE_GXM_ERROR_NULL_PROGRAM and nothing will be drawn.

Drawing geometry flushes any pending state changes to the GPU, if any state has changed since the last draw call. To reduce most of the CPU overhead involved in flushing state, precomputed vertex or fragment state can be used via the SceGxmPrecomputedVertexState and SceGxmPrecomputedFragmentState objects. While using these objects has CPU cost benefits, using precomputed state can mean that you have a larger memory footprint than flushing state dynamically. To further reduce overheads, the draw call itself can be precomputed by using a SceGxmPrecomputedDraw object with sceGxmDrawPrecomputed().

If the current vertex or fragment program has a non-empty default uniform buffer, and if you are not using precomputed state for that pipeline, the default uniform buffer needs to have been reserved or set. This is done by calling sceGxmReserveVertexDefaultUniformBuffer() or sceGxmSetVertexDefaultUniformBuffer() for the vertex pipeline, and by calling sceGxmReserveFragmentDefaultUniformBuffer() or sceGxmSetFragmentDefaultUniformBuffer() for the fragment pipeline. Failing to reserve or set a (non-empty) default uniform buffer before a draw call will result in the SCE_GXM_ERROR_UNIFORM_BUFFER_NOT_RESERVED error code being returned, and nothing will be drawn. Once a default uniform buffer is reserved successfully, it persists until either the scene is ended or a different program is set on that pipeline. When using the same program for consecutive draw calls, it is not necessary to reserve it again before the next draw call unless different uniform values need to be written.

When drawing using the immediate context, the data for all GPU resources (textures, vertex streams and uniform buffers) must have been fully written to memory before the draw function is called. This is because the immediate context may need to split a scene into multiple jobs internally to free up ring buffer space, which potentially causes the GPU to begin processing the draw call immediately after the function returns.

When drawing using a deferred context, the writing of data for GPU resources (textures, vertex streams and uniform buffers) may be deferred until the command list containing the draw call is used on the immediate context. This is done by calling sceGxmExecuteCommandList().

If, when drawing using a deferred context, additional buffer space is required to generate GPU data structures, one or more of the SceGxmDeferredContextCallback functions will be called to acquire more space. Should the callback function fail to provide enough memory, the draw call will return an `SCE_GXM_ERROR_RESERVE_FAILED` error, and it will not be present in the current command list. Future draw calls on the deferred context will continue to request memory through the buffer callbacks and fail in the same way if the callback continues to fail. It is always possible to end the command list using `sceGxmEndCommandList()`, and the resulting command list will still be valid to execute on the immediate context using `sceGxmExecuteCommandList()`. However, only draw calls that succeeded without error will be present in the command list.

sceGxmDrawPrecomputed

Draws indexed geometry using a precomputed draw object.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmDrawPrecomputed(
    SceGxmContext *context,
    const SceGxmPrecomputedDraw *precomputedDraw
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *precomputedDraw*	A pointer to the precomputed draw commands. The precomputed draw commands must not be released until another program or `NULL` is set on the `SceGxmContext`, and until fragment processing for the current scene has completed.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.
`SCE_GXM_ERROR_NOT_WITHIN_SCENE`	The operation failed since the call is not between calls to `sceGxmBeginScene()` and `sceGxmEndScene()`. This error will only be returned from an immediate context.
`SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST`	The operation failed since the call is not between calls to `sceGxmBeginCommandList()` and `sceGxmEndCommandList()`. This error will only be returned from a deferred context.
`SCE_GXM_ERROR_NULL_PROGRAM`	The operation failed because a vertex and/or fragment program was not set.
`SCE_GXM_ERROR_UNIFORM_BUFFER_NOT_RESERVED`	The operation failed because a required default uniform buffer has not been reserved.
`SCE_GXM_ERROR_INVALID_DEPTH_STENCIL_CONFIGURATION`	The operation failed because of an invalid depth/stencil configuration.

Value	Description
SCE_GXM_ERROR_RESERVE_FAILED	The operation failed due to a buffer callback function failing to provide sufficient memory for the GPU data structures required by this draw call. This error will only be returned from a deferred context. Immediate context reservations always succeed, but they potentially have high latency if scenes are split to recycle memory.
SCE_GXM_ERROR_INVALID_PRECOMPUTED_FRAGMENT_STATE	The operation failed because the precomputed fragment state program does not match the current fragment program. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_PRECOMPUTED_VERTEX_STATE	The operation failed because the precomputed vertex state program does not match the current vertex program. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_PRECOMPUTED_DRAW	The operation failed because the precomputed draw call vertex program does not match the current vertex program. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_INDEX_COUNT	The operation failed because an invalid index count or index wrap was supplied. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_POLYGON_MODE	The operation failed due to an invalid polygon mode. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_PRIMITIVE_TYPE	The operation failed because a primitive type of SCE_GXM_PRIMITIVE_POINTS was supplied but the current vertex program does not output PSIZE. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_PRECISION	The operation failed because one of the textures does not support the precision of a query format used in shader code. This error is only returned when running the debug version of libgxm.

Value	Description
`SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_COMPONENT_COUNT`	The operation failed because one of the textures is using a swizzle that produces an incorrect number of components for the query formats used in the shader code. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_SAMPLER_FILTER_MODE`	The operation failed because one of the textures has a filtering mode that is incompatible with the query used in the shader code. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_TEXTURE`	The operation failed because a texture was invalid. This can occur if the format or dimensionality of the texture is not compatible with its usage in the shader code. It can also occur if the texture control words themselves are malformed. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_TEXTURE_DATA_POINTER`	The operation failed because a texture has a `NULL` data pointer. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_TEXTURE_PALETTE_POINTER`	The operation failed because a palettized texture has a `NULL` palette pointer. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_OUTPUT_REGISTER_SIZE`	The operation failed because there is a mismatch in output register size between the fragment program and the color surface. This error is only returned when using the immediate context while running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_FRAGMENT_MSAA_MODE`	The operation failed because there is a mismatch in MSAA usage between the fragment program and the render target. This error can only occur for fragment programs that read the output register to perform blending operations, and it is only returned when using the immediate context while running the debug version of libgxm.

Value	Description
SCE_GXM_ERROR_INVALID_VISIBILITY_BUFFER_POINTER	The operation failed because visibility testing is enabled but the visibility buffer pointer is NULL. This error is only returned when using the immediate context while running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_VISIBILITY_INDEX	The operation failed because visibility testing is enabled but the visibility index is invalid. This error is only returned when using the immediate context while running the debug version of libgxm.
SCE_GXM_ERROR_BUFFER_OVERRUN	The operation failed because the memory directly after the default uniform buffer reservation was modified between the reserve call and the draw call. This indicates a memory overrun when writing the default uniform buffer contents. This error is only returned when running the debug version of libgxm.

Description

Draws indexed geometry using a precomputed draw object. Drawing geometry using a precomputed draw object overrides all vertex streams currently set on the context. Instead vertex streams are used that have been set on the precomputed draw object.

This function may only be called inside a scene or command list. When using the immediate context, calling this function outside of a scene will result in the SCE_GXM_ERROR_NOT_WITHIN_SCENE error code being returned. When using a deferred context, calling this function outside of a command list will result in the SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST error code being returned.

Drawing geometry is only valid when both the current vertex program and current fragment program are non-NULL. If either are NULL, this function will return the error code SCE_GXM_ERROR_NULL_PROGRAM and nothing will be drawn.

Drawing geometry flushes any pending state changes to the GPU, if any state has changed since the last draw call. To reduce most of the CPU overhead involved in flushing state, precomputed vertex or fragment state can be used via the SceGxmPrecomputedVertexState and SceGxmPrecomputedFragmentState objects. While using these objects has CPU cost benefits, using precomputed state can mean that you have a larger memory footprint than flushing state dynamically.

If the current vertex or fragment program has a non-empty default uniform buffer, and if the precomputed state is not being used for that pipeline, the default uniform buffer needs to have been reserved or set. This is done by calling sceGxmReserveVertexDefaultUniformBuffer() or sceGxmSetVertexDefaultUniformBuffer() for the vertex pipeline, and by calling sceGxmReserveFragmentDefaultUniformBuffer() or sceGxmSetFragmentDefaultUniformBuffer() for the fragment pipeline. Failing to reserve or set a (non-empty) default uniform buffer before a draw call will result in the SCE_GXM_ERROR_UNIFORM_BUFFER_NOT_RESERVED error code being returned, and nothing will be drawn. Once a default uniform buffer is reserved successfully, it persists until either the scene is ended or a different program is set on that pipeline. When using the same program for consecutive draw calls, it is not necessary to reserve it again before the next draw call unless different uniform values need to be written.

When drawing using the immediate context, the data for all GPU resources (textures, vertex streams and uniform buffers) needs to have been fully written to memory before the draw function is called. This is because the immediate context may need to split a scene into multiple jobs internally to free up ring buffer space, which potentially causes the GPU to begin processing the draw call immediately after the function returns.

When drawing using a deferred context, the writing of data for GPU resources (textures, vertex streams and uniform buffers) may be deferred until the command list containing the draw call is used on the immediate context. This is done by calling `sceGxmExecuteCommandList()`.

If, when drawing using a deferred context, additional buffer space is required to generate GPU data structures, one or more of the `SceGxmDeferredContextCallback` functions will be called to acquire more space. Should the callback function fail to provide enough memory, the draw call will return an `SCE_GXM_ERROR_RESERVE_FAILED` error code, and it will not be present in the current command list. Future draw calls on the deferred context will continue to request memory through the buffer callbacks and fail in the same way if the callback continues to fail. It is always possible to end the command list using `sceGxmEndCommandList()`, and the resulting command list will still be valid to execute on the immediate context using `sceGxmExecuteCommandList()`. However, only draw calls that succeeded without error will be present in the command list.

sceGxmEndCommandList

Ends rendering to a command list.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmEndCommandList(
    SceGxmContext *deferredContext,
    SceGxmCommandList *commandList
);
```

Arguments

[in,out] *deferredContext*	A pointer to a deferred context.
[out] *commandList*	A pointer to a SceGxmCommandList.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed as a pointer to the deferred context or command list was NULL.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed as the context is not a deferred context.
SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST	The operation failed since the deferred context is not currently within a command list.

Description

Ends rendering to a command list. This function is only supported on a deferred context, and it will return the SCE_GXM_ERROR_INVALID_VALUE error code if called using the immediate context.

The implementation of this function will not trigger a memory callback function, so it will always succeed even if draw calls within the command list failed due to lack of memory.

Once the function has completed successfully, the resulting SceGxmCommandList may be executed on the immediate context using sceGxmExecuteCommandList().

Ending a command list will cause the current default uniform buffer reservations to be lost. These must be reserved again using sceGxmBeginCommandList() if a new command list is started on this deferred context. All other state on the deferred context will persist for future command lists, but no internal GPU data structures persist to the next command list. This ensures that the deferred context buffers can be reset between command lists, and that new command lists only reference GPU data structures that are generated using the new buffers.

Various state is stored with the command list such as the maximum region clip dimensions, and whether the draw calls within the region clip require depth, stencil or mask data to be part of the depth/stencil format. This state will be validated during sceGxmExecuteCommandList(), and it must be compatible with the scene the command list will be made part of. sceGxmExecuteCommandList() will return an error if the state is not compatible. Please see the documentation of sceGxmExecuteCommandList() for more details.

sceGxmEndScene

Ends the scene, which immediately creates and submits a job to the firmware layer.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmEndScene(
    SceGxmContext *immediateContext,
    const SceGxmNotification *vertexNotification,
    const SceGxmNotification *fragmentNotification
);
```

Arguments

[in,out] *immediateContext*	A pointer to the immediate context.
[in] *vertexNotification*	A pointer to a notification object used to identify completion of vertex processing. Set to `NULL` if this is not required.
[in] *fragmentNotification*	A pointer to a notification object used to identify completion of fragment processing. Set to `NULL` if this is not required.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because the context is not an immediate context.
`SCE_GXM_ERROR_NOT_WITHIN_SCENE`	The operation failed because there is no matching call to `sceGxmBeginScene()`.

Description

Ends the scene, which immediately creates and submits a job to the firmware layer. This function is only supported on the immediate context, and it will return the `SCE_GXM_ERROR_INVALID_VALUE` error code if called using a deferred context.

The optional notification objects can be used to signal that the GPU has finished processing the scene on the vertex or fragment pipeline.

sceGxmExecuteCommandList

Links a command list into the current scene to be executed by the GPU.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmExecuteCommandList(
    SceGxmContext *immediateContext,
    SceGxmCommandList *commandList
);
```

Arguments

[in,out] *immediateContext*	A pointer to an immediate context.
[in,out] *commandList*	A pointer to a `SceGxmCommandList`.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a pointer to the immediate context or command list was `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because the *immediateContext* parameter does not point to the immediate context, or the command list is not valid.
`SCE_GXM_ERROR_NOT_WITHIN_SCENE`	The operation failed because the call is not between calls to `sceGxmBeginScene()` and `sceGxmEndScene()`. This function must be called within a scene.
`SCE_GXM_ERROR_INVALID_REGION_CLIP_IN_COMMAND_LIST`	The operation failed because a draw call within the command list uses a region clip outside the bounds of the scene valid region.
`SCE_GXM_ERROR_INVALID_DEPTH_STENCIL_CONFIGURATION`	The operation failed because a draw call within the command list using a depth test, stencil test or mask update is not supported by the store depth/stencil buffer.
`SCE_GXM_ERROR_INVALID_VISIBILITY_BUFFER_POINTER`	The operation failed because visibility testing is used by draw calls within the command list, but the visibility buffer pointer is `NULL`. This error is only returned when running the debug version of libgxm.

Value	Description
`SCE_GXM_ERROR_INVALID_VISIBILITY_INDEX`	The operation failed because visibility testing, which uses a visibility index that is greater than the maximum value supported by the visibility buffer stride, is used by draw calls within the command list. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_OUTPUT_REGISTER_SIZE`	The operation failed because there is a output register size mismatch between a fragment program used by draw calls within the command list and the color surface. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_FRAGMENT_MSAA_MODE`	The operation failed because there is a MSAA usage mismatch between a fragment program used by draw calls within the command list and the render target. This error can only occur for fragment programs that read the output register to perform blending operations, and it is only returned when running the debug version of libgxm.

Description

Links a command list into the current scene to be executed by the GPU. This function is only supported on the immediate context, and it will return the `SCE_GXM_ERROR_INVALID_VALUE` error code if called using a deferred context.

Executing a command list is implemented by placing jump commands in the VDM stream executed by the GPU during vertex processing. A jump from the immediate context will jump to the start of the command list, and the command list VDM stream is patched to jump back to the immediate context. Because of this, the command list must not be executed in another scene until vertex processing for this scene or, if using `sceGxmMidSceneFlush()`, job is completed.

To ensure the GPU is not given an illegal combination of region clip and valid region, the tile-aligned maximum Y value of region clip over all draw calls in the command list must be at most the tile-aligned maximum Y value of the valid region of the current scene. If this condition is not met, the command list execution will fail, and the `SCE_GXM_ERROR_INVALID_REGION_CLIP_IN_COMMAND_LIST` error code will be returned. This clamping operation is performed automatically for region clip state on the immediate context; for command lists, this restriction must only be handled manually since the valid region is not known at the time the command list is built on a deferred context.

To ensure correct behavior during a partial render, if the command list contains draw calls that use depth tests other than NEVER or ALWAYS, the store depth/stencil surface for this scene must contain depth data as part of its format. Similarly, if the command list contains draw calls that use a stencil test other than NEVER or ALWAYS, the store depth/stencil surface for this scene must contain stencil data as part of its format. If the command contains draw calls that use a mask update fragment program, the store depth/stencil surface for this scene must contain mask data as part of its format. If any of these conditions are not met, executing the command list will fail, and a `SCE_GXM_ERROR_INVALID_DEPTH_STENCIL_CONFIGURATION` error code will be returned.

sceGxmFinish

Blocks until all rendering has finished on the GPU.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmFinish(
    SceGxmContext *context
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_THREAD`	The operation failed because the function was called from the display queue thread.

Description

Blocks until all rendering has finished on the GPU. Blocks until all rendering up to and including the last call to `sceGxmEndScene()` has completed.

sceGxmFragmentProgramGetPassType

Gets the pass type for the given fragment program.

Definition

```
#include <gxm/fragment_program.h>
SceGxmPassType sceGxmFragmentProgramGetPassType(
    const SceGxmFragmentProgram *fragmentProgram
);
```

Arguments

[in] *fragmentProgram*	A pointer to the fragment program. This must not be `NULL`.

Return Values

The pass type of the program.

Description

Gets the pass type for the given fragment program.

sceGxmFragmentProgramGetProgram

Gets the underlying program for the given fragment program.

Definition

```
#include <gxm/fragment_program.h>
const SceGxmProgram *sceGxmFragmentProgramGetProgram(
    const SceGxmFragmentProgram *fragmentProgram
);
```

Arguments

[in] *fragmentProgram*	A pointer to the fragment program. This must not be NULL.

Return Values

A pointer to the program.

Description

Gets the underlying program for the given fragment program.

sceGxmFragmentProgramIsEnabled

Tests whether this fragment program performs fragment shading.

Definition

```
#include <gxm/fragment_program.h>
bool sceGxmFragmentProgramIsEnabled(
    const SceGxmFragmentProgram *fragmentProgram
);
```

Arguments

[in] *fragmentProgram*	A pointer to the fragment program. This must not be `NULL`.

Return Values

If the fragment program performs fragment shading, true is returned; otherwise false is returned, which indicates fragment shading is disabled for this fragment program.

Description

Tests whether this fragment program performs fragment shading. If a fragment program has no side-effects, the shader patcher will configure the fragment program to disable fragment shading entirely. During rendering, this setting is combined with `sceGxmSetFrontFragmentProgramEnable()` and `sceGxmSetBackFragmentProgramEnable()`. This has the effect that either setting can disable fragment shading.

Fragment shading is disabled by the shader patcher if the fragment program satisfies the following requirements:

- The output register is not modified. This is to say the blend or mask operation has no effect on the output pixel either through programmable blending or blending code added by the shader patcher.
- A discard or depth replace is not used. Fragment shading is required in order for the GPU to perform a discard or depth replace.
- Writeable uniform buffers are not written to.

sceGxmGetContextType

Gets the type of a context.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmGetContextType(
    const SceGxmContext *context,
    SceGxmContextType *type
);
```

Arguments

[in] *context*	A pointer to a context.
[out] *type*	A pointer to storage for a `SceGxmContextType` value.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.

Description

Gets the type of a context.

sceGxmGetDeferredContextFragmentBuffer

Gets the current write address within the fragment buffer of a deferred context.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmGetDeferredContextFragmentBuffer(
    const SceGxmContext *deferredContext,
    void **mem
);
```

Arguments

[in] *deferredContext*	A pointer to a deferred context.
[out] *mem*	Receives the current fragment buffer write address.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a pointer was NULL.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because the *deferredContext* parameter does not point to a deferred context.
SCE_GXM_ERROR_WITHIN_COMMAND_LIST	The operation failed because the deferred context is already within a command list.

Description

Gets the current write address within the fragment buffer of a deferred context. This function is only supported on a deferred context, and it will return the SCE_GXM_ERROR_INVALID_VALUE error code if called using the immediate context.

This function is only supported when the deferred context is not currently building a command list. Calling this function within a command list will result in the SCE_GXM_ERROR_WITHIN_COMMAND_LIST error code being returned.

sceGxmGetDeferredContextVdmBuffer

Gets the current write address within the VDM buffer of a deferred context.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmGetDeferredContextVdmBuffer(
    const SceGxmContext *deferredContext,
    void **mem
);
```

Arguments

[in] *deferredContext*	A pointer to a deferred context.
[out] *mem*	Receives the current VDM buffer write address.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a pointer was `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because the *deferredContext* parameter does not point to a deferred context.
`SCE_GXM_ERROR_WITHIN_COMMAND_LIST`	The operation failed because the deferred context is already within a command list.

Description

Gets the current write address within the VDM buffer of a deferred context. This function is only supported on a deferred context, and it will return the `SCE_GXM_ERROR_INVALID_VALUE` error code if called using the immediate context.

This function is only supported when the deferred context is not currently building a command list. Calling this function within a command list will the result in the `SCE_GXM_ERROR_WITHIN_COMMAND_LIST` error code being returned.

sceGxmGetDeferredContextVertexBuffer

Gets the current write address within the vertex buffer of a deferred context.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmGetDeferredContextVertexBuffer(
    const SceGxmContext *deferredContext,
    void **mem
);
```

Arguments

[in] *deferredContext*	A pointer to a deferred context.
[out] *mem*	Receives the current vertex buffer write address.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a pointer was set to NULL.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because the *deferredContext* parameter does not point to a deferred context.
SCE_GXM_ERROR_WITHIN_COMMAND_LIST	The operation failed because the deferred context is already within a command list.

Description

Gets the current write address within the vertex buffer of a deferred context. This function is only supported on a deferred context, and it will return the SCE_GXM_ERROR_INVALID_VALUE error code if called using the immediate context.

This function is only supported when the deferred context is not currently building a command list. Calling this function within a command list will result in the SCE_GXM_ERROR_WITHIN_COMMAND_LIST error code being returned.

sceGxmGetNotificationRegion

Gets the start of the notification region created during `sceGxmInitialize()`.

Definition

```
#include <gxm/init.h>
uint32_t *sceGxmGetNotificationRegion(void);
```

Arguments

None

Return Values

A pointer to a region of memory which can hold 32-bit values for `SceGxmNotification` objects.

Description

Gets the start of the notification region created during `sceGxmInitialize()`. Within this region, `SCE_GXM_NOTIFICATION_COUNT` 32-bit notification values can be used.

All values within the notification region are initialized to zero when libgxm is initialized.

sceGxmGetParameterBufferThreshold

Returns the parameter buffer size used for geometry data.

Definition

```
#include <gxm/init.h>
SceGxmErrorCode sceGxmGetParameterBufferThreshold(
    uint32_t *parameterBufferSize
);
```

Arguments

[out] *parameterBufferSize*	A pointer to storage for the parameter buffer geometry data memory size.

Return Values

Value	Description
`SCE_OK`	The operation completed successfully.
`SCE_GXM_ERROR_UNINITIALIZED`	The operation failed because the library was not initialized.
`SCE_GXM_ERROR_INVALID_POINTER`	The *parameterBufferSize* pointer was `NULL`.

Description

Returns the parameter buffer size used for geometry data. Specifically, this is the size of the parameter buffer memory area reserved during the call to `sceGxmInitialize()`, whose size is specified by `SceGxmInitializeParams.parameterBufferSize` minus the memory area reserved internally to store page management data and for use during partial renders.

sceGxmGetPrecomputedDrawSize

Computes the amount of memory needed for a precomputed draw for the given vertex program.

Definition

```
#include <gxm/precomputation.h>
uint32_t sceGxmGetPrecomputedDrawSize(
    const SceGxmVertexProgram *vertexProgram
);
```

Arguments

[in] *vertexProgram*	A pointer to the vertex program.

Return Values

The memory needed for the precomputed draw in bytes.

Description

Computes the amount of memory needed for a precomputed draw for the given vertex program.

sceGxmGetPrecomputedFragmentStateSize

Computes the amount of memory needed for precomputed fragment state for the given fragment program.

Definition

```
#include <gxm/precomputation.h>
uint32_t sceGxmGetPrecomputedFragmentStateSize(
    const SceGxmFragmentProgram *fragmentProgram
);
```

Arguments

[in] *fragmentProgram*	A pointer to the fragment program.

Return Values

The size of the state in bytes.

Description

Computes the amount of memory needed for precomputed fragment state for the given fragment program.

sceGxmGetPrecomputedVertexStateSize

Computes the amount of memory needed for precomputed vertex state for the given vertex program.

Definition

```
#include <gxm/precomputation.h>
uint32_t sceGxmGetPrecomputedVertexStateSize(
    const SceGxmVertexProgram *vertexProgram
);
```

Arguments

[in] *vertexProgram*	A pointer to the vertex program.

Return Values

The size in bytes of the vertex program state.

Description

Computes the amount of memory needed for precomputed vertex state for the given vertex program.

sceGxmGetRenderTargetMemSize

Computes the driver memory size needed for the given set of render target parameters.

Definition

```
#include <gxm/render_target.h>
SceGxmErrorCode sceGxmGetRenderTargetMemSize(
    const SceGxmRenderTargetParams *params,
    uint32_t *driverMemSize
);
```

Arguments

[in] *params*	A pointer to render target parameters.
[out] *driverMemSize*	A pointer to storage for the driver memory size.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid parameter value.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to a `NULL` pointer.

Description

Computes the driver memory size needed for the given set of render target parameters. The memory will be used for render target GPU data structures should be allocated as an uncached LPDDR memblock using `sceKernelAllocMemBlock()`.

sceGxmInitialize

Initializes the libgxm library.

Definition

```
#include <gxm/init.h>
SceGxmErrorCode sceGxmInitialize(
    const SceGxmInitializeParams *params
);
```

Arguments

[in] *params*	A pointer to a populated `SceGxmInitializeParams` structure.

Return Values

Value	Description
`SCE_OK`	The operation completed successfully.
`SCE_GXM_ERROR_INVALID_POINTER`	The `SceGxmInitializeParams` pointer was `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	One or more parameters were invalid.
`SCE_GXM_ERROR_ALREADY_INITIALIZED`	The operation failed because libgxm is already initialized.
`SCE_GXM_ERROR_OUT_OF_MEMORY`	There was no memory to perform the operation.
`SCE_GXM_ERROR_DRIVER`	The operation failed due to a driver error.

Description

Initializes the libgxm library. Internally this function will prepare this process for rendering, creating the parameter buffer with the given size.

This function must be called before any other libgxm object is created, such as a context, or sync object.

sceGxmIsDebugVersion

Determines whether the debug version of libgxm is currently being used.

Definition

```
#include <gxm/init.h>
bool sceGxmIsDebugVersion(void);
```

Arguments

None

Return Values

Value	Description
`true`	The debug version of libgxm is being used.
`false`	The debug version of libgxm is not being used.

Description

Determines whether the debug version of libgxm is currently being used.

sceGxmMapFragmentUsseMemory

Maps memory for fragment USSE code usage.

Definition

```
#include <gxm/memory.h>
SceGxmErrorCode sceGxmMapFragmentUsseMemory(
    void *base,
    uint32_t size,
    uint32_t *offset
);
```

Arguments

[in] *base*	A 4K-aligned base address of the region to map.
[in] *size*	A 4K-aligned size in bytes of the region to map. This cannot be greater than 8MB.
[in] *offset*	A pointer to a 32-bit value to hold the USSE offset.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_GXM_ERROR_UNINITIALIZED`	The operation failed because the library was not initialized.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a pointer was invalid.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because a parameter was invalid.
`SCE_GXM_ERROR_DRIVER`	The operation failed due to a driver error.

Description

Maps memory for fragment USSE code usage. If successful, this mapping operation returns a USSE offset to address the memory as fragment USSE code.

sceGxmMapMemory

Maps memory for GPU usage.

Definition

```
#include <gxm/memory.h>
SceGxmErrorCode sceGxmMapMemory(
    void *base,
    uint32_t size,
    uint32_t attribs
);
```

Arguments

[in] *base*	A 4K-aligned base address of the region to map.
[in] *size*	A 4K-aligned size in bytes of the region to map.
[in] *attribs*	Bitwise combined attributes from `SceGxmMemoryAttribFlags`.

Return Values

Value	Description
SCE_OK	The operation was completed successfully.
SCE_GXM_ERROR_UNINITIALIZED	The operation failed because the library was not initialized.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a pointer was invalid.
SCE_GXM_ERROR_DRIVER	The operation failed due to a driver error.

Description

Maps memory for GPU usage. Once mapped, pointers within the region of memory described by *base* and *size* may be used with libgxm functions directly. It is not valid to call this function with a memory range where all or part of that range has already been mapped.

sceGxmMapVertexUsseMemory

Maps memory for vertex USSE code usage.

Definition

```
#include <gxm/memory.h>
SceGxmErrorCode sceGxmMapVertexUsseMemory(
    void *base,
    uint32_t size,
    uint32_t *offset
);
```

Arguments

[in] *base*	A 4K-aligned base address of the region to map.
[in] *size*	A 4K-aligned size in bytes of the region to map. This cannot be greater than 8MB.
[in] *offset*	A pointer to a 32-bit value to hold the USSE offset.

Return Values

Value	Description
SCE_OK	The operation was completed successfully.
SCE_GXM_ERROR_UNINITIALIZED	The operation failed because the library was not initialized.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a pointer was invalid.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because a parameter was invalid.
SCE_GXM_ERROR_DRIVER	The operation failed due to a driver error.

Description

Maps memory for vertex USSE code usage. If successful, this mapping operation returns a USSE offset to address the memory as vertex USSE code.

sceGxmMidSceneFlush

Flushes vertex processing, creating and submitting a vertex-processing-only job to the firmware layer.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmMidSceneFlush(
    SceGxmContext *immediateContext,
    uint32_t flags,
    SceGxmSyncObject *vertexSyncObject,
    const SceGxmNotification *vertexNotification
);
```

Arguments

[in,out] *immediateContext*	A pointer to the immediate context.
[in] *flags*	Bitwise combined flags from SceGxmMidSceneFlags.
[in] *vertexSyncObject*	Reserved for future use. Must be NULL.
[in] *vertexNotification*	A Pointer to a notification object used to identify completion of vertex processing. Set to NULL if this is not required.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a parameter was unexpectedly NULL.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because the context is not an immediate context.
SCE_GXM_ERROR_NOT_WITHIN_SCENE	The operation failed because there is no matching call to sceGxmBeginScene().

Description

Flushes vertex processing, creating and submitting a vertex-processing-only job to the firmware layer. This function is only supported on the immediate context, and it will return the SCE_GXM_ERROR_INVALID_VALUE error code if called using a deferred context.

The default behavior of this flush operation is to lose the reservation for the vertex default uniform buffer. To preserve this buffer, pass SCE_GXM_MIDSCENE_PRESERVE_DEFAULT_UNIFORM_BUFFERS as the *flags* parameter. This has some overhead due to the underlying copy operation, but allows the caller to continue calling draw functions without having to manually reserve and write the vertex default uniform buffer contents again. This flag has no effect if the current vertex program is NULL or the vertex default uniform buffer has not yet been reserved.

The optional notification object can be used to signal that the GPU has finished processing this job on the vertex pipeline. This can be used to synchronize resources used by the vertex pipeline with the GPU, such as dynamic vertex data.

sceGxmNotificationWait

Waits until a given notification has completed.

Definition

```
#include <gxm/init.h>
SceGxmErrorCode sceGxmNotificationWait(
    const SceGxmNotification *notification
);
```

Arguments

[in] *notification*	A pointer to the notification struct.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed due to the notification pointer being out of range.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to the notification pointer being NULL.
SCE_GXM_ERROR_INVALID_THREAD	The operation failed because the function was called from the display queue thread.

Description

Waits until a given notification has completed. A notification is considered to be completed when the following is true:

```
*notification->address == notification->value
```

Note that this function blocks execution (suspending the calling thread) until the notification has completed.

This function is equivalent to polling the notification with a call to sceGxmWaitEvent() between polling attempts.

sceGxmPadHeartbeat

Heartbeat function for PA.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmPadHeartbeat(
    const SceGxmColorSurface *displaySurface,
    SceGxmSyncObject *displaySyncObject
);
```

Arguments

[in] *displaySurface*	A surface to display, usually the back buffer.
[in] *displaySyncObject*	The display sync object to use with this surface.

Return Values

Value	Description
SCE_OK	The operation was successful.

Description

Heartbeat function for PA. This function should be called after the last scene of your frame, before adding a display queue entry to swap the buffers.

Notes

The *displaySurface* and *displaySyncObject* arguments can be NULL when Razor HUD display is not required. In this case, Razor HUD display cannot be enabled. However, other features such as GPU Traces and GPU Live Metrics can be enabled.

sceGxmPopUserMarker

Inserts a user pop marker into the captured render data.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmPopUserMarker(
    SceGxmContext *context
);
```

Arguments

[in,out] `context` A pointer to the rendering context.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to the `context` pointer being `NULL`.
SCE_GXM_ERROR_NOT_WITHIN_SCENE	The operation failed since the call is not between calls to `sceGxmBeginScene()` and `sceGxmEndScene()`. This error will only be returned from an immediate context.
SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST	The operation failed since the call is not between calls to `sceGxmBeginCommandList()` and `sceGxmEndCommandList()`. This error will only be returned from a deferred context.
SCE_GXM_ERROR_RESERVE_FAILED	The operation failed due to the VDM buffer callback function failing to provide sufficient memory for the user marker. This error will only be returned from a deferred context.

Description

Inserts a user pop marker into the captured render data.

Notes

User marker operations only take place when the Razor GPU Capture module is currently loaded. When using an immediate context, operations are also skipped if a Razor GPU Capture is not being written.

sceGxmPrecomputedDrawInit

Initializes a precomputed draw command.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedDrawInit(
    SceGxmPrecomputedDraw *precomputedDraw,
    const SceGxmVertexProgram *vertexProgram,
    void *memBlock
);
```

Arguments

[out] *precomputedDraw*	A pointer to the precomputed state.
[in] *vertexProgram*	A pointer to the vertex program. This must persist for the lifetime of the `SceGxmPrecomputedDraw` structure.
[out] *memBlock*	A pointer to a block of memory to use for precomputed data.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because one or more of the pointers supplied was `NULL`.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed because one or more of the pointers were not aligned.

Description

Initializes a precomputed draw command. The memory supplied to this function via *memBlock* should be aligned to `SCE_GXM_PRECOMPUTED_ALIGNMENT`. The size of the memory should be at least that returned by `sceGxmGetPrecomputedDrawSize()`. The memory must be mapped for the GPU with at least read access.

This memory is used to store precomputed data structures to draw some geometry using particular vertex streams.

Before this precomputed draw can be used, the function `sceGxmPrecomputedDrawSetAllVertexStreams()` must be called to set the vertex streams. Either the `sceGxmPrecomputedDrawSetParams()` or the `sceGxmPrecomputedDrawSetParamsInstanced()` function must also be called to set the parameters of the draw call. The object can then be drawn by calling `sceGxmDrawPrecomputed()` with a `SceGxmContext`.

sceGxmPrecomputedDrawSetAllVertexStreams

Sets all vertex stream base addresses for the precomputed draw command.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedDrawSetAllVertexStreams(
    SceGxmPrecomputedDraw *precomputedDraw,
    const void *const *streamDataArray
);
```

Arguments

[in,out] *precomputedDraw*	A pointer to precomputed draw command.
[in] *streamDataArray*	An array of stream base addresses.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.

Description

Sets all vertex stream base addresses for the precomputed draw command. It is the caller's responsibility to ensure the precomputed draw command is not currently being used by the GPU during patching.

sceGxmPrecomputedDrawSetParams

Sets the parameters for the precomputed draw command.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedDrawSetParams(
    SceGxmPrecomputedDraw *precomputedDraw,
    SceGxmPrimitiveType primType,
    SceGxmIndexFormat indexType,
    const void *indexData,
    uint32_t indexCount
);
```

Arguments

[in,out] *precomputedDraw*	A pointer to the precomputed draw command.
[in] *primType*	The type of the primitive.
[in] *indexType*	The type of the index data in memory.
[in] *indexData*	A pointer to the index data.
[in] *indexCount*	The number of indices.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because one or more of the pointers supplied was `NULL`.

Description

Sets the parameters for the precomputed draw command. It is the caller's responsibility to ensure this object is not currently being used by the GPU.

sceGxmPrecomputedDrawSetParamsInstanced

Sets the parameters for the precomputed instanced draw command.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedDrawSetParamsInstanced(
    SceGxmPrecomputedDraw *precomputedDraw,
    SceGxmPrimitiveType primType,
    SceGxmIndexFormat indexType,
    const void *indexData,
    uint32_t indexCount,
    uint32_t indexWrap
);
```

Arguments

[in,out] *precomputedDraw*	A pointer to the precomputed draw command.
[in] *primType*	The type of the primitive.
[in] *indexType*	The type of the index data in memory.
[in] *indexData*	A pointer to the index data.
[in] *indexCount*	The number of indices.
[in] *indexWrap*	The number of indices to draw for each instance.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.

Description

Sets the parameters for the precomputed instanced draw command. It is the caller's responsibility to ensure this object is not currently being used by the GPU.

sceGxmPrecomputedDrawSetVertexStream

Sets a vertex stream base addresses for the precomputed draw command.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedDrawSetVertexStream(
    SceGxmPrecomputedDraw *precomputedDraw,
    uint32_t streamIndex,
    const void *streamData
);
```

Arguments

[in,out] *precomputedDraw*	A pointer to precomputed draw command.
[in] *streamIndex*	The index of the vertex stream.
[in] *streamData*	A pointer to the vertex stream data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.
SCE_GXM_ERROR_INVALID_VALUE	The stream index was not valid.

Description

Sets a vertex stream base addresses for the precomputed draw command. It is the caller's responsibility to ensure the precomputed draw command is not currently being used by the GPU during patching.

sceGxmPrecomputedFragmentStateGetDefault UniformBuffer

Gets the default uniform buffer for the precomputed fragment state.

Definition

```
#include <gxm/precomputation.h>
void *sceGxmPrecomputedFragmentStateGetDefaultUniformBuffer(
    const SceGxmPrecomputedFragmentState *precomputedState
);
```

Arguments

[in] *precomputedState*	The pointer to the precomputed state.

Return Values

A pointer to the default uniform buffer.

Description

Gets the default uniform buffer for the precomputed fragment state.

sceGxmPrecomputedFragmentStateInit

Initializes precomputed fragment state using the given memory.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedFragmentStateInit(
    SceGxmPrecomputedFragmentState *precomputedState,
    const SceGxmFragmentProgram *fragmentProgram,
    void *memBlock
);
```

Arguments

[out] *precomputedState*	A pointer to the precomputed state.
[in] *fragmentProgram*	A pointer to the fragment program. This must persist for the lifetime of the SceGxmPrecomputedFragmentState structure.
[out] *memBlock*	A pointer to a block of memory to use for precomputed data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.
SCE_GXM_ERROR_INVALID_ALIGNMENT	The operation failed because one or more of the pointers were not aligned.
SCE_GXM_ERROR_UNSUPPORTED	The operation failed because the fragment program was a mask update fragment program, which is not supported.

Description

Initializes precomputed fragment state using the given memory. The memory supplied to this function via *memBlock* should be aligned to SCE_GXM_PRECOMPUTED_ALIGNMENT. The size of the memory should be at least that returned by sceGxmGetPrecomputedFragmentStateSize(). The memory must be mapped for the GPU with at least read access.

This memory is used to store precomputed data structures to upload fragment program uniforms and texture state for future draw calls. The memory does not contain the default uniform buffer. If the fragment program requires a default uniform buffer, then one should be allocated separately and assigned using sceGxmPrecomputedFragmentStateSetDefaultUniformBuffer().

sceGxmPrecomputedFragmentStateSetAllTextures

Sets all the textures for the precomputed fragment state.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedFragmentStateSetAllTextures(
    SceGxmPrecomputedFragmentState *precomputedState,
    const SceGxmTexture *textureArray
);
```

Arguments

[in,out] *precomputedState*	A pointer to precomputed fragment state.
[in] *textureArray*	A pointer to the texture array. The structures are copied during this function and therefore do not need to persist after the call.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because one or more of the pointers supplied was `NULL`.
`SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_PRECISION`	The operation failed because one of the textures does not support the precision of a query format used in shader code. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_COMPONENT_COUNT`	The operation failed because one of the textures is using a swizzle that produces an incorrect number of components for the query formats used in the shader code. This error is only returned when running the debug version of libgxm.

Value	Description
`SCE_GXM_ERROR_INVALID_SAMPLER_FILTER_MODE`	The operation failed because one of the textures has a filtering mode that is incompatible with the query used in the shader code. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_TEXTURE`	The operation failed because a texture was invalid. This can occur if the format or dimensionality of the texture is not compatible with its usage in the shader code, or the texture control words themselves are malformed. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_TEXTURE_DATA_POINTER`	The operation failed because a texture has a `NULL` data pointer. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_TEXTURE_PALETTE_POINTER`	The operation failed because a palettized texture has a `NULL` palette pointer. This error is only returned when running the debug version of libgxm.

Description

Sets all the textures for the precomputed fragment state. The textures will be accessed within this array using their sampler resource index (i.e. the `TEXUNIT`*n* binding in the shader source code). This implementation is more efficient than setting every texture individually.

It is the caller's responsibility to ensure this precomputed state is not currently being used by the GPU while it is being patched.

sceGxmPrecomputedFragmentStateSetAllUniformBuffers

Sets all the uniform buffers for the precomputed fragment state.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedFragmentStateSetAllUniformBuffers(
    SceGxmPrecomputedFragmentState *precomputedState,
    const void *const *bufferDataArray
);
```

Arguments

[in,out] *precomputedState*	A pointer to the precomputed state.
[in] *bufferDataArray*	An array of pointers to uniform buffer data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.

Description

Sets all the uniform buffers for the precomputed fragment state. The buffer addresses will be accessed within this array using their buffer index (i.e. the BUFFER*n* binding in the shader source code). This implementation is more efficient than setting every uniform buffer individually.

Note that it is the caller's responsibility to ensure this precomputed state is not currently being used by the GPU while it is being patched.

sceGxmPrecomputedFragmentStateSetDefault UniformBuffer

Sets the default uniform buffer for the precomputed fragment state.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedFragmentStateSetDefaultUniformBuffer(
    SceGxmPrecomputedFragmentState *precomputedState,
    const void *defaultBuffer
);
```

Arguments

[in,out] *precomputedState*	A pointer to the precomputed state.
[in] *defaultBuffer*	A pointer to the default uniform buffer data.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because one or more of the pointers supplied was `NULL`.

Description

Sets the default uniform buffer for the precomputed fragment state.

sceGxmPrecomputedFragmentStateSetTexture

Sets a single texture for the precomputed fragment state.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedFragmentStateSetTexture(
    SceGxmPrecomputedFragmentState *precomputedState,
    uint32_t textureIndex,
    const SceGxmTexture *texture
);
```

Arguments

[in,out] *precomputedState*	A pointer to the precomputed state.
[in] *textureIndex*	The `TEXUNIT` index to set the texture as.
[in] *texture*	A pointer to the texture. The structure is copied during this function and therefore does not need to persist after the call.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because one or more of the pointers supplied was `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The texture index was not valid.
`SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_PRECISION`	The operation failed because the texture does not support the precision of a query format used in shader code. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_COMPONENT_COUNT`	The operation failed because the texture is using a swizzle that produces an incorrect number of components for the query formats used in the shader code. This error is only returned when running the debug version of libgxm.

Value	Description
`SCE_GXM_ERROR_INVALID_SAMPLER_FILTER_MODE`	The operation failed because one of the textures has a filtering mode that is incompatible with the query used in the shader code. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_TEXTURE_DATA_POINTER`	The operation failed because the texture has a `NULL` data pointer. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_TEXTURE`	The operation failed because a texture was invalid. This can occur if the format or dimensionality of the texture is not compatible with its usage in the shader code, or the texture control words themselves are malformed. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_TEXTURE_PALETTE_POINTER`	The operation failed because the palettized texture has a `NULL` palette pointer. This error is only returned when running the debug version of libgxm.

Description

Sets a single texture for the precomputed fragment state. This function is convenient when changing a single texture. However, calling this function multiple times to set multiple textures is not as efficient as calling `sceGxmPrecomputedFragmentStateSetAllTextures()` once.

Note that it is the caller's responsibility to ensure this precomputed state is not currently being used by the GPU while it is being patched.

sceGxmPrecomputedFragmentStateSetUniformBuffer

Sets a single uniform buffer for the precomputed fragment state.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedFragmentStateSetUniformBuffer(
    SceGxmPrecomputedFragmentState *precomputedState,
    uint32_t bufferIndex,
    const void *bufferData
);
```

Arguments

[in,out] *precomputedState*	A pointer to the precomputed state.
[in] *bufferIndex*	The buffer index to set the base address for.
[in] *bufferData*	A pointer to the uniform buffer data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.
SCE_GXM_ERROR_INVALID_VALUE	The buffer index was not valid.

Description

Sets a single uniform buffer for the precomputed fragment state. This function is convenient when changing a uniform buffer. However, calling this function multiple times to set multiple uniform buffers is not as efficient as calling `sceGxmPrecomputedFragmentStateSetAllUniformBuffers()` once.

Note that it is the caller's responsibility to ensure this precomputed state is not currently being used by the GPU while it is being patched.

sceGxmPrecomputedVertexStateGetDefaultUniformBuffer

Gets the default uniform buffer for the precomputed vertex state.

Definition

```
#include <gxm/precomputation.h>
void *sceGxmPrecomputedVertexStateGetDefaultUniformBuffer(
    const SceGxmPrecomputedVertexState *precomputedState
);
```

Arguments

[in] *precomputedState*	A pointer to the precomputed state.

Return Values

A pointer to the default uniform buffer.

Description

Gets the default uniform buffer for the precomputed vertex state.

sceGxmPrecomputedVertexStateInit

Initializes precomputed vertex state using the given memory.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedVertexStateInit(
    SceGxmPrecomputedVertexState *precomputedState,
    const SceGxmVertexProgram *vertexProgram,
    void *memBlock
);
```

Arguments

[out] *precomputedState*	A pointer to the precomputed state.
[in] *vertexProgram*	A pointer to the vertex program. This must persist for the lifetime of the SceGxmPrecomputedVertexState structure.
[out] *memBlock*	A pointer to a block of memory to use for precomputed data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_ALIGNMENT	The operation failed because one or more of the pointers were not aligned.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.

Description

Initializes precomputed vertex state using the given memory. The memory supplied to this function via *memBlock* should be aligned to SCE_GXM_PRECOMPUTED_ALIGNMENT. The size of the memory should be at least that returned by sceGxmGetPrecomputedVertexStateSize(). The memory must be mapped for the GPU with at least read access.

This memory is used to store precomputed data structures to upload vertex program uniforms and texture state for future draw calls. The memory does not contain the default uniform buffer. If the vertex program requires a default uniform buffer, then one should be allocated separately and assigned using sceGxmPrecomputedVertexStateSetDefaultUniformBuffer().

sceGxmPrecomputedVertexStateSetAllTextures

Sets all the textures for the precomputed vertex state.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedVertexStateSetAllTextures(
    SceGxmPrecomputedVertexState *precomputedState,
    const SceGxmTexture *textureArray
);
```

Arguments

[in,out] *precomputedState*	A pointer to the precomputed state.
[in] *textureArray*	A pointer to the texture array. The structures are copied during this function and therefore do not need to persist after the call.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.
SCE_GXM_ERROR_UNSUPPORTED	The format of one of the textures is not supported as a vertex texture.
SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_PRECISION	The operation failed because one of the textures does not support the precision of a query format used in shader code. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_COMPONENT_COUNT	The operation failed because one of the textures is using a swizzle that produces an incorrect number of components for the query formats used in the shader code. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_SAMPLER_FILTER_MODE	The operation failed because one of the textures has a filtering mode that is incompatible with the query used in the shader code. This error is only returned when running the debug version of libgxm.

Value	Description
`SCE_GXM_ERROR_INVALID_TEXTURE`	The operation failed because a texture was invalid. This can occur if the format or dimensionality of the texture is not compatible with its usage in the shader code. It can also occur if the texture control words themselves are malformed. This error is only returned when running the debug version of libgxm.
`SCE_GXM_ERROR_INVALID_TEXTURE_DATA_POINTER`	The operation failed because a texture has a `NULL` data pointer. This error is only returned when running the debug version of libgxm.

Description

Sets all the textures for the precomputed vertex state. The textures will be accessed within this array using their sampler resource index (i.e. the TEXUNITn binding in the shader source code). This implementation is more efficient than setting every texture individually using `sceGxmPrecomputedVertexStateSetTexture()`.

Note that it is the caller's responsibility to ensure this precomputed state is not currently being used by the GPU while it is being patched.

Notes

Textures whose formats are based on `SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P2`, `SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P3`, `SCE_GXM_TEXTURE_BASE_FORMAT_YUV422`, `SCE_GXM_TEXTURE_BASE_FORMAT_P4` and `SCE_GXM_TEXTURE_BASE_FORMAT_P8` are not supported for use as vertex textures.

sceGxmPrecomputedVertexStateSetAllUniformBuffers

Sets all the uniform buffers for the precomputed vertex state.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedVertexStateSetAllUniformBuffers(
    SceGxmPrecomputedVertexState *precomputedState,
    const void *const *bufferDataArray
);
```

Arguments

[in,out] *precomputedState*	A pointer to the precomputed state.
[in] *bufferDataArray*	An array of pointers to uniform buffer data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.

Description

Sets all the uniform buffers for the precomputed vertex state. The buffer addresses will be accessed within this array using their buffer index (i.e. the BUFFER*n* binding in the shader source code). This implementation is more efficient than setting every uniform buffer individually.

Note that it is the caller's responsibility to ensure this precomputed state is not currently being used by the GPU while it is being patched.

sceGxmPrecomputedVertexStateSetDefaultUniformBuffer

Sets the default uniform buffer for the precomputed vertex state.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedVertexStateSetDefaultUniformBuffer(
    SceGxmPrecomputedVertexState *precomputedState,
    const void *defaultBuffer
);
```

Arguments

[in,out] *precomputedState*	A pointer to the precomputed state.
[in] *defaultBuffer*	A pointer to the default uniform buffer data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.

Description

Sets the default uniform buffer for the precomputed vertex state.

sceGxmPrecomputedVertexStateSetTexture

Sets a single texture for the precomputed vertex state.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedVertexStateSetTexture(
    SceGxmPrecomputedVertexState *precomputedState,
    uint32_t textureIndex,
    const SceGxmTexture *texture
);
```

Arguments

[in,out] *precomputedState*	A pointer to the precomputed state.
[in] *textureIndex*	The TEXUNIT index to set the texture as.
[in] *texture*	A pointer to the texture. The structure is copied during this function and therefore do not need to persist after the call.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.
SCE_GXM_ERROR_INVALID_VALUE	The texture index was not valid.
SCE_GXM_ERROR_UNSUPPORTED	The format of one of the textures is not supported as a vertex texture.
SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_PRECISION	The operation failed because the texture does not support the precision of a query format used in shader code. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_COMPONENT_COUNT	The operation failed because the texture is using a swizzle that produces an incorrect number of components for the query formats used in the shader code. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_SAMPLER_FILTER_MODE	The operation failed because one of the textures has a filtering mode that is incompatible with the query used in the shader code. This error is only returned when running the debug version of libgxm.
SCE_GXM_ERROR_INVALID_TEXTURE	The operation failed because a texture was invalid. This can occur if the format or dimensionality of the texture is not compatible with its usage in the shader code. It can also occur if the texture control words themselves are malformed. This error is only returned when running the debug version of libgxm.

Value	Description
SCE_GXM_ERROR_INVALID_TEXTURE_DATA_POINTER	The operation failed because the texture has a `NULL` data pointer. This error is only returned when running the debug version of libgxm.

Description

Sets a single texture for the precomputed vertex state. This function is convenient when changing a single texture. However, calling this function multiple times to set multiple textures is not as efficient as calling `sceGxmPrecomputedVertexStateSetAllTextures()` once.

Note that it is the caller's responsibility to ensure this precomputed state is not currently being used by the GPU while it is being patched.

Notes

Textures whose formats are based on `SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P2`, `SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P3`, `SCE_GXM_TEXTURE_BASE_FORMAT_YUV422`, `SCE_GXM_TEXTURE_BASE_FORMAT_P4` and `SCE_GXM_TEXTURE_BASE_FORMAT_P8` are not supported for use as vertex textures.

sceGxmPrecomputedVertexStateSetUniformBuffer

Sets a single uniform buffer for the precomputed vertex state.

Definition

```
#include <gxm/precomputation.h>
SceGxmErrorCode sceGxmPrecomputedVertexStateSetUniformBuffer(
    SceGxmPrecomputedVertexState *precomputedState,
    uint32_t bufferIndex,
    const void *bufferData
);
```

Arguments

[in,out] *precomputedState*	A pointer to the precomputed state.
[in] *bufferIndex*	The buffer index to set the base address for.
[in] *bufferData*	A pointer to the uniform buffer data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.
SCE_GXM_ERROR_INVALID_VALUE	The buffer index was not valid.

Description

Sets a single uniform buffer for the precomputed vertex state. This function is convenient when changing a single uniform buffer. However, calling this function multiple times to set multiple uniform buffers is not as efficient as calling sceGxmPrecomputedVertexStateSetAllUniformBuffers() once.

Note that it is the caller's responsibility to ensure this precomputed state is not currently being used by the GPU while it is being patched.

sceGxmPushUserMarker

Inserts a user push marker into the captured render data.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmPushUserMarker(
    SceGxmContext *context,
    const char *tag
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *tag*	A pointer to the marker string. The string does not need to persist after the call.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to the `context` or `tag` pointer being NULL.
SCE_GXM_ERROR_NOT_WITHIN_SCENE	The operation failed since the call is not between calls to sceGxmBeginScene() and sceGxmEndScene(). This error will only be returned from an immediate context.
SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST	The operation failed since the call is not between calls to sceGxmBeginCommandList() and sceGxmEndCommandList(). This error will only be returned from a deferred context.
SCE_GXM_ERROR_RESERVE_FAILED	The operation failed due to the VDM buffer callback function failing to provide sufficient memory for the user marker. This error will only be returned from a deferred context.

Description

Inserts a user push marker into the captured render data.

Notes

User marker operations only take place when the Razor GPU Capture module is loaded. When the operation takes place, there are significant memory and performance differences for this function depending on the type of context used.

If this function is used with an immediate context, it will have no effect unless a Razor GPU Capture is being written. If a capture is being written, the tag string will be copied into memory managed by the Razor GPU Capture module.

If this function is used with a deferred context, the tag string will be copied into the VDM Stream memory associated with that context. It will be copied even if a Razor GPU Capture is not being written.

sceGxmRemoveRazorGpuCaptureBuffer

Unregister a capture buffer from Razor.

Definition

```
#include <gxm/memory.h>
SceGxmErrorCode sceGxmRemoveRazorGpuCaptureBuffer(
    void *base
);
```

Arguments

[in] *base*	The base address of the GPU capture buffer.

Return Values

Value	Description
SCE_OK	The operation was completed successfully.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a pointer was invalid.
SCE_GXM_ERROR_RAZOR	The operation failed because of an error in the Razor layer.

Description

Unregister a capture buffer from Razor. Please see sceGxmAddRazorGpuCaptureBuffer() for details on the matching API to use for adding GPU capture buffers.

sceGxmRenderTargetGetDriverMemBlock

Retrieves the driver memblock UID that was used in the *driverMemBlock* member of the SceGxmRenderTargetParams that created the given render target.

Definition

```
#include <gxm/render_target.h>
SceGxmErrorCode sceGxmRenderTargetGetDriverMemBlock(
    const SceGxmRenderTarget *renderTarget,
    SceUID *driverMemBlock
);
```

Arguments

[in] *renderTarget*	A pointer to a render target.
[out] *driverMemBlock*	A pointer to storage for the memblock UID.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to a `NULL` pointer.
`SCE_GXM_ERROR_DRIVER`	The operation failed due to a driver error.

Description

Retrieves the driver memblock UID that was used in the *driverMemBlock* member of the SceGxmRenderTargetParams that created the given render target.

sceGxmReserveFragmentDefaultUniformBuffer

Allocates a new default uniform buffer for the current fragment program from the fragment data ring buffer.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmReserveFragmentDefaultUniformBuffer(
    SceGxmContext *context,
    void **uniformBuffer
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[out] *uniformBuffer*	A pointer to storage for a uniform buffer pointer.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.
`SCE_GXM_ERROR_NULL_PROGRAM`	The operation failed because no fragment program was set.
`SCE_GXM_ERROR_NOT_WITHIN_SCENE`	The operation failed since the call is not between calls to `sceGxmBeginScene()` and `sceGxmEndScene()`. This error will only be returned from an immediate context.
`SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST`	The operation failed since the call is not between calls to `sceGxmBeginCommandList()` and `sceGxmEndCommandList()`. This error will only be returned from a deferred context.
`SCE_GXM_ERROR_RESERVE_FAILED`	The operation failed due to a buffer callback function failing to provide sufficient memory for the reservation. This error will only be returned from a deferred context. Immediate context reservations always succeed, but they potentially have high latency if scenes are split to recycle memory.

Description

Allocates a new default uniform buffer for the current fragment program from the fragment data ring buffer. The previous fragment default uniform buffer will be recycled for future reservations as the GPU consumes the ring buffers during normal rendering. Reserving a fragment default uniform buffer replaces any previous fragment default uniform buffer set using `sceGxmSetFragmentDefaultUniformBuffer()`.

This function may only be called inside a scene or command list. When using the immediate context, calling this function outside of a scene will result in the `SCE_GXM_ERROR_NOT_WITHIN_SCENE` error code being returned. When using a deferred context, calling this function outside of a command list will result in the `SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST` error code being returned.

This function can only be called if the current fragment program is non-`NULL`. Calling this function when the current fragment program is `NULL` will return the `SCE_GXM_ERROR_NULL_PROGRAM` error code and no uniform buffer will be reserved.

If the current fragment program has a non-zero size default uniform buffer, this will be reserved from the fragment data ring buffer and the base address will be written to the *`uniformBuffer`* parameter. The contents of this buffer can then either be written directly or written by using utility functions such as `sceGxmSetUniformDataF()`. If the current fragment program has a zero size default uniform buffer, then nothing will be reserved and `NULL` will be written to the *`uniformBuffer`* parameter. In both cases, this function will return `SCE_OK`.

It is not necessary to call this function when using precomputed fragment state. In this case the default buffer is set on the `SceGxmPrecomputedFragmentState` object by calling `sceGxmPrecomputedFragmentStateSetDefaultUniformBuffer()`.

sceGxmReserveVertexDefaultUniformBuffer

Allocates a new default uniform buffer for the current vertex program from the vertex data ring buffer.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmReserveVertexDefaultUniformBuffer(
    SceGxmContext *context,
    void **uniformBuffer
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[out] *uniformBuffer*	A pointer to storage for a uniform buffer pointer.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a parameter was unexpectedly NULL.
SCE_GXM_ERROR_NULL_PROGRAM	The operation failed because no vertex program was set.
SCE_GXM_ERROR_NOT_WITHIN_SCENE	The operation failed since the call is not between calls to sceGxmBeginScene() and sceGxmEndScene(). This error will only be returned from an immediate context.
SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST	The operation failed since the call is not between calls to sceGxmBeginCommandList() and sceGxmEndCommandList(). This error will only be returned from a deferred context.
SCE_GXM_ERROR_RESERVE_FAILED	The operation failed due to a buffer callback function failing to provide sufficient memory for the reservation. This error will only be returned from a deferred context. Immediate context reservations always succeed, but they potentially have high latency if scenes are split to recycle memory.

Description

Allocates a new default uniform buffer for the current vertex program from the vertex data ring buffer. The previous vertex default uniform buffer will be recycled for future reservations as the GPU consumes the ring buffers during normal rendering. Reserving a vertex default uniform buffer replaces any previous vertex default uniform buffer set using `sceGxmSetVertexDefaultUniformBuffer()`.

This function may only be called inside a scene or command list. When using the immediate context, calling this function outside of a scene will result in the `SCE_GXM_ERROR_NOT_WITHIN_SCENE` error code being returned. When using a deferred context, calling this function outside of a command list will result in the `SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST` error code being returned.

This function can only be called if the current vertex program is non-`NULL`. Calling this function when the current vertex program is `NULL` will return the `SCE_GXM_ERROR_NULL_PROGRAM` error code and no uniform buffer will be reserved.

If the current vertex program has a non-zero size default uniform buffer, this will be reserved from the vertex data ring buffer and the base address will be written to the *`uniformBuffer`* parameter. The contents of this buffer can then either be written directly or written by using utility functions such as `sceGxmSetUniformDataF()`. If the current vertex program has a zero size default uniform buffer, then nothing will be reserved and `NULL` will be written to the *`uniformBuffer`* parameter. In both cases, this function will return `SCE_OK`.

It is not necessary to call this function when using precomputed vertex state. In this case the default uniform buffer is set on the `SceGxmPrecomputedVertexState` object by calling `sceGxmPrecomputedVertexStateSetDefaultUniformBuffer()`.

sceGxmSetBackDepthBias

Sets values that offset the computed depth value for back-facing primitives when two-sided rendering has been enabled.

Definition

```
#include <gxm/context.h>
void sceGxmSetBackDepthBias(
    SceGxmContext *context,
    int32_t factor,
    int32_t units
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *factor*	A signed slope value in the range [-16, 15].
[in] *units*	A signed bias value in the range [-16, 15].

Return Values

None

Description

Sets values that offset the computed depth value for back-facing primitives when two-sided rendering has been enabled. The `factor` parameter scales the maximum Z slope, with respect to the X or Y of the primitive, while the `units` parameter scales the minimum resolvable depth buffer value. The results are summed to produce a single value that offsets the depth value for a fragment. It is only necessary to call this function when two-sided rendering is enabled. When not using two-sided rendering, then the setting of `sceGxmSetFrontDepthBias()` is used for both sides.

The documentation for `sceGxmSetFrontDepthBias()` describes how the parameters affect the depth value of each sample on the triangle.

See Also

`sceGxmSetTwoSidedEnable`

sceGxmSetBackDepthFunc

Sets the comparison mode to be applied to depth values for back-facing primitives when two-sided rendering has been enabled.

Definition

```
#include <gxm/context.h>
void sceGxmSetBackDepthFunc(
    SceGxmContext *context,
    SceGxmDepthFunc depthFunc
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *depthFunc*	The depth comparison function.

Return Values

None

Description

Sets the comparison mode to be applied to depth values for back-facing primitives when two-sided rendering has been enabled. It is only necessary to call this function when two-sided rendering is enabled. When not using two-sided rendering, then the setting of `sceGxmSetFrontDepthFunc()` is used for both sides.

See Also

`sceGxmSetTwoSidedEnable`

sceGxmSetBackDepthWriteEnable

Enables depth writes for back-facing primitives when two-sided rendering has been enabled.

Definition

```
#include <gxm/context.h>
void sceGxmSetBackDepthWriteEnable(
    SceGxmContext *context,
    SceGxmDepthWriteMode enable
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `enable`	A flag specifying whether to enable or disable depth writes.

Return Values

None

Description

Enables depth writes for back-facing primitives when two-sided rendering has been enabled. If not enabled, the depth buffer is not updated regardless of whether any depth test has passed. It is only necessary to call this function when two-sided rendering is enabled. When not using two-sided rendering, then the setting of `sceGxmSetFrontDepthWriteEnable()` is used for both sides.

See Also

`sceGxmSetTwoSidedEnable`

sceGxmSetBackFragmentProgramEnable

Enables fragment program processing for back-facing primitives when two-sided rendering has been enabled.

Definition

```
#include <gxm/context.h>
void sceGxmSetBackFragmentProgramEnable(
    SceGxmContext *context,
    SceGxmFragmentProgramMode enable
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *enable*	A flag specifying whether to enable or disable fragment program processing.

Return Values

None

Description

Enables fragment program processing for back-facing primitives when two-sided rendering has been enabled. It is only necessary to call this function when two-sided rendering is enabled. When not using two-sided rendering, then the setting of `sceGxmSetFrontFragmentProgramEnable()` is used for both sides.

See Also

`sceGxmSetTwoSidedEnable`, `sceGxmFragmentProgramIsEnabled`

sceGxmSetBackLineFillLastPixelEnable

Enables filling of the last pixel of a line for back-facing primitives when two-sided rendering has been enabled.

Definition

```
#include <gxm/context.h>
void sceGxmSetBackLineFillLastPixelEnable(
    SceGxmContext *context,
    SceGxmLineFillLastPixelMode enable
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `enable`	A flag specifying whether to enable or disable filling of the last pixel.

Return Values

None

Description

Enables filling of the last pixel of a line for back-facing primitives when two-sided rendering has been enabled. If not enabled, the last pixel of a line is not filled. It is only necessary to call this function when two-sided rendering is enabled. When not using two-sided rendering the setting of `sceGxmSetFrontLineFillLastPixelEnable()` is used for both sides.

See Also

`sceGxmSetTwoSidedEnable`

sceGxmSetBackPointLineWidth

Sets the width of back-facing points and lines when two-sided rendering has been enabled.

Definition

```
#include <gxm/context.h>
void sceGxmSetBackPointLineWidth(
    SceGxmContext *context,
    uint32_t width
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `width`	The width of the points and lines (1-16).

Return Values

None

Description

Sets the width of back-facing points and lines when two-sided rendering has been enabled. It is only necessary to call this function when two-sided rendering is enabled. When not using two-sided rendering, then the setting of `sceGxmSetFrontPointLineWidth()` is used for both sides.

This setting only applies to primitives rendered using polygon mode `SCE_GXM_POLYGON_MODE_LINE`, `SCE_GXM_POLYGON_MODE_TRIANGLE_LINE` or `SCE_GXM_POLYGON_MODE_TRIANGLE_POINT`. Point primitives that use one of the `SCE_GXM_POLYGON_MODE_POINT` polygon modes must always use the PSIZE output from the vertex program.

See Also

`sceGxmSetTwoSidedEnable`

sceGxmSetBackPolygonMode

Sets the polygon mode for back-facing primitives when two-sided rendering has been enabled.

Definition

```
#include <gxm/context.h>
void sceGxmSetBackPolygonMode(
    SceGxmContext *context,
    SceGxmPolygonMode mode
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *mode*	The polygon mode.

Return Values

None

Description

Sets the polygon mode for back-facing primitives when two-sided rendering has been enabled. It is only necessary to call this function when two-sided rendering is enabled. When not using two-sided rendering, then the setting of `sceGxmSetFrontPolygonMode()` is used for both sides.

See Also

`sceGxmSetTwoSidedEnable`

sceGxmSetBackStencilFunc

Sets the stencil function and operations for back-facing primitives when two-sided rendering has been enabled.

Definition

```
#include <gxm/context.h>
void sceGxmSetBackStencilFunc(
    SceGxmContext *context,
    SceGxmStencilFunc func,
    SceGxmStencilOp stencilFail,
    SceGxmStencilOp depthFail,
    SceGxmStencilOp depthPass,
    uint8_t compareMask,
    uint8_t writeMask
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `func`	The stencil comparison function.
[in] `stencilFail`	The stencil operation performed if the stencil test fails.
[in] `depthFail`	The stencil operation performed if the depth test fails.
[in] `depthPass`	The stencil operation performed if the depth test passes.
[in] `compareMask`	A mask of bits used when performing stencil buffer comparison. The current stencil value is anded with this value prior to the test being carried out.
[in] `writeMask`	A per-bit mask applied to the stencil value after stencil operations.

Return Values

None

Description

Sets the stencil function and operations for back-facing primitives when two-sided rendering has been enabled. It is only necessary to call this function when two-sided rendering is enabled. When not using two-sided rendering, then the setting of `sceGxmSetFrontStencilFunc()` is used for both sides.

If two-sided rendering is enabled but stencil testing is not required, the stencil comparison function should be set to `SCE_GXM_STENCIL_FUNC_ALWAYS`, and all stencil operations should be set to `SCE_GXM_STENCIL_OP_KEEP` because this specific setting reduces the amount of parameter buffer used for each primitive block.

See Also

`sceGxmSetTwoSidedEnable`

sceGxmSetBackStencilRef

Sets the stencil reference value for back-facing primitives when two-sided rendering has been enabled.

Definition

```
#include <gxm/context.h>
void sceGxmSetBackStencilRef(
    SceGxmContext *context,
    uint8_t sref
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *sref*	The reference value used for stencil testing.

Return Values

None

Description

Sets the stencil reference value for back-facing primitives when two-sided rendering has been enabled. It is only necessary to call this function when two-sided rendering is enabled. When not using two-sided rendering, then the setting of `sceGxmSetFrontStencilRef()` is used for both sides.

See Also

`sceGxmSetTwoSidedEnable`

sceGxmSetBackVisibilityTestEnable

Enables or disables the visibility test for back-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetBackVisibilityTestEnable(
    SceGxmContext *context,
    SceGxmVisibilityTestMode enable
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *enable*	Specifies whether to enable or disable the visibility test.

Return Values

None

Description

Enables or disables the visibility test for back-facing primitives. It is only necessary to call this function when two-sided rendering is enabled. When not using two-sided rendering the setting of `sceGxmSetFrontVisibilityTestEnable()` is used for both sides.

sceGxmSetBackVisibilityTestIndex

Sets the visibility test operation for back-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetBackVisibilityTestIndex(
    SceGxmContext *context,
    uint32_t index
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `index`	The index in the range [0, 16383].

Return Values

None

Description

Sets the visibility test operation for back-facing primitives. The visibility test index is used as an offset within an array of 32-bit visibility test results that are written by each GPU core.

It is only necessary to call this function when two-sided rendering is enabled. When not using two-sided rendering the setting of `sceGxmSetFrontVisibilityTestIndex()` is used for both sides.

sceGxmSetBackVisibilityTestOp

Sets the visibility test operation for back-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetBackVisibilityTestOp(
    SceGxmContext *context,
    SceGxmVisibilityTestOp op
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *op*	The operation to perform for visible pixels.

Return Values

None

Description

Sets the visibility test operation for back-facing primitives. When visibility testing is enabled, this operation is performed for each visible pixel.

It is only necessary to call this function when two-sided rendering is enabled. When not using two-sided rendering, then the setting of `sceGxmSetFrontVisibilityTestOp()` is used for both sides.

sceGxmSetCullMode

Sets the culling mode for primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetCullMode(
    SceGxmContext *context,
    SceGxmCullMode mode
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *mode*	The cull mode.

Return Values

None

Description

Sets the culling mode for primitives.

sceGxmSetDefaultRegionClipAndViewport

A helper function that internally calls `sceGxmSetRegionClip()` and `sceGxmSetViewport()` to cover a rectangular region from `(0,0)` to `(`*xMax*`,`*yMax*`)`.

Definition

```
#include <gxm/context.h>
void sceGxmSetDefaultRegionClipAndViewport(
    SceGxmContext *context,
    uint32_t xMax,
    uint32_t yMax
);
```

Arguments

[in,out] *context*	A pointer to a rendering context.
[in] *xMax*	The inclusive maximum x value of the rectangle in pixels.
[in] *yMax*	The inclusive maximum y value of the rectangle in pixels.

Return Values

None

Description

A helper function that internally calls `sceGxmSetRegionClip()` and `sceGxmSetViewport()` to cover a rectangular region from `(0,0)` to `(`*xMax*`,`*yMax*`)`. This function is provided for convenience, and it allows deferred contexts to set the region clip and viewport to the same values that would be set up by `sceGxmBeginScene()` on the immediate context. It is equivalent to calling:

```
uint32_t xMin = 0, yMin = 0;
sceGxmSetRegionClip(context, SCE_GXM_REGION_CLIP_OUTSIDE, xMin, yMin, xMax,
    yMax);
    sceGxmSetViewport(
    context,
    0.5f*(float)(1 + xMax + xMin),
    0.5f*(float)(1 + xMax - xMin),
    0.5f*(float)(1 + yMax + yMin),
    -0.5f*(float)(1 + yMax - yMin),
    0.5f,
    0.5f);
```

This function is called automatically for immediate contexts as part of the implementation of `sceGxmBeginScene()`.

sceGxmSetDeferredContextFragmentBuffer

Sets a new fragment buffer for a deferred context.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmSetDeferredContextFragmentBuffer(
    SceGxmContext *deferredContext,
    void *mem,
    uint32_t size
);
```

Arguments

[in,out] *deferredContext*	A pointer to a deferred context.
[in] *mem*	The base address of the buffer. This should be aligned to 4 bytes. Set to `NULL` if memory is to be allocated as required via a callback.
[in] *size*	The size of the memory. This should be aligned to 4 bytes. Set to 0 if memory is to be allocated as required via a callback.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because the pointer to the deferred context was `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because the *deferredContext* parameter does not point to a deferred context, or there is inconsistency in the values supplied to either *mem* or *size*.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed because an invalid address was supplied, or there was a size alignment issue.
`SCE_GXM_ERROR_WITHIN_COMMAND_LIST`	The operation failed because the deferred context is already within a command list.

Description

Sets a new fragment buffer for a deferred context. This function is only supported on a deferred context, and it will return the `SCE_GXM_ERROR_INVALID_VALUE` error code if called using the immediate context.

This function is only supported when the deferred context is not currently building a command list. Calling this function within a command list will result in the `SCE_GXM_ERROR_WITHIN_COMMAND_LIST` error code being returned.

Setting a `NULL` base address with a buffer size of zero is supported. If these settings are used, this will, while the next command list is being constructed, result in the callback function being called the first time memory is required for this buffer. If the *size* parameter is non-zero, it must be a minimum of `SCE_GXM_MINIMUM_DEFERRED_CONTEXT_BUFFER_SIZE` bytes.

sceGxmSetDeferredContextVdmBuffer

Sets a new VDM buffer for a deferred context.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmSetDeferredContextVdmBuffer(
    SceGxmContext *deferredContext,
    void *mem,
    uint32_t size
);
```

Arguments

[in,out] *deferredContext*	A pointer to a deferred context.
[in] *mem*	The base address of the buffer. This should be aligned to 4 bytes. Set to NULL if memory is to be allocated as required via a callback.
[in] *size*	The size of the buffer. This should be aligned to 4 bytes. Set to 0 if memory is to be allocated as required via a callback.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because the pointer to the deferred context was NULL.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because the *deferredContext* parameter does not point to a deferred context, or there is inconsistency in the values supplied to either *mem* or *size*.
SCE_GXM_ERROR_INVALID_ALIGNMENT	The operation failed because an invalid address was supplied, or there was a size alignment issue.
SCE_GXM_ERROR_WITHIN_COMMAND_LIST	The operation failed because the deferred context is already within a command list.

Description

Sets a new VDM buffer for a deferred context. This function is only supported on a deferred context, and it will return the SCE_GXM_ERROR_INVALID_VALUE error code if called using the immediate context.

This function is only supported when the deferred context is not currently building a command list. Calling this function within a command list will result in the SCE_GXM_ERROR_WITHIN_COMMAND_LIST error code being returned.

Setting a NULL base address with a buffer size of zero is supported. If these settings are used, this will, while the next command list is being constructed, result in the callback function being called the first time memory is required for this buffer. If the *size* parameter is non-zero, it must be a minimum of SCE_GXM_MINIMUM_DEFERRED_CONTEXT_BUFFER_SIZE bytes.

sceGxmSetDeferredContextVertexBuffer

Sets a new vertex buffer for a deferred context.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmSetDeferredContextVertexBuffer(
    SceGxmContext *deferredContext,
    void *mem,
    uint32_t size
);
```

Arguments

[in,out] *deferredContext*	A pointer to a deferred context.
[in] *mem*	The base address of the buffer. This should be aligned to 4 bytes. Set to `NULL` if memory is to be allocated as required via a callback.
[in] *size*	The size of the memory. This should be aligned to 4 bytes. Set to 0 if memory is to be allocated as required via a callback.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because the pointer to the deferred context was `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because the *deferredContext* parameter does not point to a deferred context, or there is inconsistency in the values supplied to either *mem* or *size*.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed because an invalid address was supplied, or there was a size alignment issue.
`SCE_GXM_ERROR_WITHIN_COMMAND_LIST`	The operation failed because the deferred context is already within a command list.

Description

Sets a new vertex buffer for a deferred context. This function is only supported on a deferred context, and it will return the `SCE_GXM_ERROR_INVALID_VALUE` error code if called using the immediate context.

This function is only supported when the deferred context is not currently building a command list. Calling this function within a command list will result in the `SCE_GXM_ERROR_WITHIN_COMMAND_LIST` error code being returned.

Setting a `NULL` base address with a buffer size of zero is supported. If these settings are used, this will, while the next command list is being constructed, result in the callback function being called the first time memory is required for this buffer. If the *size* parameter is non-zero, it must be a minimum of `SCE_GXM_MINIMUM_DEFERRED_CONTEXT_BUFFER_SIZE` bytes.

sceGxmSetFragmentDefaultUniformBuffer

Sets a new default uniform buffer for future draw calls.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmSetFragmentDefaultUniformBuffer(
    SceGxmContext *context,
    const void *bufferData
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *bufferData*	A pointer to the uniform buffer data. The GPU data pointed to must persist until fragment processing for the current scene has completed.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.

Description

Sets a new default uniform buffer for future draw calls. This function may be called at any time and replaces any previous reservation made using `sceGxmReserveFragmentDefaultUniformBuffer()`. The fragment default uniform buffer persists until either a new buffer is set, or a new buffer is reserved using `sceGxmReserveFragmentDefaultUniformBuffer()`.

It is not necessary to call this function when using precomputed fragment state. In this case the default uniform buffer is set on the `SceGxmPrecomputedFragmentState` object by calling `sceGxmPrecomputedFragmentStateSetDefaultUniformBuffer()`.

sceGxmSetFragmentProgram

Sets a fragment program for future draw calls.

Definition

```
#include <gxm/context.h>
void sceGxmSetFragmentProgram(
    SceGxmContext *context,
    const SceGxmFragmentProgram *fragmentProgram
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *fragmentProgram*	A pointer to the fragment program to set or NULL. The program must not be released until another program or NULL is set on the SceGxmContext, and until fragment processing for the current scene has completed.

Return Values

None

Description

Sets a fragment program for future draw calls. This function may be called at any time. The fragment program set will persist indefinitely.

Fragment program uniforms do not have any default values when the program has been set. Unless the caller will be using a precomputed fragment state with this program the default buffer should be reserved using sceGxmReserveFragmentDefaultUniformBuffer() and filled with data before drawing. Once reserved, the default buffer remains valid until a new fragment program is set or the scene has ended.

The fragment program pointed to by *fragmentProgram* must persist in memory after this call until a different program is set by a future call. The context allows a NULL program to be set for the purpose of allowing all fragment programs to be destroyed. Note that the context will return an error if the user attempts a draw call with a NULL fragment program.

It is still necessary to set the fragment program when using a precomputed fragment state. This is because the fragment program also defines state, which must be flushed to the GPU.

sceGxmSetFragmentTexture

Sets a fragment program texture for future draw calls.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmSetFragmentTexture(
    SceGxmContext *context,
    uint32_t textureIndex,
    const SceGxmTexture *texture
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *textureIndex*	The `TEXUNIT` index to set the texture as.
[in] *texture*	A pointer to the texture. The structure is copied during this function, so does not need to persist after the call. The GPU data pointed to by the structure must persist until fragment processing for the current scene has completed.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The texture index was not valid.

Description

Sets a fragment program texture for future draw calls. This function may be called at any time. The fragment texture persists indefinitely.

It is not necessary to call this function when using a precomputed fragment state. In this case the textures can be patched directly on the `SceGxmPrecomputedFragmentState` object by calling `sceGxmPrecomputedFragmentStateSetAllTextures()`.

The *textureIndex* parameter must be between 0 and `(SCE_GXM_MAX_TEXTURE_UNITS - 1)`.

The texture control words pointed to by *texture* are copied by value during this call and do not need to persist in memory afterwards. Note that the texture data must remain valid in memory until the GPU has finished fragment processing for the current scene.

sceGxmSetFragmentUniformBuffer

Sets a fragment uniform buffer base address for future draw calls.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmSetFragmentUniformBuffer(
    SceGxmContext *context,
    uint32_t bufferIndex,
    const void *bufferData
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *bufferIndex*	The buffer index to set the base address for.
[in] *bufferData*	A pointer to the uniform buffer data. The GPU data pointed to must persist until fragment processing for the current scene has completed.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The buffer index was not valid.

Description

Sets a fragment uniform buffer base address for future draw calls. This function may be called at any time. The fragment uniform buffer persists indefinitely.

It is not necessary to call this function when using a precomputed fragment state. In this case uniform buffers can be patched directly on the `SceGxmPrecomputedFragmentState` object by calling `sceGxmPrecomputedFragmentStateSetAllUniformBuffers()`.

The *bufferIndex* parameter must be between 0 and `(SCE_GXM_MAX_UNIFORM_BUFFERS - 1)`.

The *bufferData* parameter should be aligned to 64 bytes if the buffer is being used as a writable uniform buffer. This is due to the behavior of system level cache flush operations.

sceGxmSetFrontDepthBias

Sets values that offset the computed depth value for front-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetFrontDepthBias(
    SceGxmContext *context,
    int32_t factor,
    int32_t units
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `factor`	A signed slope value in the range [-16, 15].
[in] `units`	A signed bias value in the range [-16, 15].

Return Values

None

Description

Sets values that offset the computed depth value for front-facing primitives. The *`factor`* parameter scales the maximum Z slope, with respect to the X or Y of the primitive, while the *`units`* parameter scales the minimum resolvable depth buffer value. The results are summed to produce a single value that offsets the depth value for a fragment. If two-sided rendering has not been enabled then this setting applies to both front and back-facing primitives.

Depth bias is always enabled and applied to the on-chip depth value at F32 precision. The equation used for depth bias is as follows:

```
final_z = raw_z + factor*z_slope + units*z_epsilon;
```

The terms in this equation are defined as follows:

- `raw_z` is the z value of this sample on the triangle
- `z_slope` is the maximum Z slope at this sample, defined by `z_slope = fabsf(dz/dx) + fabsf(dz/dy)`
- `z_epsilon` is the smallest value that would affect this sample, defined by `{ int n; frexp(raw_z, &n); z_epsilon = ldexp(1.0f, n - 23); }`
- `factor` and `units` are the arguments to this function
- `final_z` is the final z value of this sample used for the depth test and write

See Also

`sceGxmSetTwoSidedEnable`

sceGxmSetFrontDepthFunc

Sets the comparison mode to be applied to depth values for front-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetFrontDepthFunc(
    SceGxmContext *context,
    SceGxmDepthFunc depthFunc
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *depthFunc*	The depth comparison function.

Return Values

None

Description

Sets the comparison mode to be applied to depth values for front-facing primitives. If two-sided rendering has not been enabled then this setting applies to both front and back-facing primitives.

See Also

sceGxmSetTwoSidedEnable

sceGxmSetFrontDepthWriteEnable

Enables depth writes for front-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetFrontDepthWriteEnable(
    SceGxmContext *context,
    SceGxmDepthWriteMode enable
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *enable*	A flag specifying whether to enable or disable depth writes.

Return Values

None

Description

Enables depth writes for front-facing primitives. If not enabled, the depth buffer is not updated regardless of whether any depth test has passed. If two-sided rendering has not been enabled then this setting applies to both front and back-facing primitives.

See Also

sceGxmSetTwoSidedEnable

sceGxmSetFrontFragmentProgramEnable

Enables fragment program processing for front-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetFrontFragmentProgramEnable(
    SceGxmContext *context,
    SceGxmFragmentProgramMode enable
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *enable*	A flag specifying whether to enable or disable fragment program processing.

Return Values

None

Description

Enables fragment program processing for front-facing primitives. If two-sided rendering has not been enabled then this setting applies to both front and back-facing primitives.

See Also

sceGxmSetTwoSidedEnable, sceGxmFragmentProgramIsEnabled

sceGxmSetFrontLineFillLastPixelEnable

Enables filling of the last pixel of a line for front-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetFrontLineFillLastPixelEnable(
    SceGxmContext *context,
    SceGxmLineFillLastPixelMode enable
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `enable`	A flag specifying whether to enable or disable filling of the last pixel.

Return Values

None

Description

Enables filling of the last pixel of a line for front-facing primitives. If not enabled, the last pixel of a line is not filled. If two-sided rendering has not been enabled then this setting applies to both front and back-facing primitives.

See Also

`sceGxmSetTwoSidedEnable`

sceGxmSetFrontPointLineWidth

Sets the width of front-facing points and lines in pixels.

Definition

```
#include <gxm/context.h>
void sceGxmSetFrontPointLineWidth(
    SceGxmContext *context,
    uint32_t width
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `width`	The width of the points and lines (1-16).

Return Values

None

Description

Sets the width of front-facing points and lines in pixels. If two-sided rendering has not been enabled then this setting applies to both front and back-facing primitives.

This setting only applies to primitives rendered using polygon mode `SCE_GXM_POLYGON_MODE_LINE`, `SCE_GXM_POLYGON_MODE_TRIANGLE_LINE` or `SCE_GXM_POLYGON_MODE_TRIANGLE_POINT`. Point primitives that use one of the `SCE_GXM_POLYGON_MODE_POINT` polygon modes must always use the PSIZE output from the vertex program.

See Also

`sceGxmSetTwoSidedEnable`

sceGxmSetFrontPolygonMode

Sets the polygon mode for front-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetFrontPolygonMode(
    SceGxmContext *context,
    SceGxmPolygonMode mode
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *mode*	The polygon mode.

Return Values

None

Description

Sets the polygon mode for front-facing primitives. If two-sided rendering has not been enabled then this setting applies to both front and back-facing primitives.

See Also

sceGxmSetTwoSidedEnable

sceGxmSetFrontStencilFunc

Sets the stencil function and operations for front-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetFrontStencilFunc(
    SceGxmContext *context,
    SceGxmStencilFunc func,
    SceGxmStencilOp stencilFail,
    SceGxmStencilOp depthFail,
    SceGxmStencilOp depthPass,
    uint8_t compareMask,
    uint8_t writeMask
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *func*	The stencil comparison function.
[in] *stencilFail*	The stencil operation performed if the stencil test fails.
[in] *depthFail*	The stencil operation performed if the depth test fails.
[in] *depthPass*	The stencil operation performed if the depth test passes.
[in] *compareMask*	A mask of bits used when performing stencil buffer comparison. The current stencil value is anded with this value prior to the test being carried out.
[in] *writeMask*	A bitwise mask applied to the stencil value after stencil operations.

Return Values

None

Description

Sets the stencil function and operations for front-facing primitives. If two-sided rendering has not been enabled then this setting applies to both front and back-facing primitives.

If stencil testing is not required, the stencil comparison function should be set to `SCE_GXM_STENCIL_FUNC_ALWAYS`, and all stencil operations should be set to `SCE_GXM_STENCIL_OP_KEEP` because this specific setting reduces the amount of parameter buffer used for each primitive block.

See Also

`sceGxmSetTwoSidedEnable`

sceGxmSetFrontStencilRef

Sets the stencil reference value for front-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetFrontStencilRef(
    SceGxmContext *context,
    uint8_t sref
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *sref*	The reference value used for stencil testing.

Return Values

None

Description

Sets the stencil reference value for front-facing primitives. If two-sided rendering has not been enabled then this setting applies to both front and back-facing primitives.

See Also

sceGxmSetTwoSidedEnable

sceGxmSetFrontVisibilityTestEnable

Enables or disables the visibility test for front-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetFrontVisibilityTestEnable(
    SceGxmContext *context,
    SceGxmVisibilityTestMode enable
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *enable*	Specifies whether to enable or disable the visibility test.

Return Values

None

Description

Enables or disables the visibility test for front-facing primitives. If two-sided rendering has not been enabled then this setting applies to both front and back-facing primitives.

sceGxmSetFrontVisibilityTestIndex

Sets the visibility test index for front-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetFrontVisibilityTestIndex(
    SceGxmContext *context,
    uint32_t index
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `index`	The index in the range [0, 16383].

Return Values

None

Description

Sets the visibility test index for front-facing primitives. The visibility test index is used as an offset within an array of 32-bit visibility test results that are written by each GPU core.

If two-sided rendering has not been enabled then this setting applies to both front and back-facing primitives.

sceGxmSetFrontVisibilityTestOp

Sets the visibility test operation for front-facing primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetFrontVisibilityTestOp(
    SceGxmContext *context,
    SceGxmVisibilityTestOp op
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *op*	The operation to perform for visible pixels.

Return Values

None

Description

Sets the visibility test operation for front-facing primitives. When visibility testing is enabled, this operation is performed for each visible pixel.

If two-sided rendering has not been enabled then this setting applies to both front and back-facing primitives.

sceGxmSetPrecomputedFragmentState

Sets or unsets a precomputed fragment state for future draw calls.

Definition

```
#include <gxm/context.h>
void sceGxmSetPrecomputedFragmentState(
    SceGxmContext *context,
    const SceGxmPrecomputedFragmentState *precomputedState
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *precomputedState*	A pointer to the precomputed state or NULL. The precomputed state must not be released until another program or NULL is set on the SceGxmContext, and until fragment processing for the current scene has completed.

Return Values

None

Description

Sets or unsets a precomputed fragment state for future draw calls. This function may be called at any time. The precomputed state persists indefinitely.

If *precomputedState* is non-NULL, this SceGxmPrecomputedFragmentState object overrides the fragment default uniform buffer reservation, all fragment uniform buffers and all fragment textures set on the context. In this case those patched into the fragment state are used instead. The precomputed state pointed to by *precomputedState* must persist in memory after this call until a different precomputed state or NULL is set by a future call.

If *precomputedState* is NULL the context reverts back to using the fragment uniform buffers and fragment textures currently set on the context.

sceGxmSetPrecomputedVertexState

Sets or unsets a precomputed vertex state for future draw calls.

Definition

```
#include <gxm/context.h>
void sceGxmSetPrecomputedVertexState(
    SceGxmContext *context,
    const SceGxmPrecomputedVertexState *precomputedState
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *precomputedState*	A pointer to the precomputed state or `NULL`. The precomputed state must not be released until another program or `NULL` is set on the `SceGxmContext`, and until vertex processing for the current scene has completed.

Return Values

None

Description

Sets or unsets a precomputed vertex state for future draw calls. This function may be called at any time. The precomputed state persists indefinitely.

If *precomputedState* is non-`NULL`, this `SceGxmPrecomputedVertexState` object overrides the vertex default uniform buffer reservation, all vertex uniform buffers and all vertex textures set on the context. In this case those patched into the vertex state are used instead. The precomputed state pointed to by *precomputedState* must persist in memory after this call until a different precomputed state or `NULL` is set by a future call.

If *precomputedState* is `NULL` the context reverts back to using the vertex uniform buffers and vertex textures currently set on the context.

sceGxmSetRegionClip

Defines a rectangular area and region clip mode that controls which tiles are active during vertex processing.

Definition

```
#include <gxm/context.h>
void sceGxmSetRegionClip(
    SceGxmContext *context,
    SceGxmRegionClipMode mode,
    uint32_t xMin,
    uint32_t yMin,
    uint32_t xMax,
    uint32_t yMax
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *mode*	The region clip mode.
[in] *xMin*	The left edge (inclusive) of the clip region in pixels.
[in] *yMin*	The top edge (inclusive) of the clip region in pixels.
[in] *xMax*	The right edge (inclusive) of the clip region in pixels.
[in] *yMax*	The bottom edge (inclusive) of the clip region in pixels.

Return Values

None

Description

Defines a rectangular area and region clip mode that controls which tiles are active during vertex processing. Only active tiles write their data to the parameter buffer for fragment shading. Geometry that intersects inactive tiles will not be shaded within those tiles.

Region clip can be changed at any time, but please note that `sceGxmBeginScene()` resets the region clip to the size of the render target or valid region. When using a deferred context, region clip must be set manually, using this function or `sceGxmSetDefaultRegionClipAndViewport()`, before the first draw call of the deferred context.

Notes

The coordinates given here are specified in pixels and are inclusive, so a region starting at the origin should use a region clip of (0, 0, width - 1, height - 1) to clip accurately. Although the clip coordinates are supplied in pixels, they will be internally aligned to `SCE_GXM_TILE_SIZEX` and `SCE_GXM_TILE_SIZEY` within this function.

sceGxmSetTwoSidedEnable

Enables two-sided rendering of primitives.

Definition

```
#include <gxm/context.h>
void sceGxmSetTwoSidedEnable(
    SceGxmContext *context,
    SceGxmTwoSidedMode enable
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `enable`	Enable/disable two sided rendering.

Return Values

None

Description

Enables two-sided rendering of primitives. Primitives are determined to be front facing or back facing based on the winding order of their screen-space coordinates. Primitives with a clockwise winding order are treated as front-facing, while primitives with a counterclockwise winding order are treated as back-facing. When two-sided rendering is enabled, state must be set independently for front and back-facing primitives. When two-sided rendering is not enabled, all primitives use the front-facing state.

sceGxmSetUniformDataF

Writes data into a uniform buffer for a given uniform parameter from floating-point inputs.

Definition

```
#include <gxm/uniforms.h>
SceGxmErrorCode sceGxmSetUniformDataF(
    void *uniformBuffer,
    const SceGxmProgramParameter *parameter,
    uint32_t componentOffset,
    uint32_t componentCount,
    const float *sourceData
);
```

Arguments

[out] `uniformBuffer`	The uniform buffer base address.
[in] `parameter`	A pointer to the program parameter that describes the layout.
[in] `componentOffset`	The destination component offset.
[in] `componentCount`	The number of components to write.
[in] `sourceData`	A pointer to the input data in float format.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The component offset or count were outside of the range of the parameter.

Description

Writes data into a uniform buffer for a given uniform parameter from floating-point inputs. This function expects a contiguous array of floating point input values as the parameter *sourceData*. These values are written to memory according to the type, component count and array size of the parameter. The parameters *componentOffset* and *componentCount* control how many scalar components to write, and at what offset (in components) in the output to start writing.

Use of this function is optional: the layout of uniform data in a uniform buffer is entirely defined by its declaration. Uniform parameters generated by the shader compiler will always:

- Have a component count between 1 and 4.
- Have an array size of 1 or greater.

Each component will be a scalar type from `SceGxmParameterType`. The rules for uniform parameter layout in memory are as follows:

- Floating-point vector array elements must start on a 64-bit boundary.
- All other array element types must start on a 32-bit boundary.

These rules are to ensure that vector instructions can always be used directly with vector data without intermediate copies having to be made, and to ensure that indexing is always efficient.

For example: an array of `float3` elements (i.e. type `SCE_GXM_PARAMETER_TYPE_F32` with component count of 3) will have each array element aligned to 64 bits, effectively adding 4 bytes of padding between each array element. However, an array of `float` elements (i.e. type `SCE_GXM_PARAMETER_TYPE_F32` with component count of 1) is scalar, so only 32-bit alignment is required and the array does not contain any padding. Similarly, an array of `char4` and array of `char` both consumed 4 bytes per element due to 32-bit alignment.

This function is provided for convenience only, since it supports all possible uniform parameter types. Data can be copied to uniform buffers by any means provided the alignment rules above are followed.

sceGxmSetUserMarker

Inserts a user set marker into the captured render data.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmSetUserMarker(
    SceGxmContext *context,
    const char *tag
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `tag`	A pointer to the marker string. The string does not need to persist after the call.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to the `context` or `tag` pointer being `NULL`.
`SCE_GXM_ERROR_NOT_WITHIN_SCENE`	The operation failed since the call is not between calls to `sceGxmBeginScene()` and `sceGxmEndScene()`. This error will only be returned from an immediate context.
`SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST`	The operation failed since the call is not between calls to `sceGxmBeginCommandList()` and `sceGxmEndCommandList()`. This error will only be returned from a deferred context.
`SCE_GXM_ERROR_RESERVE_FAILED`	The operation failed due to the VDM buffer callback function failing to provide sufficient memory for the user marker. This error will only be returned from a deferred context.

Description

Inserts a user set marker into the captured render data.

Notes

User marker operations only take place when the Razor GPU Capture module is currently loaded. When the operation takes place, there are significant memory and performance differences for this function depending on the type of context used.

If this function is used with an immediate context, it will have no effect unless a Razor GPU Capture is being written. If a capture is being written, the tag string will be copied into memory managed by the Razor GPU Capture module.

If this function is used with a deferred context, the tag string will be copied into the VDM Stream memory associated with that context. It will be copied even if a Razor GPU Capture is not being written.

sceGxmSetVertexDefaultUniformBuffer

Sets a new vertex default uniform buffer for future draw calls.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmSetVertexDefaultUniformBuffer(
    SceGxmContext *context,
    const void *bufferData
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *bufferData*	A pointer to the uniform buffer data. The GPU data pointed to must persist until vertex processing for the current scene has completed.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.

Description

Sets a new vertex default uniform buffer for future draw calls. This function may be called at any time and replaces any previous reservation made using `sceGxmReserveVertexDefaultUniformBuffer()`. The vertex default uniform buffer persists until either a new buffer is set, or a new buffer is reserved using `sceGxmReserveVertexDefaultUniformBuffer()`.

It is not necessary to call this function when using precomputed vertex state. In this case the default uniform buffer is set on the `SceGxmPrecomputedVertexState` object by calling `sceGxmPrecomputedVertexStateSetDefaultUniformBuffer()`.

sceGxmSetVertexProgram

Sets a vertex program for future draw calls.

Definition

```
#include <gxm/context.h>
void sceGxmSetVertexProgram(
    SceGxmContext *context,
    const SceGxmVertexProgram *vertexProgram
);
```

Arguments

[in,out] *context*	A pointer to a context.
[in] *vertexProgram*	A pointer to the vertex program to set or `NULL`. The program must not be released until another program or `NULL` is set on the `SceGxmContext`, and until vertex processing for the current scene has completed.

Return Values

None

Description

Sets a vertex program for future draw calls. This function may be called at any time. The vertex program set will persist indefinitely.

Vertex program uniforms do not have any default values when the program has been set. Unless the caller will be using a precomputed vertex state with this program, the default buffer should be reserved using `sceGxmReserveVertexDefaultUniformBuffer()` and filled with data before drawing. Once reserved, the default buffer remains valid until a new vertex program is set or the scene is ended.

The vertex program pointed to by *vertexProgram* must persist in memory after this call until a different program is set by a future call. The context allows a `NULL` program to be set for the purpose of allowing all vertex programs to be destroyed. Note that the context will return an error if the user attempts a draw call with a `NULL` vertex program.

It is still necessary to set the vertex program when using a precomputed vertex state or precomputed draw calls (even if you are using both). This is because the vertex program also defines state, which must be flushed to the GPU.

sceGxmSetVertexStream

Sets a vertex stream address for future draw calls.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmSetVertexStream(
    SceGxmContext *context,
    uint32_t streamIndex,
    const void *streamData
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `streamIndex`	The index of the vertex stream.
[in] `streamData`	A pointer to the vertex stream data. The GPU data pointed to must persist until vertex processing for the current scene has completed.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The stream index was not valid.

Description

Sets a vertex stream address for future draw calls. This function may be called at any time. The stream address persists indefinitely.

It is not necessary to call this function when using precomputed draw calls. In this case stream addresses can be patched directly on the `SceGxmPrecomputedDraw` object by calling `sceGxmPrecomputedDrawSetAllVertexStreams()`.

The `streamIndex` parameter must be between 0 and `(SCE_GXM_MAX_VERTEX_STREAMS - 1)`.

sceGxmSetVertexTexture

Sets a vertex program texture for future draw calls.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmSetVertexTexture(
    SceGxmContext *context,
    uint32_t textureIndex,
    const SceGxmTexture *texture
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `textureIndex`	The `TEXUNIT` index to set the texture as.
[in] `texture`	A pointer to the texture. The structure is copied during this function and therefore does not need to persist after the call. The GPU data pointed to by the structure must persist until vertex processing for the current scene has completed.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The texture index was not valid.
`SCE_GXM_ERROR_UNSUPPORTED`	The format of the texture is not supported as a vertex texture.

Description

Sets a vertex program texture for future draw calls. This function may be called at any time. The vertex texture persists indefinitely.

It is not necessary to call this function when using a precomputed vertex state. In this case textures can be patched directly on the `SceGxmPrecomputedVertexState` object by calling `sceGxmPrecomputedVertexStateSetAllTextures()`.

The *`textureIndex`* parameter must be between 0 and `(SCE_GXM_MAX_TEXTURE_UNITS - 1)`.

The texture control words pointed to by *`texture`* are copied by value during this call and do not need to persist in memory afterwards. Note that the texture data must remain valid in memory until the GPU has finished vertex processing for the current scene.

Notes

Textures whose formats are based on `SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P2`, `SCE_GXM_TEXTURE_BASE_FORMAT_YUV420P3`, `SCE_GXM_TEXTURE_BASE_FORMAT_YUV422`, `SCE_GXM_TEXTURE_BASE_FORMAT_P4` and `SCE_GXM_TEXTURE_BASE_FORMAT_P8` are not supported for use as vertex textures.

sceGxmSetVertexUniformBuffer

Sets a vertex uniform buffer base address for future draw calls.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmSetVertexUniformBuffer(
    SceGxmContext *context,
    uint32_t bufferIndex,
    const void *bufferData
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `bufferIndex`	The buffer index to set the base address for.
[in] `bufferData`	A pointer to the uniform buffer data. The GPU data pointed to must persist until vertex processing for the current scene has completed.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The buffer index was not valid.

Description

Sets a vertex uniform buffer base address for future draw calls. This function may be called at any time. The vertex uniform buffer persists indefinitely.

It is not necessary to call this function when using a precomputed vertex state. In this case uniform buffers can be patched directly on the `SceGxmPrecomputedVertexState` object by calling `sceGxmPrecomputedVertexStateSetAllUniformBuffers()`.

The `bufferIndex` parameter must be between 0 and `(SCE_GXM_MAX_UNIFORM_BUFFERS - 1)`.

The `bufferData` parameter should be aligned to 64 bytes if the buffer is being used as a writable uniform buffer. This is due to the behavior of system level cache flush operations.

sceGxmSetViewport

Sets values that define a viewport transformation, used to transform clipping space coordinates generated by the USSE (cx,cy,cz,cw) into screen space coordinates (sx,sy,sz,sw).

Definition

```
#include <gxm/context.h>
void sceGxmSetViewport(
    SceGxmContext *context,
    float xOffset,
    float xScale,
    float yOffset,
    float yScale,
    float zOffset,
    float zScale
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *xOffset*	The offset applied to X.
[in] *xScale*	The scale applied to X.
[in] *yOffset*	The offset applied to Y.
[in] *yScale*	The scale applied to Y.
[in] *zOffset*	The offset applied to Z.
[in] *zScale*	The scale applied to Z.

Return Values

None

Description

Sets values that define a viewport transformation, used to transform clipping space coordinates generated by the USSE (cx,cy,cz,cw) into screen space coordinates (sx,sy,sz,sw). The viewport values can be changed at any time, but please note that sceGxmBeginScene() resets the viewport values to the size of the render target or valid region. When using a deferred context, the viewport values must be set manually, using this function or sceGxmSetDefaultRegionClipAndViewport(), before the first draw call of the deferred context.

The value is computed as shown in the supplied pseudo-code. The W clamping and buffering states used are those set using sceGxmSetWClampValue(), sceGxmSetWClampEnable(), and sceGxmSetWBufferEnable().

If the viewport transform is disabled by calling sceGxmSetViewportEnable, then this step is bypassed, with the USSE outputs assumed to already be in screen space coordinates.

```
if (WClamp enabled)
       if (cw < WClamp value)
              cw = WClamp value

sx = xOffset + xScale * (cx/cw)
sy = yOffset + yScale * (cy/cw)
if (WBuffer enabled)
       sz = zOffset + zScale/cw
```

```
else
       sz = zOffset + zScale * (cz/cw)
sw = 1.0/cw
```

sceGxmSetViewportEnable

Enables or disables the viewport transform, allowing for data coming out of the USSE to be used without clipping or viewport transformation taking place when disabled.

Definition

```
#include <gxm/context.h>
void sceGxmSetViewportEnable(
    SceGxmContext *context,
    SceGxmViewportMode enable
);
```

Arguments

[in,out] `context`	A pointer to the rendering context.
[in] `enable`	Specifies whether to enable or disable the viewport transform.

Return Values

None

Description

Enables or disables the viewport transform, allowing for data coming out of the USSE to be used without clipping or viewport transformation taking place when disabled. When the viewport transfer is disabled, coordinates passed through for rasterization need to fall within the guard band defined by the GPU. This is a range of -1024 to 7167 for each axis. Use of coordinates outside of this range may result in incorrect rasterization.

See `sceGxmSetViewport()` for details of the viewport transform.

sceGxmSetVisibilityBuffer

Updates the visibility buffers for the next scene.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmSetVisibilityBuffer(
    SceGxmContext *immediateContext,
    void *bufferBase,
    uint32_t stridePerCore
);
```

Arguments

[in,out] *immediateContext*	A pointer to the immediate context.
[in] *bufferBase*	The base address of the visibility buffer. The address must persist until fragment processing for the current scene has completed.
[in] *stridePerCore*	The stride between cores through the visibility buffer.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a parameter was unexpectedly NULL.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because the context is not an immediate context.
SCE_GXM_ERROR_INVALID_ALIGNMENT	The operation failed due to either the base address or stride per core not being aligned to SCE_GXM_VISIBILITY_ALIGNMENT bytes.
SCE_GXM_ERROR_WITHIN_SCENE	The operation failed since the call is between calls to sceGxmBeginScene() and sceGxmEndScene(). This function may not be called within a scene.

Description

Updates the visibility buffers for the next scene. This function cannot be called from within a sceGxmBeginScene()/sceGxmEndScene() pair. The visibility buffers will be used for all scenes started after this function returns.

This function is only supported on the immediate context, and it will return the SCE_GXM_ERROR_INVALID_VALUE error code if called using a deferred context.

The visibility buffer must be mapped with read/write access for the GPU, or page faults will occur. The mapped region must extend from *bufferBase* for SCE_GXM_GPU_CORE_COUNT times *stridePerCore* bytes. In addition, the visibility slot index must be no more than *stridePerCore*/4 to ensure that the per-core buffers do not overlap. Both the base address and stride must be aligned to SCE_GXM_VISIBILITY_ALIGNMENT bytes.

sceGxmSetWarningEnabled

Configures warnings that are output when running the debug version of libgxm.

Definition

```
#include <gxm/init.h>
SceGxmErrorCode sceGxmSetWarningEnabled(
    SceGxmWarning warning,
    bool enable
);
```

Arguments

[in] *warning*	The warning to configure.
[in] *enable*	true if the warning is to be enabled, else false.

Return Values

Value	Description
SCE_OK	The operation completed successfully.
SCE_GXM_ERROR_UNINITIALIZED	The operation failed because libgxm is not initialized.

Description

Configures warnings that are output when running the debug version of libgxm. When running the debug version of libgxm, warnings may be output to TTY indicating the occurrence of events that an application developer would wish to informed of. By default all of the warning types present in the `SceGxmWarning` enumeration are enabled. This function can be called to enable/disable warnings.

Notes

When not running the debug version of libgxm, this function has no effect.

sceGxmSetWBufferEnable

Enables W buffering mode during viewport transformation defined by `sceGxmSetViewport()`.

Definition

```
#include <gxm/context.h>
void sceGxmSetWBufferEnable(
    SceGxmContext *context,
    SceGxmWBufferMode enable
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *enable*	Specifies whether to enable or disable W buffering.

Return Values

None

Description

Enables W buffering mode during viewport transformation defined by `sceGxmSetViewport()`.

sceGxmSetWClampEnable

Enables clamping of the W value passed to viewport transformation to the value set by sceGxmSetWClampValue().

Definition

```
#include <gxm/context.h>
void sceGxmSetWClampEnable(
    SceGxmContext *context,
    SceGxmWClampMode enable
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *enable*	Specifies whether to enable or disable W clamping.

Return Values

None

Description

Enables clamping of the W value passed to viewport transformation to the value set by sceGxmSetWClampValue(). See sceGxmSetViewport() for a description of how this interacts with the viewport transform.

sceGxmSetWClampValue

Sets a value used to clamp the W passed to viewport transformation when `sceGxmSetWClampEnable()` has been called with the *enable* parameter set to true.

Definition

```
#include <gxm/context.h>
void sceGxmSetWClampValue(
    SceGxmContext *context,
    float clampValue
);
```

Arguments

[in,out] *context*	A pointer to the rendering context.
[in] *clampValue*	The value with which to clamp W prior to viewport transformation. See `sceGxmSetViewport()` for a description of how this interacts with the viewport transformation.

Return Values

None

Description

Sets a value used to clamp the W passed to viewport transformation when `sceGxmSetWClampEnable()` has been called with the *enable* parameter set to true.

sceGxmSetYuvProfile

Updates a YUV color profile for the next scene.

Definition

```
#include <gxm/context.h>
SceGxmErrorCode sceGxmSetYuvProfile(
    SceGxmContext *immediateContext,
    uint32_t cscIndex,
    SceGxmYuvProfile profile
);
```

Arguments

[in,out] *immediateContext*	A pointer to the immediate context.
[in] *cscIndex*	The CSC index (0 or 1).
[in] *profile*	The YUV color profile.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a parameter was unexpectedly `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter, or because the context is not an immediate context.
`SCE_GXM_ERROR_WITHIN_SCENE`	The operation failed since the call is between calls to `sceGxmBeginScene()` and `sceGxmEndScene()`. This function may not be called within a scene.

Description

Updates a YUV color profile for the next scene. This function cannot be called from within a `sceGxmBeginScene()`/`sceGxmEndScene()` pair. The YUV color profile will be used for all scenes started after this function returns.

This function is only supported on the immediate context, and it will return the `SCE_GXM_ERROR_INVALID_VALUE` error code if called using a deferred context.

The *cscIndex* parameter defines whether the profile being set will be active for textures that use YUV swizzles referencing CSC0 or CSC1.

The default profile for CSC0 is `SCE_GXM_YUV_PROFILE_BT601_STANDARD`, and the default profile for CSC1 is `SCE_GXM_YUV_PROFILE_BT709_STANDARD`.

sceGxmSyncObjectCreate

Creates a sync object.

Definition

```
#include <gxm/sync_object.h>
SceGxmErrorCode sceGxmSyncObjectCreate(
    SceGxmSyncObject **syncObject
);
```

Arguments

[out] *syncObject*	A pointer to storage for a sync object pointer.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_GXM_ERROR_UNINITIALIZED`	The operation failed as libgxm is not initialized.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed as the sync object pointer was `NULL`.
`SCE_GXM_ERROR_DRIVER`	The operation failed due to a driver error.

Description

Creates a sync object. Currently sync objects are used purely to synchronize rendering with display operations in the display queue.

sceGxmSyncObjectDestroy

Destroys a sync object.

Definition

```
#include <gxm/sync_object.h>
SceGxmErrorCode sceGxmSyncObjectDestroy(
    SceGxmSyncObject *syncObject
);
```

Arguments

[in,out] *syncObject*	A sync object pointer.

Return Values

Value	Description
SCE_OK	The operation was completed successfully.
SCE_GXM_ERROR_UNINITIALIZED	The operation failed as libgxm is not initialized.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed as the sync object pointer was NULL.
SCE_GXM_ERROR_DRIVER	The operation failed due to a driver error.

Description

Destroys a sync object.

sceGxmTerminate

Terminates the libgxm library.

Definition

```
#include <gxm/init.h>
SceGxmErrorCode sceGxmTerminate(void);
```

Arguments

None

Return Values

Value	Description
`SCE_OK`	The operation completed successfully.
`SCE_GXM_ERROR_UNINITIALIZED`	The operation failed because libgxm is not initialized.

Description

Terminates the libgxm library. This function should be called before the process exits, after all other libgxm objects have been destroyed.

sceGxmTextureGetData

Gets a pointer to the data of the given texture.

Definition

```
#include <gxm/texture.h>
void *sceGxmTextureGetData(
    const SceGxmTexture *texture
);
```

Arguments

[in] *texture*	A pointer to the texture.

Return Values

A pointer to the texture data.

Description

Gets a pointer to the data of the given texture.

sceGxmTextureGetFormat

Gets the texture format.

Definition

```
#include <gxm/texture.h>
SceGxmTextureFormat sceGxmTextureGetFormat(
    const SceGxmTexture *texture
);
```

Arguments

[in] `texture`	A pointer to the texture.

Return Values

The format of the texture.

Description

Gets the texture format.

sceGxmTextureGetGammaMode

Gets the texture gamma mode.

Definition

```
#include <gxm/texture.h>
SceGxmTextureGammaMode sceGxmTextureGetGammaMode(
    const SceGxmTexture *texture
);
```

Arguments

[in] `texture`	A pointer to the texture.

Return Values

The gamma mode currently used for the texture.

Description

Gets the texture gamma mode.

sceGxmTextureGetHeight

Gets the height of the given texture.

Definition

```
#include <gxm/texture.h>
uint32_t sceGxmTextureGetHeight(
    const SceGxmTexture *texture
);
```

Arguments

[in] *texture*	A pointer to the texture.

Return Values

The height of the texture.

Description

Gets the height of the given texture.

sceGxmTextureGetLodBias

Gets the lod bias of the given texture.

Definition

```
#include <gxm/texture.h>
uint32_t sceGxmTextureGetLodBias(
    const SceGxmTexture *texture
);
```

Arguments

[in] `texture`	A pointer to the texture.

Return Values

The integer representation of the lod bias value (0..63).

Description

Gets the lod bias of the given texture. See `sceGxmTextureSetLodBias()`.

sceGxmTextureGetLodMin

Gets the minimum lod of the given texture.

Definition

```
#include <gxm/texture.h>
uint32_t sceGxmTextureGetLodMin(
    const SceGxmTexture *texture
);
```

Arguments

[in] *texture*	A pointer to the texture.

Return Values

The minimum lod value.

Description

Gets the minimum lod of the given texture. See `sceGxmTextureSetLodMin()`.

sceGxmTextureGetMagFilter

Gets the filter mode for when the texture is magnified.

Definition

```
#include <gxm/texture.h>
SceGxmTextureFilter sceGxmTextureGetMagFilter(
    const SceGxmTexture *texture
);
```

Arguments

[in] `texture`	A pointer to the texture.

Return Values

The mag filter mode of the texture.

Description

Gets the filter mode for when the texture is magnified.

sceGxmTextureGetMinFilter

Gets the filter mode for when the texture is minified.

Definition

```
#include <gxm/texture.h>
SceGxmTextureFilter sceGxmTextureGetMinFilter(
    const SceGxmTexture *texture
);
```

Arguments

[in] *texture*	A pointer to the texture.

Return Values

The min filter mode of the texture.

Description

Gets the filter mode for when the texture is minified.

sceGxmTextureGetMipFilter

Gets the mip filter mode of the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmTextureMipFilter sceGxmTextureGetMipFilter(
    const SceGxmTexture *texture
);
```

Arguments

[in] *texture*	A pointer to the texture.

Return Values

The mip filter mode of the texture.

Description

Gets the mip filter mode of the given texture.

sceGxmTextureGetMipmapCount

Gets the number of mipmaps in the given texture.

Definition

```
#include <gxm/texture.h>
uint32_t sceGxmTextureGetMipmapCount(
    const SceGxmTexture *texture
);
```

Arguments

[in] *texture*	A pointer to the texture.

Return Values

The mipmap count.

Description

Gets the number of mipmaps in the given texture. See `sceGxmTextureSetMipmapCount()`.

sceGxmTextureGetNormalizeMode

Gets the texture normalize mode.

Definition

```
#include <gxm/texture.h>
SceGxmTextureNormalizeMode sceGxmTextureGetNormalizeMode(
    const SceGxmTexture *texture
);
```

Arguments

[in] `texture`	A pointer to the texture.

Return Values

The normalize mode currently used for the texture.

Description

Gets the texture normalize mode.

sceGxmTextureGetPalette

Gets a pointer to the palette data for the given texture.

Definition

```
#include <gxm/texture.h>
void *sceGxmTextureGetPalette(
    const SceGxmTexture *texture
);
```

Arguments

[in] *texture*	A pointer to the texture.

Return Values

A pointer to the palette data or `NULL`.

Description

Gets a pointer to the palette data for the given texture.

sceGxmTextureGetStride

Gets the stride in bytes of the given texture.

Definition

```
#include <gxm/texture.h>
uint32_t sceGxmTextureGetStride(
    const SceGxmTexture *texture
);
```

Arguments

[in] `texture`	A pointer to the texture.

Return Values

The stride of the texture in bytes.

Description

Gets the stride in bytes of the given texture.

Notes

For types other than `SCE_GXM_TEXTURE_LINEAR_STRIDED`, 0 is returned.

sceGxmTextureGetType

Gets the type of the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmTextureType sceGxmTextureGetType(
    const SceGxmTexture *texture
);
```

Arguments

[in] `texture`	A pointer to the texture.

Return Values

The type of the texture.

Description

Gets the type of the given texture.

sceGxmTextureGetUAddrMode

Gets the U addressing mode of the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmTextureAddrMode sceGxmTextureGetUAddrMode(
    const SceGxmTexture *texture
);
```

Arguments

[in] *texture*	A pointer to the texture.

Return Values

The U addressing mode of the texture.

Description

Gets the U addressing mode of the given texture.

sceGxmTextureGetVAddrMode

Gets the V addressing mode of the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmTextureAddrMode sceGxmTextureGetVAddrMode(
    const SceGxmTexture *texture
);
```

Arguments

[in] *texture*	A pointer to the texture.

Return Values

The V addressing mode of the texture.

Description

Gets the V addressing mode of the given texture.

sceGxmTextureGetWidth

Gets the width of the given texture.

Definition

```
#include <gxm/texture.h>
uint32_t sceGxmTextureGetWidth(
    const SceGxmTexture *texture
);
```

Arguments

[in] *texture*	A pointer to the texture.

Return Values

The width of the texture.

Description

Gets the width of the given texture.

sceGxmTextureInitCube

Initializes the texture control words for cube texture data.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureInitCube(
    SceGxmTexture *texture,
    const void *data,
    SceGxmTextureFormat texFormat,
    uint32_t width,
    uint32_t height,
    uint32_t mipCount
);
```

Arguments

[out] *texture*	A pointer to texture to be initialized.
[in] *data*	A pointer to the texture data.
[in] *texFormat*	The format of the texture.
[in] *width*	The width of the texture (1..4096).
[in] *height*	The height of the texture (1..4096).
[in] *mipCount*	The number of mipmaps in the texture (0..13).

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_ALIGNMENT	The operation failed due to invalid buffer alignment.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed due to an invalid input parameter.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to an invalid input pointer.
SCE_GXM_ERROR_UNSUPPORTED	The operation failed due to an unsupported texture format.

Description

Initializes the texture control words for cube texture data. The `data` parameter must have an alignment that is valid for the format of the texture. Please see the *GPU User's Guide* for each format's alignment requirement. The largest value the alignment can be is defined by `SCE_GXM_TEXTURE_ALIGNMENT`.

If the texture memory layout does not include mipmaps, a value of zero should be specified for *mipCount*; otherwise the memory layout assumes that all mip levels down to 1x1 are present.

Notes

Not supported for YUV texture formats.

sceGxmTextureInitCubeArbitrary

Initializes the texture control words for cube texture data with arbitrary dimensions.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureInitCubeArbitrary(
    SceGxmTexture *texture,
    const void *data,
    SceGxmTextureFormat texFormat,
    uint32_t width,
    uint32_t height,
    uint32_t mipCount
);
```

Arguments

[out] `texture`	A pointer to texture to be initialized.
[in] `data`	A pointer to the texture data.
[in] `texFormat`	The format of the texture.
[in] `width`	The width of the texture (1..4096).
[in] `height`	The height of the texture (1..4096).
[in] `mipCount`	The number of mipmaps in the texture (0..13).

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed due to invalid buffer alignment.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation failed due to an unsupported texture format.

Description

Initializes the texture control words for cube texture data with arbitrary dimensions. The `data` parameter must have an alignment that is valid for the format of the texture. Please see the *GPU User's Guide* for each format's alignment requirement. The largest value the alignment can be is defined by `SCE_GXM_TEXTURE_ALIGNMENT`.

If the texture memory layout does not include mipmaps, a value of zero should be specified for *`mipCount`*; otherwise the memory layout assumes that all mip levels down to 1x1 are present.

Notes

Not supported for YUV texture formats.

sceGxmTextureInitLinear

Initializes the texture control words for linear texture data.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureInitLinear(
    SceGxmTexture *texture,
    const void *data,
    SceGxmTextureFormat texFormat,
    uint32_t width,
    uint32_t height,
    uint32_t mipCount
);
```

Arguments

[out] *texture*	A pointer to texture to be initialized.
[in] *data*	A pointer to the texture data.
[in] *texFormat*	The format of the texture.
[in] *width*	The width of the texture (1..4096).
[in] *height*	The height of the texture (1..4096).
[in] *mipCount*	The number of mipmaps in the texture (0..13).

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_ALIGNMENT	The operation failed due to invalid buffer alignment.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed due to an invalid input parameter.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to an invalid input pointer.
SCE_GXM_ERROR_UNSUPPORTED	The operation failed due to an unsupported texture format.

Description

Initializes the texture control words for linear texture data. The `data` parameter must have an alignment that is valid for the format of the texture. Please see the *GPU User's Guide* for each format's alignment requirement. The largest value the alignment can be is defined by `SCE_GXM_TEXTURE_ALIGNMENT`.

The stride of the *data* is formed by aligning the *width* to the nearest multiple of `SCE_GXM_TEXTURE_IMPLICIT_STRIDE_ALIGNMENT`.

If the texture memory layout does not include mipmaps, a value of zero can be specified for *mipCount*.

Notes

Not supported for block compressed texture formats.

sceGxmTextureInitLinearStrided

Initializes the texture control words for linear strided texture data.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureInitLinearStrided(
    SceGxmTexture *texture,
    const void *data,
    SceGxmTextureFormat texFormat,
    uint32_t width,
    uint32_t height,
    uint32_t byteStride
);
```

Arguments

[out] `texture`	A pointer to the texture to be initialized.
[in] `data`	A pointer to the texture data.
[in] `texFormat`	The format of the texture.
[in] `width`	The width of the texture (1..4096).
[in] `height`	The height of the texture (1..4096).
[in] `byteStride`	The stride of the texture in bytes.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed due to invalid buffer alignment.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation failed due to an unsupported texture format.

Description

Initializes the texture control words for linear strided texture data. The `data` parameter must have an alignment that is valid for the format of the texture. Please see the *GPU User's Guide* for each format's alignment requirement. The largest value the alignment can be is defined by `SCE_GXM_TEXTURE_ALIGNMENT`.

Notes

Not supported for 24-bit, block compressed or planar YUV texture formats.

sceGxmTextureInitSwizzled

Initializes the texture control words for swizzled texture data.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureInitSwizzled(
    SceGxmTexture *texture,
    const void *data,
    SceGxmTextureFormat texFormat,
    uint32_t width,
    uint32_t height,
    uint32_t mipCount
);
```

Arguments

[out] `texture`	A pointer to the texture to be initialized.
[in] `data`	A pointer to the texture data.
[in] `texFormat`	The format of the texture data.
[in] `width`	The width of the texture (1..4096).
[in] `height`	The height of the texture (1..4096).
[in] `mipCount`	The number of mipmaps in the texture (0..13).

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed due to invalid buffer alignment.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation failed due to an unsupported texture format.

Description

Initializes the texture control words for swizzled texture data. The `data` parameter must have an alignment that is valid for the format of the texture. Please see the *GPU User's Guide* for each format's alignment requirement. The largest value the alignment can be is defined by `SCE_GXM_TEXTURE_ALIGNMENT`.

If the texture memory layout does not include mipmaps a value of zero can be specified for `mipCount`.

Notes

Not supported for YUV texture formats.

sceGxmTextureInitSwizzledArbitrary

Initializes the texture control words for swizzled texture data with arbitrary dimensions.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureInitSwizzledArbitrary(
    SceGxmTexture *texture,
    const void *data,
    SceGxmTextureFormat texFormat,
    uint32_t width,
    uint32_t height,
    uint32_t mipCount
);
```

Arguments

[out] `texture`	A pointer to the texture to be initialized.
[in] `data`	A pointer to the texture data.
[in] `texFormat`	The format of the texture data.
[in] `width`	The width of the texture (1..4096).
[in] `height`	The height of the texture (1..4096).
[in] `mipCount`	The number of mipmaps in the texture (0..13).

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed due to invalid buffer alignment.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation failed due to an unsupported texture format.

Description

Initializes the texture control words for swizzled texture data with arbitrary dimensions. The `data` parameter must have an alignment that is valid for the format of the texture. Please see the *GPU User's Guide* for each format's alignment requirement. The largest value the alignment can be is defined by `SCE_GXM_TEXTURE_ALIGNMENT`.

If the texture memory layout does not include mipmaps, a value of zero can be specified for `mipCount`.

Notes

Not supported for YUV texture formats.

sceGxmTextureInitTiled

Initializes the texture control words for tiled texture data.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureInitTiled(
    SceGxmTexture *texture,
    const void *data,
    SceGxmTextureFormat texFormat,
    uint32_t width,
    uint32_t height,
    uint32_t mipCount
);
```

Arguments

[out] `texture`	A pointer to the texture to be initialized.
[in] `data`	A pointer to the texture data.
[in] `texFormat`	The format of the texture.
[in] `width`	The width of the texture (32..4096).
[in] `height`	The height of the texture (32..4096).
[in] `mipCount`	The number of mipmaps in the texture (0..13).

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed due to invalid buffer alignment.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation failed due to an unsupported texture format.

Description

Initializes the texture control words for tiled texture data. The `data` parameter must have an alignment that is valid for the format of the texture. Please see the *GPU User's Guide* for each format's alignment requirement. The largest value the alignment can be is defined by `SCE_GXM_TEXTURE_ALIGNMENT`.

If the texture memory layout does not include mipmaps a value of zero can be specified for `mipCount`.

Notes

Not supported for block compressed texture formats.

sceGxmTextureSetData

Sets the pointer to the data of the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetData(
    SceGxmTexture *texture,
    const void *data
);
```

Arguments

[in,out] `texture`	A pointer to the texture.
[in] `data`	A pointer to the texture data.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed due to invalid data alignment.

Description

Sets the pointer to the data of the given texture. The data pointer must have an alignment that is valid for the format of the texture. Please see the *GPU User's Guide* for each format's alignment requirement. The largest value the alignment can be is defined by `SCE_GXM_TEXTURE_ALIGNMENT`.

sceGxmTextureSetFormat

Sets the texture format.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetFormat(
    SceGxmTexture *texture,
    SceGxmTextureFormat texFormat
);
```

Arguments

[in,out] `texture`	A pointer to the texture.
[in] `texFormat`	The texture format.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed due to invalid alignment of texture data.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation was not supported.

Description

Sets the texture format.

Notes

The number of bits per pixel for the format needs to match the number of bits per pixel for the format used during initialization.

The specified texture format cannot refer to a YUV format for textures with a type of `SCE_GXM_TEXTURE_CUBE`.

The specified texture format must support gamma for textures with a gamma mode other than `SCE_GXM_TEXTURE_GAMMA_NONE`.

The specified texture format cannot refer to YUV, block compressed or palettized formats for textures using border addressing modes.

The specified texture format cannot refer to block compressed formats for textures with a type of `SCE_GXM_TEXTURE_TILED`, `SCE_GXM_TEXTURE_LINEAR` or `SCE_GXM_TEXTURE_LINEAR_STRIDED`.

sceGxmTextureSetGammaMode

Sets the texture gamma mode.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetGammaMode(
    SceGxmTexture *texture,
    SceGxmTextureGammaMode gammaMode
);
```

Arguments

[in,out] *texture*	A pointer to the texture.
[in] *gammaMode*	The gamma mode to apply on the specified texture.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to an invalid input pointer.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because a gamma mode was specified that is not supported by the texture format.

Description

Sets the texture gamma mode.

sceGxmTextureSetHeight

Sets the height of the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetHeight(
    SceGxmTexture *texture,
    uint32_t height
);
```

Arguments

[in,out] *texture*	A pointer to the texture.
[in] *height*	The texture height (1..4096).

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_INVALID_VALUE`	The specified height was outside the range 1 to 4096.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The specified `SCE_GXM_TEXTURE_SWIZZLED` or `SCE_GXM_TEXTURE_CUBE` texture height was not a power of 2.

Description

Sets the height of the given texture.

Notes

For texture types `SCE_GXM_TEXTURE_SWIZZLED` and `SCE_GXM_TEXTURE_CUBE` the height must be a power of 2.

sceGxmTextureSetLodBias

Sets the value that offsets the computed mip level when reducing the texture.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetLodBias(
    SceGxmTexture *texture,
    uint32_t bias
);
```

Arguments

[in,out] *texture*	A pointer to the texture.
[in] *bias*	The integer representation of the lod bias (0..63).

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation is not supported for `SCE_GXM_TEXTURE_LINEAR_STRIDED` textures.
`SCE_GXM_ERROR_INVALID_VALUE`	The bias was not in the range [0, 63].

Description

Sets the value that offsets the computed mip level when reducing the texture. The final value is computed as (bias-31)/8, which gives a value in the range [-3.875, +4] in 0.125 increments.

Notes

Not supported for texture type `SCE_GXM_TEXTURE_LINEAR_STRIDED`.

sceGxmTextureSetLodMin

Sets the minimum lod for the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetLodMin(
    SceGxmTexture *texture,
    uint32_t lodMin
);
```

Arguments

[in,out] *texture*	A pointer to the texture.
[in] *lodMin*	The minimum lod value to use.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation is not supported for `SCE_GXM_TEXTURE_LINEAR_STRIDED` textures.
`SCE_GXM_ERROR_INVALID_VALUE`	The minimum lod value was not in the range `[0, sceGxmTextureGetMipmapCount()]`.

Description

Sets the minimum lod for the given texture.

Notes

Not supported for texture type `SCE_GXM_TEXTURE_LINEAR_STRIDED`.

sceGxmTextureSetMagFilter

Sets the filter mode for when the texture is magnified.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetMagFilter(
    SceGxmTexture *texture,
    SceGxmTextureFilter magFilter
);
```

Arguments

[in,out] *texture*	A pointer to the texture.
[in] *magFilter*	The mag filter mode.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to an invalid input pointer.

Description

Sets the filter mode for when the texture is magnified.

sceGxmTextureSetMinFilter

Sets the filter mode for when the texture is minified.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetMinFilter(
    SceGxmTexture *texture,
    SceGxmTextureFilter minFilter
);
```

Arguments

[in,out] `texture`	A pointer to the texture.
[in] `minFilter`	The min filter mode.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_UNSUPPORTED`	Operation not supported on `SCE_GXM_TEXTURE_LINEAR_STRIDED` texture.

Description

Sets the filter mode for when the texture is minified.

sceGxmTextureSetMipFilter

Sets the mip filter mode of the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetMipFilter(
    SceGxmTexture *texture,
    SceGxmTextureMipFilter mipFilter
);
```

Arguments

[in,out] *texture*	A pointer to the texture.
[in] *mipFilter*	The mip filter mode.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation is not supported for `SCE_GXM_TEXTURE_LINEAR_STRIDED` texture.

Description

Sets the mip filter mode of the given texture.

sceGxmTextureSetMipmapCount

Sets the number of mipmaps in the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetMipmapCount(
    SceGxmTexture *texture,
    uint32_t mipCount
);
```

Arguments

[in,out] `texture`	A pointer to the texture.
[in] `mipCount`	The number of mipmaps (0..13).

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_UNSUPPORTED`	Not supported for `SCE_GXM_TEXTURE_LINEAR_STRIDED` textures.
`SCE_GXM_ERROR_INVALID_VALUE`	The mip count must be in the range [0, 13].

Description

Sets the number of mipmaps in the given texture. The possible range of values are:

```
 0 No mipmaps 1 Top level only .. 13 Levels, size is 4096x4096 down to 1x1
```

The distinction between “No mipmaps” and “Top level only” is required for cube maps and planar YUV formats only. The former mode indicates that no mipmaps are present, and that the faces or planes are stored back to back without any additional alignment. For all other texture types the two modes are equivalent.

Notes

Not supported for texture type `SCE_GXM_TEXTURE_LINEAR_STRIDED`.

sceGxmTextureSetNormalizeMode

Sets the texture normalize mode.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetNormalizeMode(
    SceGxmTexture *texture,
    SceGxmTextureNormalizeMode normalizeMode
);
```

Arguments

[in,out] *texture*	A pointer to the texture.
[in] *normalizeMode*	The normalization mode.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to an invalid input pointer.

Description

Sets the texture normalize mode. When enabled, the results of integer-to-float conversion for block-compressed and YUV texture formats, and those texture formats that include 8, 16, and 24-bit integer components will be normalized to produce results in the range [0.0, 1.0] for unsigned integer formats and [-1.0, 1.0] for signed integer formats.

This setting has no effect when the texture format is floating point.

Notes

When the texture has a gamma mode other than SCE_GXM_TEXTURE_GAMMA_NONE, or the texture format is based on SCE_GXM_TEXTURE_BASE_FORMAT_U2U10U10U10, results will always be normalized.

sceGxmTextureSetPalette

Sets a pointer to the palette data for the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetPalette(
    SceGxmTexture *texture,
    const void *paletteData
);
```

Arguments

[in,out] *texture*	A pointer to the texture.
[in] *paletteData*	A pointer to palette data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to an invalid input pointer.
SCE_GXM_ERROR_INVALID_ALIGNMENT	The operation failed due to invalid palette alignment.

Description

Sets a pointer to the palette data for the given texture. The palette data must be aligned to at least SCE_GXM_PALETTE_ALIGNMENT bytes.

Notes

For textures with a format of SCE_GXM_TEXTURE_BASE_FORMAT_P4 or SCE_GXM_TEXTURE_BASE_FORMAT_P8, the palette address must be non-NULL. For textures with other formats, the palette address must be NULL.

sceGxmTextureSetStride

Sets the stride in bytes of the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetStride(
    SceGxmTexture *texture,
    uint32_t byteStride
);
```

Arguments

[in,out] *texture*	A pointer to the texture.
[in] *byteStride*	The stride in bytes.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to an invalid input pointer.
SCE_GXM_ERROR_UNSUPPORTED	The texture was not of type SCE_GXM_TEXTURE_LINEAR_STRIDED.
SCE_GXM_ERROR_INVALID_ALIGNMENT	The stride was not correctly aligned.
SCE_GXM_ERROR_INVALID_VALUE	The stride must be in the range [4, 131072].

Description

Sets the stride in bytes of the given texture.

Notes

Only supported for texture type SCE_GXM_TEXTURE_LINEAR_STRIDED.

sceGxmTextureSetUAddrMode

Sets the U addressing mode of the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetUAddrMode(
    SceGxmTexture *texture,
    SceGxmTextureAddrMode addrMode
);
```

Arguments

[in,out] `texture`	A pointer to the texture.
[in] `addrMode`	The U addressing mode.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to an invalid input pointer.
SCE_GXM_ERROR_UNSUPPORTED	The operation is not supported.

Description

Sets the U addressing mode of the given texture.

Notes

Border addressing modes are only supported for texture types of `SCE_GXM_TEXTURE_SWIZZLED` or `SCE_GXM_TEXTURE_SWIZZLED_ARBITRARY`.

Border addressing modes are not supported for palettized, block compressed, or YUV texture formats.

An addressing mode of `SCE_GXM_TEXTURE_ADDR_MIRROR` is only supported for textures of type `SCE_GXM_TEXTURE_SWIZZLED`.

Only addressing mode `SCE_GXM_TEXTURE_ADDR_CLAMP` is supported for textures of type `SCE_GXM_TEXTURE_LINEAR_STRIDED`.

sceGxmTextureSetVAddrMode

Sets the V addressing mode of the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetVAddrMode(
    SceGxmTexture *texture,
    SceGxmTextureAddrMode addrMode
);
```

Arguments

[in,out] *texture*	A pointer to the texture.
[in] *addrMode*	The V addressing mode.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to an invalid input pointer.
SCE_GXM_ERROR_UNSUPPORTED	The operation is not supported.

Description

Sets the V addressing mode of the given texture.

Notes

Border addressing modes are only supported for texture types of SCE_GXM_TEXTURE_SWIZZLED or SCE_GXM_TEXTURE_SWIZZLED_ARBITRARY.

Border addressing modes are not supported for palettized, block compressed, or YUV texture formats.

An addressing mode of SCE_GXM_TEXTURE_ADDR_MIRROR is only supported for textures of type SCE_GXM_TEXTURE_SWIZZLED.

Only addressing mode SCE_GXM_TEXTURE_ADDR_CLAMP is supported for textures of type SCE_GXM_TEXTURE_LINEAR_STRIDED.

sceGxmTextureSetWidth

Sets the width of the given texture.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureSetWidth(
    SceGxmTexture *texture,
    uint32_t width
);
```

Arguments

[in,out] `texture`	A pointer to the texture.
[in] `width`	The texture width (1..4096).

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_INVALID_VALUE`	The specified width was outside the range 1 to 4096.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The specified `SCE_GXM_TEXTURE_SWIZZLED` or `SCE_GXM_TEXTURE_CUBE` texture width was not a power of 2.

Description

Sets the width of the given texture.

Notes

For texture types `SCE_GXM_TEXTURE_SWIZZLED` and `SCE_GXM_TEXTURE_CUBE` the width must be a power of 2.

sceGxmTextureValidate

Checks texture control words are consistent and do not set unexpected bits.

Definition

```
#include <gxm/texture.h>
SceGxmErrorCode sceGxmTextureValidate(
    const SceGxmTexture *texture
);
```

Arguments

[in] *texture*	A pointer to the texture.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation failed because the texture configuration is not supported.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because a control word field has an invalid value.
`SCE_GXM_ERROR_INVALID_TEXTURE_DATA_POINTER`	The operation failed because the texture has an invalid data pointer.
`SCE_GXM_ERROR_INVALID_TEXTURE_PALETTE_POINTER`	The operation failed because the texture has an invalid palette pointer.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed because the texture data has an invalid alignment.

Description

Checks texture control words are consistent and do not set unexpected bits.

Notes

When using macros to create texture control words, care must be taken to only set fields which are valid for the type of texture. Due to the fact that some fields overlap, this function cannot check against the following (invalid) cases:

- Setting `T0_DADJUST` on a `LINEAR_STRIDED` texture.
- Setting `T0_MIPMAPCLAMP` on a `LINEAR_STRIDED` texture.
- Setting `T0_MINFILTER` on a `LINEAR_STRIDED` texture.
- Setting `T0_MIPFILTER` on a `LINEAR_STRIDED` texture.
- Setting `T0_STRIDELO`/`T0_STRIDEHI`/`T0_STRIDEEX` on a texture that is not LINEAR_STRIDED.
- Setting `T1_USIZE`/`T1_VSIZE` on a texture that is not SWIZZLED or CUBE.
- Setting `T1_WIDTH`/`T1_HEIGHT` on a texture that is SWIZZLED or CUBE.

sceGxmTransferCopy

Performs a copy of data using transfer hardware, optionally converting between RGB formats, between YUV formats, or between memory layouts.

Definition

```
#include <gxm/transfer.h>
SceGxmErrorCode sceGxmTransferCopy(
    uint32_t width,
    uint32_t height,
    uint32_t colorKeyValue,
    uint32_t colorKeyMask,
    SceGxmTransferColorKeyMode colorKeyMode,
    SceGxmTransferFormat srcFormat,
    SceGxmTransferType srcType,
    const void *srcAddress,
    uint32_t srcX,
    uint32_t srcY,
    int32_t srcStride,
    SceGxmTransferFormat destFormat,
    SceGxmTransferType destType,
    void *destAddress,
    uint32_t destX,
    uint32_t destY,
    int32_t destStride,
    SceGxmSyncObject *syncObject,
    uint32_t syncFlags,
    const SceGxmNotification *notification
);
```

Arguments

[in] *width*	The width of the transfer in pixels (1..1024).
[in] *height*	The height of the transfer in pixels (1..1024).
[in] *colorKeyValue*	The value used for color key comparison.
[in] *colorKeyMask*	The mask applied before color key processing. A value of 1 indicates that the bit should be used in the color key test.
[in] *colorKeyMode*	The mode used to specify whether color keying is enabled, or whether matching colors are passed or rejected.
[in] *srcFormat*	The source format.
[in] *srcType*	The source type.
[in] *srcAddress*	The source address.
[in] *srcX*	The X position associated with the top left of the source area (0..8191).
[in] *srcY*	The Y position associated with the top left of the source area (0..8191).
[in] *srcStride*	The source stride in bytes (-32768..32767).
[in] *destFormat*	The destination format.
[in] *destType*	The destination type.
[in] *destAddress*	The destination address.
[in] *destX*	The X position associated with the top left of the destination area (0..8191).
[in] *destY*	The Y position associated with the top left of the destination area (0..8191).
[in] *destStride*	The destination stride in bytes (-32768..32767).
[in] *syncObject*	An optional sync object to synchronize against transfer processing.
[in] *syncFlags*	The flags that define vertex and fragment synchronization behavior for the transfer.
[in] *notification*	A pointer to a notification object used to identify completion of transfer processing. Set this parameter to `NULL` if this is not required.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_UNSUPPORTED	The operation failed due to an unsupported format.
SCE_GXM_ERROR_INVALID_ALIGNMENT	The operation failed due to invalid alignment.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to an invalid pointer.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed due to an invalid input parameter.
SCE_GXM_ERROR_INVALID_THREAD	The operation failed because the function was called from the display queue thread.
SCE_GXM_ERROR_DRIVER	The operation failed due to a driver error.

Description

Performs a copy of data using transfer hardware, optionally converting between RGB formats, between YUV formats, or between memory layouts. For both RGB and YUV formats, color keying can also be applied to source data.

Notes

The format of `colorKeyValue` and `colorKeyMask` parameters is either U8U8U8U8_ABGR (for RGB formats) or the destination YUV format (for YUV formats). However, if both the `srcFormat` and `destFormat` parameters are `SCE_GXM_TRANSFER_FORMAT_U2U10U10U10_ABGR` then the format is U2U10U10U10_ABGR. If the `srcFormat` parameter refers to an RGB format that includes components that are less than 8 bits in size (`SCE_GXM_TRANSFER_FORMAT_U4U4U4U4_ABGR` for example), the source data is expanded by shifting to the U8U8U8U8_ABGR format. After the shift the data occupies the most significant bits of the 8-bit output. The least significant bits are then formed by replicating the most significant bits of the source. As such the values of the `colorKeyValue` and `colorKeyMask` parameters should take this into account. This is done by either defining `colorKeyValue` to include the replicated bits or by specifying a `colorKeyMask` value that masks off replicated bits.

It is not permitted for both `srcType` and `destType` to be set to `SCE_GXM_TRANSFER_SWIZZLED`. If this form of copy operation is required, it can be performed by setting both `srcType` and `destType` parameters to `SCE_GXM_TRANSFER_LINEAR`.

It is not permitted for `srcType` to be `SCE_GXM_TRANSFER_TILED` if `destType` is `SCE_GXM_TRANSFER_SWIZZLED`.

It is not permitted for `srcType` to be `SCE_GXM_TRANSFER_SWIZZLED` if `destType` is `SCE_GXM_TRANSFER_TILED`.

`width` and `height` must both be a power of two if `srcType` or `destType` are `SCE_GXM_TRANSFER_SWIZZLED`.

`width` and `height` must both be aligned to 32 if `srcType` or `destType` are `SCE_GXM_TRANSFER_TILED`.

`width` must be even if `srcFormat` or `dstFormat` are a YUV format.

`srcStride` must be a multiple of the number of bytes per pixel associated with `srcFormat`.

`srcStride` must match the width in bytes if `srcType` is `SCE_GXM_TRANSFER_TILED` or `SCE_GXM_TRANSFER_SWIZZLED`.

`srcX` and `srcY` must both be 0 if `srcType` is `SCE_GXM_TRANSFER_TILED` or `SCE_GXM_TRANSFER_SWIZZLED`.

`srcX` must be even if `srcFormat` is a YUV format.

`srcX` + `width` must be less than 8192.

`srcY` + `height` must be less than 8192.

`destStride` must be a multiple of the number of bytes per pixel associated with `destFormat`.

`destStride` must match the width in bytes if `destType` is `SCE_GXM_TRANSFER_TILED` or `SCE_GXM_TRANSFER_SWIZZLED`.

`destX` and `destY` must both be 0 if `destType` is `SCE_GXM_TRANSFER_TILED` or `SCE_GXM_TRANSFER_SWIZZLED`.

`destX` must be even if `destFormat` is a YUV format.

`destX` + `width` must be less than 8192.

`destY` + `height` must be less than 8192.

`srcFormat` and `destFormat` must both be of the same fundamental type (RGB, YUV, or RAW).

`colorKeyMode` must be `SCE_GXM_TRANSFER_COLORKEY_NONE` for RAW formats.

`colorKeyMode` must be `SCE_GXM_TRANSFER_COLORKEY_NONE` if `srcType` or `destType` are not `SCE_GXM_TRANSFER_LINEAR`.

`colorKeyMode` must be `SCE_GXM_TRANSFER_COLORKEY_NONE` if either `srcFormat` or `destFormat` are `SCE_GXM_TRANSFER_FORMAT_U2U10U10U10_ABGR` and `srcFormat` is not equal to `destFormat`.

If `srcFormat` is a format of type RAW, then `destFormat` must be the same format.

If `srcFormat` is a format of type RAW, then `srcType` cannot be `SCE_GXM_TRANSFER_TILED`.

If `destFormat` is a format of type RAW, then `destType` cannot be `SCE_GXM_TRANSFER_TILED`.

If `srcFormat` is `SCE_GXM_TRANSFER_FORMAT_RAW64`, then a maximum of 524288 pixels can be copied (1024x512).

If `srcFormat` is `SCE_GXM_TRANSFER_FORMAT_RAW128`, then a maximum of 262144 pixels can be copied (512x512).

sceGxmTransferDownscale

Performs a fixed 50% downscale of source data using a box filter.

Definition

```
#include <gxm/transfer.h>
SceGxmErrorCode sceGxmTransferDownscale(
    SceGxmTransferFormat srcFormat,
    const void *srcAddress,
    uint32_t srcX,
    uint32_t srcY,
    uint32_t srcWidth,
    uint32_t srcHeight,
    int32_t srcStride,
    SceGxmTransferFormat destFormat,
    void *destAddress,
    uint32_t destX,
    uint32_t destY,
    int32_t destStride,
    SceGxmSyncObject *syncObject,
    uint32_t syncFlags,
    const SceGxmNotification *notification
);
```

Arguments

[in] `srcFormat`	The source format.
[in] `srcAddress`	The source address.
[in] `srcX`	The X position associated with the top left of the source area (0..8191).
[in] `srcY`	The Y position associated with the top left of the source area (0..8191).
[in] `srcWidth`	The width of the source area in pixels (2..1024).
[in] `srcHeight`	The height of the source area in pixels (2..1024).
[in] `srcStride`	The source stride in bytes (-32768..32767).
[in] `destFormat`	The destination format.
[in] `destAddress`	The destination address.
[in] `destX`	The X position associated with the top left of the destination area (0..8191).
[in] `destY`	The Y position associated with the top left of the destination area (0..8191)
[in] `destStride`	The destination stride in bytes (-32768..32767).
[in] `syncObject`	An optional sync object to synchronize against transfer processing.
[in] `syncFlags`	The flags defining vertex and fragment synchronization behavior for the transfer.
[in] `notification`	A pointer to a notification object used to identify completion of transfer processing. Set this parameter to `NULL` if this is not required.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation failed due to an unsupported format.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed due to invalid alignment.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid pointer.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_GXM_ERROR_INVALID_THREAD`	The operation failed because the function was called from the display queue thread.
`SCE_GXM_ERROR_DRIVER`	The operation failed due to a driver error.

Description

Performs a fixed 50% downscale of source data using a box filter.

Notes

This function can only be used on linear data.
`srcFormat` and `destFormat` must both be RGB formats.
`srcStride` must be a multiple of the number of bytes per pixel associated with `srcFormat`.
`srcWidth` and `srcHeight` must both be even.
`srcWidth` must be less than 16, or aligned to 16.
`srcX` + `srcWidth` must be less than 8192.
`srcY` + `srcHeight` must be less than 8192.
`destStride` must be a multiple of the number of bytes per pixel associated with `destFormat`.
`destX` + (`srcWidth` / 2) must be less than 8192.
`destY` + (`srcHeight` / 2) must be less than 8192.

sceGxmTransferFill

Performs a fill of RGB or YUV data with a specific value.

Definition

```
#include <gxm/transfer.h>
SceGxmErrorCode sceGxmTransferFill(
    uint32_t fillColor,
    SceGxmTransferFormat destFormat,
    void *destAddress,
    uint32_t destX,
    uint32_t destY,
    uint32_t destWidth,
    uint32_t destHeight,
    int32_t destStride,
    SceGxmSyncObject *syncObject,
    uint32_t syncFlags,
    const SceGxmNotification *notification
);
```

Arguments

[in] *fillColor*	The fill color.
[in] *destFormat*	The destination format.
[in] *destAddress*	The destination address.
[in] *destX*	The X position associated with the top left of the destination area (0..8191).
[in] *destY*	The Y position associated with the top left of the destination area (0..8191).
[in] *destWidth*	The width of the destination in pixels (1..1024).
[in] *destHeight*	The height of the destination in pixels (1..1024).
[in] *destStride*	The destination stride in bytes (-32768..32767).
[in] *syncObject*	An optional sync object to synchronize against transfer processing.
[in] *syncFlags*	The flags that define vertex and fragment synchronization behavior for the transfer.
[in] *notification*	A pointer to a notification object used to identify completion of transfer processing. Set this parameter to `NULL` if this is not required.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_UNSUPPORTED`	The operation failed due to an unsupported format.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed due to invalid alignment.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid pointer.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_GXM_ERROR_INVALID_THREAD`	The operation failed because the function was called from the display queue thread.
`SCE_GXM_ERROR_DRIVER`	The operation failed due to a driver error.

Description

Performs a fill of RGB or YUV data with a specific value.

Notes

This function can only be used on linear data.

The `fillColor` parameter format is either U8U8U8U8_ABGR (for RGB formats) or U8U8U8U8_AYUV (for YUV formats). However, if the `destFormat` parameter is `SCE_GXM_TRANSFER_FORMAT_U2U10U10U10_ABGR`, the format should be U2U10U10U10_ABGR.

`destFormat` must be an RGB or YUV format.

`destWidth` must be even if `destFormat` is a YUV format.

`destStride` must be a multiple of the number of bytes per pixel associated with `destFormat`.

`destX` must be even if `destFormat` is a YUV format.

`destX` + `destWidth` must be less than 8192.

`destY` + `destHeight` must be less than 8192.

sceGxmTransferFinish

Blocks until all transfers have finished.

Definition

```
#include <gxm/transfer.h>
SceGxmErrorCode sceGxmTransferFinish(void);
```

Arguments

None

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_THREAD`	The operation failed because the function was called from the display queue thread.
`SCE_GXM_ERROR_DRIVER`	The operation failed due to a driver error.

Description

Blocks until all transfers have finished.

sceGxmUnmapFragmentUsseMemory

Unmaps memory that was previously mapped as fragment USSE code.

Definition

```
#include <gxm/memory.h>
SceGxmErrorCode sceGxmUnmapFragmentUsseMemory(
    void *base
);
```

Arguments

[in] *base*	The base address of the region to unmap. This must match the base address that was used when mapping the memory using `sceGxmMapFragmentUsseMemory()`.

Return Values

Value	Description
SCE_OK	The operation was completed successfully.
SCE_GXM_ERROR_UNINITIALIZED	The operation failed because the library was not initialized.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because a pointer was invalid.
SCE_GXM_ERROR_DRIVER	The operation failed due to a driver error.

Description

Unmaps memory that was previously mapped as fragment USSE code. It is the responsibility of the caller to ensure that the GPU no longer needs this memory for rendering. This could be accomplished by calling `sceGxmFinish()` before unmapping.

sceGxmUnmapMemory

Unmaps memory, removing it from GPU usage.

Definition

```
#include <gxm/memory.h>
SceGxmErrorCode sceGxmUnmapMemory(
    void *base
);
```

Arguments

[in] *base*	The base address of the region to unmap. This must match the base address that was used when mapping the memory using `sceGxmMapMemory()`.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_GXM_ERROR_UNINITIALIZED`	The operation failed because the library was not initialized.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a pointer was invalid.
`SCE_GXM_ERROR_DRIVER`	The operation failed due to a driver error.

Description

Unmaps memory, removing it from GPU usage. It is the responsibility of the caller to ensure that the GPU no longer needs this memory for rendering. This could be accomplished by calling `sceGxmFinish()` before unmapping.

sceGxmUnmapVertexUsseMemory

Unmaps memory that was previously mapped as vertex USSE code.

Definition

```
#include <gxm/memory.h>
SceGxmErrorCode sceGxmUnmapVertexUsseMemory(
    void *base
);
```

Arguments

[in] *base*	The base address of the region to unmap. This must match the base address that was used when mapping the memory using `sceGxmMapVertexUsseMemory()`.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_GXM_ERROR_UNINITIALIZED`	The operation failed because the library was not initialized.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because a pointer was invalid.
`SCE_GXM_ERROR_DRIVER`	The operation failed due to a driver error.

Description

Unmaps memory that was previously mapped as vertex USSE code. It is the responsibility of the caller to ensure that the GPU no longer needs this memory for rendering. This could be accomplished by calling `sceGxmFinish()` before unmapping.

sceGxmVertexProgramGetProgram

Gets the underlying program for the given vertex program.

Definition

```
#include <gxm/vertex_program.h>
const SceGxmProgram *sceGxmVertexProgramGetProgram(
    const SceGxmVertexProgram *vertexProgram
);
```

Arguments

[in] *vertexProgram*	A pointer to the vertex program.

Return Values

A pointer to the program.

Description

Gets the underlying program for the given vertex program.

sceGxmWaitEvent

Waits for a GPU event to occur.

Definition

```
#include <gxm/init.h>
SceGxmErrorCode sceGxmWaitEvent(void);
```

Arguments

None

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_DRIVER`	An internal driver error occurred.

Description

Waits for a GPU event to occur. This function blocks execution of the calling thread until one of the following occurs:

- The GPU has finished a job for *any* process since the last time `sceGxmWaitEvent()` was called from this thread.
- The GPU has finished an internal firmware job since the last call to `sceGxmWaitEvent()` from this thread.
- An internal GPU driver timeout occurred while waiting.

Any of these conditions will result in the thread waking up and a return value of `SCE_OK`.

This function may be called from any thread.

Callback Functions

SceGxmDeferredContextCallback

A callback function for handling deferred context memory buffers.

Definition

```
#include <gxm/context.h>
typedef void * (*SceGxmDeferredContextCallback)(
    void *userData,
    uint32_t minSize,
    uint32_t *size
);
```

Arguments

userData	The user data pointer from the initialization parameters that were used to create the deferred context.
minSize	The minimum size the memory created by this callback can be.
size	This should receive the actual size of the memory created by the callback.

Return Values

The GPU-mapped memory created by the callback. This should be set to `NULL` if no memory was created.

Description

A callback function for handling deferred context memory buffers. The *userData* parameter provides the user data pointer from the initialization parameters used to create the deferred context.

An implementation of this callback must return GPU-mapped memory of at least *minSize* bytes, and it must write the memory's actual size to the storage pointed to by the *size* parameter. The value of *minSize* will always be at least `SCE_GXM_MINIMUM_DEFERRED_CONTEXT_BUFFER_SIZE` bytes in size. If larger blocks of memory are provided by the application, then overheads due to repeated invocation of the callback can be reduced. This is especially relevant when using the debug version of libgxm as this performs exhaustive validation of the memory address returned from this callback, which can degrade runtime performance.

If no memory is available, then the callback may return `NULL`. If this occurs during `sceGxmBeginCommandList()` then creation of the command list will fail. If this occurs within a command list, then that reservation or draw call will fail, but the command list will remain in a well-defined state. The implementation of `sceGxmEndCommandList()` will always succeed if a command list is in progress on that deferred context. In this way, command lists that are successfully begun can always be ended and later executed on an immediate context, but may contain fewer than expected draw calls if a callback function failed to provide additional memory while the command list was being constructed.

It is not valid to call any function on the deferred context in the implementation of a callback function triggered from that deferred context, as these functions are not designed to be re-entrant.

SceGxmDisplayQueueCallback

A callback function to handle displaying a buffer.

Definition

```
#include <gxm/display_queue.h>
typedef void (*SceGxmDisplayQueueCallback)(
    const void *callbackData
);
```

Arguments

callbackData	A pointer to data that was copied to internal storage during sceGxmDisplayQueueAddEntry().

Return Values

None

Description

A callback function to handle displaying a buffer. This function is called when the GPU has completed rendering. It is responsible for flipping the display buffer and blocking until the flip operation is completed. This means there is the potential for the old display buffer to be overwritten immediately.

Since the libgxm context is single threaded, no libgxm context functions or synchronization functions should be called from this callback function; otherwise undefined behavior could occur. In particular, neither the sceGxmNotificationWait() synchronization function or any function that takes a libgxm context (such as sceGxmBeginScene(), sceGxmDraw() or sceGxmFinish()) should be called.

The expected behavior is to call sceDisplaySetFrameBuf() to enqueue a new display buffer address. A call to sceDisplayWaitSetFrameBuf() should follow this if the flip operation was called with SCE_DISPLAY_UPDATETIMING_NEXTVSYNC. This ensures that future GPU operations on the old front buffer do not start until the new front buffer is being displayed.

Error Codes

Define Summary

Define	Value	Description
`SCE_GXM_ERROR_ALREADY_INITIALIZED`	-2141519871	The call failed because the library is already initialized.
`SCE_GXM_ERROR_BUFFER_OVERRUN`	-2141519826	The call failed because the memory directly after the default uniform buffer reservation was modified between the reserve call and the draw call. This indicates a memory overrun when writing the default uniform buffer contents.
`SCE_GXM_ERROR_DRIVER`	-2141519849	The call failed because of a driver error.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	-2141519867	The call failed because a parameter is incorrectly aligned.
`SCE_GXM_ERROR_INVALID_DEPTH_STENCIL_CONFIGURATION`	-2141519841	The call failed because of an invalid depth/stencil configuration.
`SCE_GXM_ERROR_INVALID_FRAGMENT_MSAA_MODE`	-2141519844	The call failed because of a mismatch between fragment program and render target MSAA usage.
`SCE_GXM_ERROR_INVALID_INDEX_COUNT`	-2141519859	The call failed because an invalid index count was supplied.
`SCE_GXM_ERROR_INVALID_MAPPING`	-2141519834	The call failed because memory was not mapped or was mapped with incorrect attributes.
`SCE_GXM_ERROR_INVALID_OUTPUT_REGISTER_SIZE`	-2141519845	The call failed because of a mismatch between fragment program and color surface output register sizes.
`SCE_GXM_ERROR_INVALID_POINTER`	-2141519868	The call failed because a pointer has an invalid value.

Define	Value	Description
`SCE_GXM_ERROR_INVALID_POLYGON_MODE`	`-2141519858`	The call failed because an invalid polygon mode was supplied for the selected primitive type.
`SCE_GXM_ERROR_INVALID_PRECOMPUTED_DRAW`	`-2141519852`	The call failed because the precomputed draw call was created for a different vertex program than the current vertex program.
`SCE_GXM_ERROR_INVALID_PRECOMPUTED_FRAGMENT_STATE`	`-2141519850`	The call failed because the precomputed fragment state was created for a different fragment program than the current fragment program.
`SCE_GXM_ERROR_INVALID_PRECOMPUTED_VERTEX_STATE`	`-2141519851`	The call failed because the precomputed vertex state was created for a different vertex program than the current vertex program.
`SCE_GXM_ERROR_INVALID_PRIMITIVE_TYPE`	`-2141519835`	The call failed because an invalid primitive type was supplied.

Define	Value	Description
SCE_GXM_ERROR_INVALID_REGION_CLIP_IN_COMMAND_LIST	-2141519829	The call failed because the draw calls within a command list use a region clip configuration that exceeds the scene valid region in the Y direction. This configuration is not correctly handled by the GPU, so it must be avoided when building a command list. When using the region clip on the immediate context, this clamping operation is applied automatically to the Y coordinates to avoid this issue. On a deferred context, it is not possible to do the clamping operation automatically because the valid region that the resulting command list will be used with is unknown.
SCE_GXM_ERROR_INVALID_SAMPLER_FILTER_MODE	-2141519830	The call failed because the filtering mode required for the texture query is not compatible with the filtering mode set on the texture being used. For example, this error code will be returned if a shadow map query is requested when the texture filtering mode is not bilinear.

Define	Value	Description
SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_COMPONENT_COUNT	-2141519856	The call failed because the component count of the texture query format does not match the swizzle of the texture format being used. For example, the texture query could be expecting 4 components, but the texture format could be using a swizzle that returns only 1 component. The list of supported query formats and swizzles for each texture format is available in the *GPU User's Guide* documentation.
SCE_GXM_ERROR_INVALID_SAMPLER_RESULT_TYPE_PRECISION	-2141519857	The call failed because the precision of the texture query format is not supported by the texture format being used. For example, the texture query could be expecting half precision but the texture format only supports float precision. The list of supported query formats for each texture format is available in the *GPU User's Guide* documentation.
SCE_GXM_ERROR_INVALID_TEXTURE	-2141519840	The call failed because an invalid texture was supplied.
SCE_GXM_ERROR_INVALID_TEXTURE_DATA_POINTER	-2141519847	The call failed because an invalid texture data pointer was supplied.
SCE_GXM_ERROR_INVALID_TEXTURE_PALETTE_POINTER	-2141519846	The call failed because an invalid texture palette pointer was supplied.
SCE_GXM_ERROR_INVALID_THREAD	-2141519848	The call failed because it was called from the display queue thread.
SCE_GXM_ERROR_INVALID_VALUE	-2141519869	The call failed because a parameter has an invalid value.

Define	Value	Description
SCE_GXM_ERROR_INVALID_VISIBILITY_BUFFER_POINTER	-2141519843	The call failed because an invalid visibility buffer pointer was supplied.
SCE_GXM_ERROR_INVALID_VISIBILITY_INDEX	-2141519842	The call failed because an invalid visibility index was supplied.
SCE_GXM_ERROR_INVALID_VISIBILITY_OP	-2141519832	The call failed because an invalid visibility operation was used.
SCE_GXM_ERROR_NOT_WITHIN_COMMAND_LIST	-2141519827	The call failed because the deferred context is not currently within a command list.
SCE_GXM_ERROR_NOT_WITHIN_SCENE	-2141519866	The call failed because the rendering context is currently not within a scene.
SCE_GXM_ERROR_NULL_PROGRAM	-2141519864	The call failed because the rendering context currently has a NULL program set.
SCE_GXM_ERROR_OUT_OF_BUFFER_MEMORY	-2141519838	The call failed because a shader patcher buffer memory allocation failed.
SCE_GXM_ERROR_OUT_OF_FRAGMENT_USSE_MEMORY	-2141519836	The call failed because a shader patcher fragment USSE memory allocation failed.
SCE_GXM_ERROR_OUT_OF_HOST_MEMORY	-2141519839	The call failed because a shader patcher host memory allocation failed.
SCE_GXM_ERROR_OUT_OF_MEMORY	-2141519870	The call failed because a memory allocation failed.
SCE_GXM_ERROR_OUT_OF_RENDER_TARGETS	-2141519833	The call failed because the maximum number of render targets have already been created.
SCE_GXM_ERROR_OUT_OF_VERTEX_USSE_MEMORY	-2141519837	The call failed because a shader patcher vertex USSE memory allocation failed.
SCE_GXM_ERROR_PATCHER_INTERNAL	-2141519862	The call failed because the internal patching process failed.
SCE_GXM_ERROR_PROGRAM_IN_USE	-2141519860	The call failed because a program from the shader patcher is still in use.

Define	Value	Description
`SCE_GXM_ERROR_RAZOR`	`-2141519831`	The call failed because of a Razor error.
`SCE_GXM_ERROR_RESERVE_FAILED`	`-2141519861`	The call failed because a buffer failed to reserve space for data.
`SCE_GXM_ERROR_UNIFORM_BUFFER_NOT_RESERVED`	`-2141519855`	The call failed because the uniform buffer has not been reserved.
`SCE_GXM_ERROR_UNINITIALIZED`	`-2141519872`	The call failed because the library is uninitialized.
`SCE_GXM_ERROR_UNSUPPORTED`	`-2141519863`	The call failed because the operation is not supported on this hardware.
`SCE_GXM_ERROR_WITHIN_COMMAND_LIST`	`-2141519828`	The call failed because the deferred context is currently within a command list.
`SCE_GXM_ERROR_WITHIN_SCENE`	`-2141519865`	The call failed because the rendering context is currently within a scene.

Other Constants

Define Summary

Define	Value	Description
SCE_GXM_COLOR_BASE_FORMAT_MASK	0xf1800000U	A mask used to extract the base color format from a SceGxmColorFormat value.
SCE_GXM_COLOR_SURFACE_ALIGNMENT	4U	The minimum alignment of color surface data.
SCE_GXM_COLOR_SWIZZLE_MASK	0x00300000U	A mask used to extract swizzle information from a SceGxmColorFormat value.
SCE_GXM_COMMAND_LIST_WORD_COUNT	8U	The number of 32-bit words used to describe a command list.
SCE_GXM_DEFAULT_FRAGMENT_RING_BUFFER_SIZE	(512*1024)	A default size for the fragment ring buffer in SceGxmContextParams.
SCE_GXM_DEFAULT_FRAGMENT_USSE_RING_BUFFER_SIZE	(16*1024)	A default size for the fragment USSE ring buffer in SceGxmContextParams.
SCE_GXM_DEFAULT_PARAMETER_BUFFER_SIZE	(16*1024*1024)	A default size for the libgxm parameter buffer.
SCE_GXM_DEFAULT_VDM_RING_BUFFER_SIZE	(128*1024)	A default size for the VDM ring buffer in SceGxmContextParams.
SCE_GXM_DEFAULT_VERTEX_RING_BUFFER_SIZE	(2*1024*1024)	A default size for the vertex ring buffer in SceGxmContextParams.
SCE_GXM_DEPTHSTENCIL_SURFACE_ALIGNMENT	16U	The minimum alignment of depth or stencil data.
SCE_GXM_GPU_CORE_COUNT	4U	The number of GPU cores present on the system.
SCE_GXM_MAX_ATTRIBUTE_OVERFETCH	3U	Overfetch may occur due to the DMA fetch for an attribute being rounded up to the next multiple of 4 bytes. This value is the largest amount of overfetch that can occur when fetching the last attribute of the last vertex of a vertex stream. If the size of all attributes is aligned to 4 bytes, then overfetch cannot occur.
SCE_GXM_MAX_RENDER_TARGETS	48U	The maximum number of render targets that can be created.
SCE_GXM_MAX_SCENES_PER_RENDERTARGET	8U	The maximum number of scenes per render target per frame.
SCE_GXM_MINIMUM_CONTEXT_HOST_MEM_SIZE	2048	The minimum required size in bytes of the host memory in SceGxmContextParams.
SCE_GXM_MINIMUM_DEFERRED_CONTEXT_BUFFER_SIZE	1024	The minimum size of a deferred context buffer.
SCE_GXM_NOTIFICATION_COUNT	512	The number of notifications created during sceGxmInitialize(). Each notification is a 32-bit word, that can be used as the address in a SceGxmNotification structure.
SCE_GXM_NUM_TEXTURE_CONTROL_WORDS	4U	The number of 32-bit words required to describe a texture.

Define	Value	Description
SCE_GXM_PALETTE_ ALIGNMENT	64U	The minimum alignment of palette data.
SCE_GXM_PBE_EMIT_ WORD_COUNT	6U	The number of 32-bit words required to describe how a tile is emitted to memory.
SCE_GXM_PRECOMPUTED_ ALIGNMENT	16U	The alignment required for memory that is used for precomputed data.
SCE_GXM_PRECOMPUTED_ DRAW_WORD_COUNT	11	The number of opaque 32-bit words in a precomputed draw command.
SCE_GXM_PRECOMPUTED_ FRAGMENT_STATE_ WORD_COUNT	9	The number of opaque 32-bit words in precomputed fragment state.
SCE_GXM_PRECOMPUTED_ VERTEX_STATE_ WORD_COUNT	7	The number of opaque 32-bit words in precomputed vertex state.
SCE_GXM_RENDER_ TARGET_MACROTILE_ COUNT_X_SHIFT	8	The right-shift to apply to the macrotile count in the X direction when combining with other flags from the SceGxmRenderTargetFlags enumeration. See the documentation for SceGxmRenderTargetParams for details of each flag.
SCE_GXM_RENDER_ TARGET_MACROTILE_ COUNT_Y_SHIFT	12	The right-shift to apply to the macrotile count in the Y direction when combining with other flags from the SceGxmRenderTargetFlags enumeration. See the documentation for SceGxmRenderTargetParams for details of each flag.
SCE_GXM_TEXTURE_ ALIGNMENT	16U	The minimum alignment of texture data that satisfies the alignment requirements of all supported texture formats. When a specific texture format is being used, the required alignment may be less than this value. For details on each format's alignment requirement, please see the *GPU User's Guide*.
SCE_GXM_TEXTURE_ BASE_FORMAT_MASK	0x9f000000U	A mask used to extract the base texture format from a SceGxmTextureFormat value.
SCE_GXM_TEXTURE_ CUBE_NO_ALIGN_ SIZE_16_32BPP	8U	The largest cube face size where the space between each cube map face mip chain does **not** need to be aligned to SCE_GXM_TEXTURE_FACE_ALIGNMENT bytes. This size applies to cube maps of 16 or 32 bits per pixel.
SCE_GXM_TEXTURE_CUBE_ NO_ALIGN_SIZE_64BPP	4U	The largest cube face size where the space between each cube map face mip chain does **not** need to be aligned to SCE_GXM_TEXTURE_FACE_ALIGNMENT bytes. This size applies to cube maps of 64 bits per pixel.
SCE_GXM_TEXTURE_CUBE_ NO_ALIGN_SIZE_8BPP	16U	The largest cube face size where the space between each cube map face mip chain does **not** need to be aligned to SCE_GXM_TEXTURE_FACE_ALIGNMENT bytes. This size applies to cube maps of 8 bits per pixel, or cube maps that use block compression.

Define	Value	Description
SCE_GXM_TEXTURE_FACE_ALIGNMENT	2048U	The minimum alignment of the space between cube map faces when each face has a mip chain, and the length of the top level mip of each face is above the "no align" size. The "no align" size will be either SCE_GXM_TEXTURE_CUBE_NO_ALIGN_SIZE_8BPP or SCE_GXM_TEXTURE_CUBE_NO_ALIGN_SIZE_16_32BPP or SCE_GXM_TEXTURE_CUBE_NO_ALIGN_SIZE_64BPP depending on the texture format of the cube map data. There is no padding between cube map faces for cubemaps less than or equal to the "no align" size.
SCE_GXM_TEXTURE_IMPLICIT_STRIDE_ALIGNMENT	8U	When using linear textures with implicit stride, the stride is formed by aligning the width to this value.
SCE_GXM_TEXTURE_SWIZZLE_MASK	0x00007000U	A mask used to extract swizzle information from a SceGxmTextureFormat value.
SCE_GXM_TILE_SHIFTX	5U	The left-shift to apply to go from tile units to pixel units in the x dimension.
SCE_GXM_TILE_SHIFTY	5U	The left-shift to apply to go from tile units to pixel units in the y dimension.
SCE_GXM_TILE_SIZEX	(1U << SCE_GXM_TILE_SHIFTX)	The size of a tile in memory in the x dimension.
SCE_GXM_TILE_SIZEY	(1U << SCE_GXM_TILE_SHIFTY)	The size of a tile in memory in the y dimension.
SCE_GXM_USSE_ALIGNMENT	16U	The alignment required for USSE programs.
SCE_GXM_VISIBILITY_ALIGNMENT	16U	The alignment required for visibility buffers.

Shader API

Data Types

SceGxmFragmentProgramInput

All possible fragment program inputs.

Definition

```
#include <gxm/program.h>
typedef enum SceGxmFragmentProgramInput {
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_WPOS = (1U << 0),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_FOG = (1U << 1),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_COLOR0 = (1U << 2),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_COLOR1 = (1U << 3),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD0 = (1U << 4),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD1 = (1U << 5),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD2 = (1U << 6),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD3 = (1U << 7),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD4 = (1U << 8),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD5 = (1U << 9),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD6 = (1U << 10),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD7 = (1U << 11),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD8 = (1U << 12),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD9 = (1U << 13),
    SCE_GXM_FRAGMENT_PROGRAM_INPUT_SPRITECOORD = (1U << 14)
} SceGxmFragmentProgramInput;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_FRAGMENT_PROGRAM_INPUT_WPOS	(1U << 0)	Position.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_FOG	(1U << 1)	Fog coordinate.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_COLOR0	(1U << 2)	Color 0.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_COLOR1	(1U << 3)	Color 1.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD0	(1U << 4)	Texture coordinate 0.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD1	(1U << 5)	Texture coordinate 1.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD2	(1U << 6)	Texture coordinate 2.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD3	(1U << 7)	Texture coordinate 3.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD4	(1U << 8)	Texture coordinate 4.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD5	(1U << 9)	Texture coordinate 5.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD6	(1U << 10)	Texture coordinate 6.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD7	(1U << 11)	Texture coordinate 7.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD8	(1U << 12)	Texture coordinate 8.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_TEXCOORD9	(1U << 13)	Texture coordinate 9.
SCE_GXM_FRAGMENT_PROGRAM_INPUT_SPRITECOORD	(1U << 14)	Sprite coordinate.

Description

All possible fragment program inputs. The inputs read by a fragment program can be found by testing the result of `sceGxmProgramGetFragmentProgramInputs` against the values of this enumeration. If the bit is set, then the fragment program reads the input.

SceGxmParameterCategory

The category of shader program parameter.

Definition

```
#include <gxm/program.h>
typedef enum SceGxmParameterCategory {
   SCE_GXM_PARAMETER_CATEGORY_ATTRIBUTE = 0,
   SCE_GXM_PARAMETER_CATEGORY_UNIFORM = 1,
   SCE_GXM_PARAMETER_CATEGORY_SAMPLER = 2,
   SCE_GXM_PARAMETER_CATEGORY_UNIFORM_BUFFER = 4
} SceGxmParameterCategory;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_PARAMETER_CATEGORY_ATTRIBUTE	0	The parameter is a vertex attribute.
SCE_GXM_PARAMETER_CATEGORY_UNIFORM	1	The parameter is a uniform.
SCE_GXM_PARAMETER_CATEGORY_SAMPLER	2	The parameter is a sampler.
SCE_GXM_PARAMETER_CATEGORY_UNIFORM_BUFFER	4	The parameter is a uniform buffer.

Description

The category of shader program parameter.

SceGxmParameterSemantic

The semantic associated with a program parameter.

Definition

```
#include <gxm/program.h>
typedef enum SceGxmParameterSemantic {
    SCE_GXM_PARAMETER_SEMANTIC_NONE,
    SCE_GXM_PARAMETER_SEMANTIC_ATTR,
    SCE_GXM_PARAMETER_SEMANTIC_BCOL,
    SCE_GXM_PARAMETER_SEMANTIC_BINORMAL,
    SCE_GXM_PARAMETER_SEMANTIC_BLENDINDICES,
    SCE_GXM_PARAMETER_SEMANTIC_BLENDWEIGHT,
    SCE_GXM_PARAMETER_SEMANTIC_COLOR,
    SCE_GXM_PARAMETER_SEMANTIC_DIFFUSE,
    SCE_GXM_PARAMETER_SEMANTIC_FOGCOORD,
    SCE_GXM_PARAMETER_SEMANTIC_NORMAL,
    SCE_GXM_PARAMETER_SEMANTIC_POINTSIZE,
    SCE_GXM_PARAMETER_SEMANTIC_POSITION,
    SCE_GXM_PARAMETER_SEMANTIC_SPECULAR,
    SCE_GXM_PARAMETER_SEMANTIC_TANGENT,
    SCE_GXM_PARAMETER_SEMANTIC_TEXCOORD
} SceGxmParameterSemantic;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_PARAMETER_SEMANTIC_NONE`	N/A	The parameter has no semantic.
`SCE_GXM_PARAMETER_SEMANTIC_ATTR`	N/A	The parameter has an ATTR semantic.
`SCE_GXM_PARAMETER_SEMANTIC_BCOL`	N/A	The parameter has a BCOL semantic.
`SCE_GXM_PARAMETER_SEMANTIC_BINORMAL`	N/A	The parameter has a BINORMAL semantic.
`SCE_GXM_PARAMETER_SEMANTIC_BLENDINDICES`	N/A	The parameter has a BLENDINDICES semantic.
`SCE_GXM_PARAMETER_SEMANTIC_BLENDWEIGHT`	N/A	The parameter has a BLENDWEIGHT semantic.
`SCE_GXM_PARAMETER_SEMANTIC_COLOR`	N/A	The parameter has a COLOR semantic.
`SCE_GXM_PARAMETER_SEMANTIC_DIFFUSE`	N/A	The parameter has a DIFFUSE semantic.
`SCE_GXM_PARAMETER_SEMANTIC_FOGCOORD`	N/A	The parameter has a FOGCOORD semantic.
`SCE_GXM_PARAMETER_SEMANTIC_NORMAL`	N/A	The parameter has a NORMAL semantic.
`SCE_GXM_PARAMETER_SEMANTIC_POINTSIZE`	N/A	The parameter has a POINTSIZE semantic.
`SCE_GXM_PARAMETER_SEMANTIC_POSITION`	N/A	The parameter has a POSITION semantic.
`SCE_GXM_PARAMETER_SEMANTIC_SPECULAR`	N/A	The parameter has a SPECULAR semantic.
`SCE_GXM_PARAMETER_SEMANTIC_TANGENT`	N/A	The parameter has a TANGENT semantic.
`SCE_GXM_PARAMETER_SEMANTIC_TEXCOORD`	N/A	The parameter has a TEXCOORD semantic.

Description

The semantic associated with a program parameter. For vertex attribute parameters, this is the semantic that was associated with the declaration. Semantics serve as an alternative to name based lookups and allow for parameters being described in terms of their function rather than their name. Each semantic is indexed with an index value between and 0 and 15.

Semantics are not used for parameters that are not vertex attributes; non- attribute parameters will always report `SCE_GXM_PARAMETER_SEMANTIC_NONE`.

SceGxmParameterType

The data type of program parameters.

Definition

```
#include <gxm/program.h>
typedef enum SceGxmParameterType {
    SCE_GXM_PARAMETER_TYPE_F32,
    SCE_GXM_PARAMETER_TYPE_F16,
    SCE_GXM_PARAMETER_TYPE_C10,
    SCE_GXM_PARAMETER_TYPE_U32,
    SCE_GXM_PARAMETER_TYPE_S32,
    SCE_GXM_PARAMETER_TYPE_U16,
    SCE_GXM_PARAMETER_TYPE_S16,
    SCE_GXM_PARAMETER_TYPE_U8,
    SCE_GXM_PARAMETER_TYPE_S8,
    SCE_GXM_PARAMETER_TYPE_AGGREGATE
} SceGxmParameterType;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_PARAMETER_TYPE_F32	N/A	32-bit floating point, equivalent to the Cg type "float".
SCE_GXM_PARAMETER_TYPE_F16	N/A	16-bit floating point, equivalent to the Cg type "half".
SCE_GXM_PARAMETER_TYPE_C10	N/A	10-bit fixed point, equivalent to the Cg type "fixed".
SCE_GXM_PARAMETER_TYPE_U32	N/A	32-bit unsigned integer.
SCE_GXM_PARAMETER_TYPE_S32	N/A	32-bit signed integer.
SCE_GXM_PARAMETER_TYPE_U16	N/A	16-bit unsigned integer.
SCE_GXM_PARAMETER_TYPE_S16	N/A	16-bit signed integer.
SCE_GXM_PARAMETER_TYPE_U8	N/A	8-bit unsigned integer.
SCE_GXM_PARAMETER_TYPE_S8	N/A	8-bit signed integer.
SCE_GXM_PARAMETER_TYPE_AGGREGATE	N/A	Aggregate type (used for uniform buffers).

Description

The data type of program parameters. For uniform parameters, this is the data type that uniform data should be stored as in the buffer. For sampler parameters, this is the result type of the texture data that is expected by the shader code.

SceGxmProgram

The opaque data type for the Cg compiler output.

Definition

```
#include <gxm/program.h>
typedef struct SceGxmProgram;
```

Description

The opaque data type for the Cg compiler output.

SceGxmProgramParameter

The opaque data type for the parameter of a program.

Definition

```
#include <gxm/program.h>
typedef struct SceGxmProgramParameter;
```

Description

The opaque data type for the parameter of a program.

SceGxmProgramType

The type of program.

Definition

```
#include <gxm/program.h>
typedef enum SceGxmProgramType {
    SCE_GXM_VERTEX_PROGRAM,
    SCE_GXM_FRAGMENT_PROGRAM
} SceGxmProgramType;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_VERTEX_PROGRAM`	N/A	The shader is a vertex program.
`SCE_GXM_FRAGMENT_PROGRAM`	N/A	The shader is a fragment program.

Description

The type of program.

SceGxmVertexProgramOutput

All possible vertex program outputs.

Definition

```
#include <gxm/program.h>
typedef enum SceGxmVertexProgramOutput {
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_POSITION = (1U << 0),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_FOG = (1U << 1),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_COLOR0 = (1U << 2),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_COLOR1 = (1U << 3),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD0 = (1U << 4),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD1 = (1U << 5),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD2 = (1U << 6),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD3 = (1U << 7),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD4 = (1U << 8),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD5 = (1U << 9),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD6 = (1U << 10),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD7 = (1U << 11),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD8 = (1U << 12),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD9 = (1U << 13),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_PSIZE = (1U << 14),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP0 = (1U << 15),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP1 = (1U << 16),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP2 = (1U << 17),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP3 = (1U << 18),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP4 = (1U << 19),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP5 = (1U << 20),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP6 = (1U << 21),
    SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP7 = (1U << 22)
} SceGxmVertexProgramOutput;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_POSITION`	`(1U << 0)`	Position.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_FOG`	`(1U << 1)`	Fog coordinate.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_COLOR0`	`(1U << 2)`	Color 0.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_COLOR1`	`(1U << 3)`	Color 1.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD0`	`(1U << 4)`	Texture coordinate 0.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD1`	`(1U << 5)`	Texture coordinate 1.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD2`	`(1U << 6)`	Texture coordinate 2.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD3`	`(1U << 7)`	Texture coordinate 3.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD4`	`(1U << 8)`	Texture coordinate 4.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD5`	`(1U << 9)`	Texture coordinate 5.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD6`	`(1U << 10)`	Texture coordinate 6.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD7`	`(1U << 11)`	Texture coordinate 7.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD8`	`(1U << 12)`	Texture coordinate 8.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_TEXCOORD9`	`(1U << 13)`	Texture coordinate 9.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_PSIZE`	`(1U << 14)`	Point size.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP0`	`(1U << 15)`	Clip plane 0.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP1`	`(1U << 16)`	Clip plane 1.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP2`	`(1U << 17)`	Clip plane 2.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP3`	`(1U << 18)`	Clip plane 3.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP4`	`(1U << 19)`	Clip plane 4.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP5`	`(1U << 20)`	Clip plane 5.

Macro	Value	Description
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP6`	`(1U << 21)`	Clip plane 6.
`SCE_GXM_VERTEX_PROGRAM_OUTPUT_CLIP7`	`(1U << 22)`	Clip plane 7.

Description

All possible vertex program outputs. The outputs written by a vertex program can be found by testing the result of `sceGxmProgramGetVertexProgramOutputs` against the values of this enumeration. If the bit is set, then the vertex program writes the output.

Functions

sceGxmProgramCheck

Returns `SCE_OK` if the given pointer looks like the compiler output.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE SceGxmErrorCode sceGxmProgramCheck(
    const SceGxmProgram *program
);
```

Arguments

[in] *program*	A pointer to the program.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because the header magic number was invalid or the version numbers are not compatible.

Description

Returns `SCE_OK` if the given pointer looks like the compiler output.

sceGxmProgramFindParameterByName

Finds a parameter by matching the given name.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE const SceGxmProgramParameter
*sceGxmProgramFindParameterByName(
    const SceGxmProgram *program,
    const char *name
);
```

Arguments

[in] *program*	A pointer to the program. This must not be `NULL`.
[in] *name*	The name of the parameter to search for.

Return Values

A pointer to the parameter or `NULL` if it was not found.

Description

Finds a parameter by matching the given name.

sceGxmProgramFindParameterBySemantic

Finds a vertex attribute parameter by matching the semantic and index.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE const SceGxmProgramParameter
*sceGxmProgramFindParameterBySemantic(
   const SceGxmProgram *program,
   SceGxmParameterSemantic semantic,
   uint32_t index
);
```

Arguments

[in] *program*	A pointer to the program. This must not be `NULL`.
[in] *semantic*	The semantic to search for.
[in] *index*	The index within this semantic to match.

Return Values

A pointer to the parameter or `NULL` if it was not found.

Description

Finds a vertex attribute parameter by matching the semantic and index. This function will only ever match vertex attributes because other categories of parameter do not have semantics.

Notes

Passing the `SCE_GXM_PARAMETER_SEMANTIC_NONE` semantic will not match any parameters.

sceGxmProgramGetDefaultUniformBufferSize

Gets the size of the default uniform buffer for the given program.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE uint32_t sceGxmProgramGetDefaultUniformBufferSize(
    const SceGxmProgram *program
);
```

Arguments

[in] *program*	A pointer to the program. This must not be `NULL`.

Return Values

The size of the default uniform buffer in bytes.

Description

Gets the size of the default uniform buffer for the given program.

sceGxmProgramGetFragmentProgramInputs

Gets the set of fragment program inputs in use by the given program.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE uint32_t sceGxmProgramGetFragmentProgramInputs(
    const SceGxmProgram *program
);
```

Arguments

[in] *program*	A pointer to the program. This must not be NULL.

Return Values

The set of fragment program inputs. 0 is returned if the program is not a fragment program.

Description

Gets the set of fragment program inputs in use by the given program. The return value is formed by combining values of the SceGxmFragmentProgramInput enumeration using bitwise OR.

sceGxmProgramGetOutputRegisterFormat

Returns the type and component count for a fragment program's output.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE SceGxmErrorCode sceGxmProgramGetOutputRegisterFormat(
    const SceGxmProgram *program,
    SceGxmParameterType *type,
    uint32_t *componentCount
);
```

Arguments

[in] *program*	A pointer to the program.
[out] *type*	Storage for the output type.
[out] *componentCount*	Storage for the output component count.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed due to an invalid input pointer.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because the supplied program was not a fragment program.

Description

Returns the type and component count for a fragment program's output.

sceGxmProgramGetParameter

Gets a parameter exposed by the given program by index.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE const SceGxmProgramParameter *sceGxmProgramGetParameter(
    const SceGxmProgram *program,
    uint32_t index
);
```

Arguments

[in] *program*	A pointer to the program. This must not be `NULL`.
[in] *index*	The index in the parameter array.

Return Values

A pointer to the parameter or `NULL` if the index is out of bounds.

Description

Gets a parameter exposed by the given program by index.

sceGxmProgramGetParameterCount

Gets the number of parameters exposed by the given program.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE uint32_t sceGxmProgramGetParameterCount(
    const SceGxmProgram *program
);
```

Arguments

[in] *program*	A pointer to the program. This must not be `NULL`.

Return Values

The number of parameters exposed.

Description

Gets the number of parameters exposed by the given program.

sceGxmProgramGetSize

Gets the total size of the compiler output.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE uint32_t sceGxmProgramGetSize(
    const SceGxmProgram *program
);
```

Arguments

[in] *program*	A pointer to the program. This must not be `NULL`.

Return Values

The size of the program in bytes.

Description

Gets the total size of the compiler output.

sceGxmProgramGetType

Gets the pipeline that the program is expected to run on.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE SceGxmProgramType sceGxmProgramGetType(
    const SceGxmProgram *program
);
```

Arguments

[in] *program*	A pointer to the program. This must not be `NULL`.

Return Values

The type of program.

Description

Gets the pipeline that the program is expected to run on.

sceGxmProgramGetVertexProgramOutputs

Gets the set of vertex program outputs in use by the given program.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE uint32_t sceGxmProgramGetVertexProgramOutputs(
    const SceGxmProgram *program
);
```

Arguments

[in] *program*	A pointer to the program. This must not be NULL.

Return Values

The set of vertex program outputs. 0 is returned if the program is not a vertex program.

Description

Gets the set of vertex program outputs in use by the given program. The return value is formed by combining values of the SceGxmVertexProgramOutput enumeration using bitwise OR.

sceGxmProgramIsDepthReplaceUsed

Checks whether the fragment program replaces the depth value.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE bool sceGxmProgramIsDepthReplaceUsed(
    const SceGxmProgram *program
);
```

Arguments

[in] *program*	A pointer to the program. This must not be `NULL`.

Return Values

True only if the fragment program uses depth replace, false otherwise.

Description

Checks whether the fragment program replaces the depth value. This function always returns false for vertex programs.

A fragment program uses depth replace if it writes to an output variable with the DEPTH semantic. This output depth value replaces the interpolated depth value over the primitive.

sceGxmProgramIsDiscardUsed

Checks whether the fragment program uses discard.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE bool sceGxmProgramIsDiscardUsed(
    const SceGxmProgram *program
);
```

Arguments

[in] *program*	A pointer to the program. This must not be `NULL`.

Return Values

True only if the fragment program uses discard, false otherwise.

Description

Checks whether the fragment program uses discard. This function always returns false for vertex programs.

sceGxmProgramIsEquivalent

Checks if the compiler output for two shaders is equivalent.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE bool sceGxmProgramIsEquivalent(
    const SceGxmProgram *programA,
    const SceGxmProgram *programB
);
```

Arguments

[in] *programA*	A pointer to the first program. This must not be `NULL`.
[in] *programB*	A pointer to the second program. This must not be `NULL`.

Return Values

Returns `true` if *programA* and *programB* are equivalent; otherwise `false` is returned.

Description

Checks if the compiler output for two shaders is equivalent. This function ignores shader association data used by Razor to disambiguate between shaders that produces identical shader compiler output.

sceGxmProgramIsFragColorUsed

Checks whether the fragment program uses the FRAGCOLOR semantic.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE bool sceGxmProgramIsFragColorUsed(
    const SceGxmProgram *program
);
```

Arguments

[in] *program*	A pointer to the program. This must not be NULL.

Return Values

If the fragment program uses FRAGCOLOR, true is returned; otherwise the function returns false.

Description

Checks whether the fragment program uses the FRAGCOLOR semantic. This function always returns false for vertex programs.

sceGxmProgramIsNativeColorUsed

Checks whether the fragment program uses `__nativecolor`.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE bool sceGxmProgramIsNativeColorUsed(
    const SceGxmProgram *program
);
```

Arguments

[in] *program*	A pointer to the program. This must not be `NULL`.

Return Values

If the fragment program uses `__nativecolor`, true is returned; otherwise the function returns false.

Description

Checks whether the fragment program uses `__nativecolor`. This function always returns false for vertex programs.

sceGxmProgramIsSpriteCoordUsed

Checks whether the fragment program uses the SPRITECOORD semantic.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE bool sceGxmProgramIsSpriteCoordUsed(
    const SceGxmProgram *program
);
```

Arguments

[in] *program*	A pointer to the program. This must not be NULL.

Return Values

True only if the fragment program uses SPRITECOORD, false otherwise.

Description

Checks whether the fragment program uses the SPRITECOORD semantic. This function always returns false for vertex programs.

sceGxmProgramParameterGetArraySize

Gets the number of elements in the array.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE uint32_t sceGxmProgramParameterGetArraySize(
   const SceGxmProgramParameter *parameter
);
```

Arguments

[in] *parameter*	A pointer to a program parameter. This must not be `NULL`.

Return Values

The size of the element array.

Description

Gets the number of elements in the array.

sceGxmProgramParameterGetCategory

Gets the type of category of a given parameter.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE SceGxmParameterCategory sceGxmProgramParameterGetCategory(
    const SceGxmProgramParameter *parameter
);
```

Arguments

[in] *parameter*	A pointer to the program parameter. This must not be `NULL`.

Return Values

The type of category of the parameter.

Description

Gets the type of category of a given parameter.

sceGxmProgramParameterGetComponentCount

Gets the number of scalar components per array element.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE uint32_t sceGxmProgramParameterGetComponentCount(
    const SceGxmProgramParameter *parameter
);
```

Arguments

[in] *parameter*	A pointer to a program parameter. This must not be `NULL`.

Return Values

The component count of the parameter.

Description

Gets the number of scalar components per array element. This will always be a value between 1 and 4.

sceGxmProgramParameterGetContainerIndex

Gets the container index of the given parameter.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE uint32_t sceGxmProgramParameterGetContainerIndex(
    const SceGxmProgramParameter *parameter
);
```

Arguments

[in] *parameter*	A pointer to a program parameter. This must not be `NULL`.

Return Values

The container index of the given parameter

Description

Gets the container index of the given parameter. Container index values are used as follows:

- Attributes: unused
- Uniforms: uniform buffer index
- Samplers: unused
- Uniform buffers: unused

When the parameter is a member of the default uniform buffer, the container index will have a value of `SCE_GXM_MAX_UNIFORM_BUFFERS`.

sceGxmProgramParameterGetIndex

Gets the index of the given parameter within the program.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE uint32_t sceGxmProgramParameterGetIndex(
    const SceGxmProgram *program,
    const SceGxmProgramParameter *parameter
);
```

Arguments

[in] *program*	A pointer to the parent program. This must not be `NULL`.
[in] *parameter*	A pointer to the program parameter. This must not be `NULL`.

Return Values

The index of the parameter within the program.

Description

Gets the index of the given parameter within the program. This index will always be greater than or equal to 0 and less than the value returned by `sceGxmProgramGetParameterCount()`.

sceGxmProgramParameterGetName

Gets the name of the given parameter.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE const char *sceGxmProgramParameterGetName(
    const SceGxmProgramParameter *parameter
);
```

Arguments

[in] *parameter*	A pointer to the program parameter. This must not be `NULL`.

Return Values

The name of the parameter.

Description

Gets the name of the given parameter.

sceGxmProgramParameterGetResourceIndex

Gets the resource index of the given parameter.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE uint32_t sceGxmProgramParameterGetResourceIndex(
    const SceGxmProgramParameter *parameter
);
```

Arguments

[in] *parameter*	A pointer to a program parameter. This must not be `NULL`.

Return Values

The resource index of the given parameter.

Description

Gets the resource index of the given parameter. Resources index values are used as follows:

- Attributes: register index
- Uniforms: word (32-bit) offset from start of buffer
- Samplers: texture unit index
- Uniform buffers: uniform buffer index

sceGxmProgramParameterGetSemantic

Gets the semantic of the given parameter.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE SceGxmParameterSemantic sceGxmProgramParameterGetSemantic(
    const SceGxmProgramParameter *parameter
);
```

Arguments

[in] *parameter*	A pointer to the program parameter. This must not be `NULL`.

Return Values

The name of the parameter.

Description

Gets the semantic of the given parameter.

sceGxmProgramParameterGetSemanticIndex

Gets the semantic index of the given parameter.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE uint32_t sceGxmProgramParameterGetSemanticIndex(
    const SceGxmProgramParameter *parameter
);
```

Arguments

[in] *parameter*	A pointer to the program parameter. This must not be `NULL`.

Return Values

The semantic index of the parameter.

Description

Gets the semantic index of the given parameter.

sceGxmProgramParameterGetType

Gets the scalar data type of the given parameter.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE SceGxmParameterType sceGxmProgramParameterGetType(
    const SceGxmProgramParameter *parameter
);
```

Arguments

[in] *parameter*	The pointer to a program parameter. This must not be `NULL`.

Return Values

The type of the parameter.

Description

Gets the scalar data type of the given parameter.

sceGxmProgramParameterIsRegFormat

For parameters of category SCE_GXM_PARAMETER_CATEGORY_ATTRIBUTE, this function returns true if the given parameter has a __regformat qualifier.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE bool sceGxmProgramParameterIsRegFormat(
    const SceGxmProgram *program,
    const SceGxmProgramParameter *parameter
);
```

Arguments

[in] *program*	A pointer to the parent program. This must not be NULL.
[in] *parameter*	A pointer to a program parameter. This must not be NULL.

Return Values

True only if the parameter is an attribute with a __regformat qualifier, false otherwise.

Description

For parameters of category SCE_GXM_PARAMETER_CATEGORY_ATTRIBUTE, this function returns true if the given parameter has a __regformat qualifier. For parameters that are not of category SCE_GXM_PARAMETER_CATEGORY_ATTRIBUTE, this function always returns false.

sceGxmProgramParameterIsSamplerCube

For parameters of category SCE_GXM_PARAMETER_CATEGORY_SAMPLER, this function returns true if the given parameter is a cube map sampler.

Definition

```
#include <gxm/program.h>
SCE_GXM_INTERFACE bool sceGxmProgramParameterIsSamplerCube(
    const SceGxmProgramParameter *parameter
);
```

Arguments

[in] *parameter*	A pointer to a program parameter. This must not be NULL.

Return Values

True only if the parameter is a cube map sampler, false otherwise.

Description

For parameters of category SCE_GXM_PARAMETER_CATEGORY_SAMPLER, this function returns true if the given parameter is a cube map sampler. For parameters that are not of category SCE_GXM_PARAMETER_CATEGORY_SAMPLER, this function always returns false.

Other Constants

Define Summary

Define	Value	Description
`SCE_GXM_MAX_TEXTURE_UNITS`	`16U`	The maximum number of texture units supported by Cg and the libgxm API.
`SCE_GXM_MAX_UNIFORM_BUFFERS`	`14U`	The maximum number of uniform buffers supported by Cg and the libgxm API.
`SCE_GXM_MAX_VERTEX_ATTRIBUTES`	`16U`	The maximum number of (vector) vertex attributes supported by Cg and the libgxm API.
`SCE_GXM_MAX_VERTEX_STREAMS`	`16U`	The maximum number of independent vertex stream addresses supported by Cg and the libgxm API.

Shader Patcher API

Data Types

SceGxmBlendFactor

The blend factors for the runtime patching of shader code by the shader patcher.

Definition

```
#include <gxm/blending.h>
typedef enum SceGxmBlendFactor {
    SCE_GXM_BLEND_FACTOR_ZERO,
    SCE_GXM_BLEND_FACTOR_ONE,
    SCE_GXM_BLEND_FACTOR_SRC_COLOR,
    SCE_GXM_BLEND_FACTOR_ONE_MINUS_SRC_COLOR,
    SCE_GXM_BLEND_FACTOR_SRC_ALPHA,
    SCE_GXM_BLEND_FACTOR_ONE_MINUS_SRC_ALPHA,
    SCE_GXM_BLEND_FACTOR_DST_COLOR,
    SCE_GXM_BLEND_FACTOR_ONE_MINUS_DST_COLOR,
    SCE_GXM_BLEND_FACTOR_DST_ALPHA,
    SCE_GXM_BLEND_FACTOR_ONE_MINUS_DST_ALPHA,
    SCE_GXM_BLEND_FACTOR_SRC_ALPHA_SATURATE,
    SCE_GXM_BLEND_FACTOR_DST_ALPHA_SATURATE
} SceGxmBlendFactor;
```

Enumeration Values

Macro	Value	Description
SCE_GXM_BLEND_FACTOR_ZERO	N/A	(0, 0, 0, 0)
SCE_GXM_BLEND_FACTOR_ONE	N/A	(1, 1, 1, 1)
SCE_GXM_BLEND_FACTOR_SRC_COLOR	N/A	(R_s, G_s, B_s, A_s)
SCE_GXM_BLEND_FACTOR_ONE_MINUS_SRC_COLOR	N/A	(1 - R_s, 1 - G_s, 1 - B_s, 1 - A_s)
SCE_GXM_BLEND_FACTOR_SRC_ALPHA	N/A	(A_s, A_s, A_s, A_s)
SCE_GXM_BLEND_FACTOR_ONE_MINUS_SRC_ALPHA	N/A	(1 - A_s, 1 - A_s, 1 - A_s, 1 - A_s)
SCE_GXM_BLEND_FACTOR_DST_COLOR	N/A	(R_d, G_d, B_d, A_d)
SCE_GXM_BLEND_FACTOR_ONE_MINUS_DST_COLOR	N/A	(1 - R_d, 1 - G_d, 1 - B_d, 1 - A_d)
SCE_GXM_BLEND_FACTOR_DST_ALPHA	N/A	(A_d, A_d, A_d, A_d)
SCE_GXM_BLEND_FACTOR_ONE_MINUS_DST_ALPHA	N/A	(1 - A_d, 1 - A_d, 1 - A_d, 1 - A_d)
SCE_GXM_BLEND_FACTOR_SRC_ALPHA_SATURATE	N/A	(i, i, i, i) where i = min(A_s, 1 - A_d)
SCE_GXM_BLEND_FACTOR_DST_ALPHA_SATURATE	N/A	(i, i, i, i) where i = min(A_d, 1 - A_s)

Description

The blend factors for the runtime patching of shader code by the shader patcher. The individual descriptions are in RGBA order.

SceGxmBlendFunc

The blend functions for the runtime patching of shader code by the shader patcher.

Definition

```
#include <gxm/blending.h>
typedef enum SceGxmBlendFunc {
    SCE_GXM_BLEND_FUNC_NONE,
    SCE_GXM_BLEND_FUNC_ADD,
    SCE_GXM_BLEND_FUNC_SUBTRACT,
    SCE_GXM_BLEND_FUNC_REVERSE_SUBTRACT,
    SCE_GXM_BLEND_FUNC_MIN,
    SCE_GXM_BLEND_FUNC_MAX
} SceGxmBlendFunc;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_BLEND_FUNC_NONE`	N/A	D = S.
`SCE_GXM_BLEND_FUNC_ADD`	N/A	D = S * factorS + D * factorD.
`SCE_GXM_BLEND_FUNC_SUBTRACT`	N/A	D = S * factorS - D * factorD.
`SCE_GXM_BLEND_FUNC_REVERSE_SUBTRACT`	N/A	D = D * factorD - S * factorS.
`SCE_GXM_BLEND_FUNC_MIN`	N/A	D = min(S, D)
`SCE_GXM_BLEND_FUNC_MAX`	N/A	D = max(S, D)

Description

The blend functions for the runtime patching of shader code by the shader patcher.

Notes

Use of `SCE_GXM_BLEND_FUNC_MIN` or `SCE_GXM_BLEND_FUNC_MAX` requires that the blend factor is set to `SCE_GXM_BLEND_FACTOR_ONE`.

SceGxmBlendInfo

Describes the blending and masking of the shader result into the color surface.

Definition

```
#include <gxm/blending.h>
typedef struct SceGxmBlendInfo {
   uint8_t colorMask;
   uint8_t colorFunc:4;
   uint8_t alphaFunc:4;
   uint8_t colorSrc:4;
   uint8_t colorDst:4;
   uint8_t alphaSrc:4;
   uint8_t alphaDst:4;
} SceGxmBlendInfo;
```

Members

colorMask	Mask bitfield using values from `SceGxmColorMask`.
colorFunc	Color blend function, from `SceGxmBlendFunc`.
alphaFunc	Alpha blend function, from `SceGxmBlendFunc`.
colorSrc	Source color blend factor, from `SceGxmBlendFactor`.
colorDst	Destination color blend factor, from `SceGxmBlendFactor`.
alphaSrc	Source alpha blend factor, from `SceGxmBlendFactor`.
alphaDst	Destination alpha blend factor, from `SceGxmBlendFactor`.

Description

Describes the blending and masking of the shader result into the color surface. Used for runtime patching of the shader code by the shader patcher.

SceGxmColorMask

The color masks for the runtime patching of shader code by the shader patcher.

Definition

```
#include <gxm/blending.h>
typedef enum SceGxmColorMask {
    SCE_GXM_COLOR_MASK_NONE = 0,
    SCE_GXM_COLOR_MASK_A = (1 << 0),
    SCE_GXM_COLOR_MASK_R = (1 << 1),
    SCE_GXM_COLOR_MASK_G = (1 << 2),
    SCE_GXM_COLOR_MASK_B = (1 << 3),
    SCE_GXM_COLOR_MASK_ALL = (SCE_GXM_COLOR_MASK_A | SCE_GXM_COLOR_MASK_B |
    SCE_GXM_COLOR_MASK_G | SCE_GXM_COLOR_MASK_R)
} SceGxmColorMask;
```

Enumeration Values

Macro	Value	Description
`SCE_GXM_COLOR_MASK_NONE`	`0`	No channels written.
`SCE_GXM_COLOR_MASK_A`	`(1 << 0)`	Alpha channel written.
`SCE_GXM_COLOR_MASK_R`	`(1 << 1)`	Red channel written.
`SCE_GXM_COLOR_MASK_G`	`(1 << 2)`	Green channel written.
`SCE_GXM_COLOR_MASK_B`	`(1 << 3)`	Blue channel written.
`SCE_GXM_COLOR_MASK_ALL`	`(SCE_GXM_COLOR_MASK_A | SCE_GXM_COLOR_MASK_B | SCE_GXM_COLOR_MASK_G | SCE_GXM_COLOR_MASK_R)`	All channels written.

Description

The color masks for the runtime patching of shader code by the shader patcher.

SceGxmRegisteredProgram

The opaque structure for a registered program.

Definition

```
#include <gxm/shader_patcher.h>
typedef struct SceGxmRegisteredProgram;
```

Description

The opaque structure for a registered program.

SceGxmShaderPatcher

The opaque structure for shader patcher internals.

Definition

```
#include <gxm/shader_patcher.h>
typedef struct SceGxmShaderPatcher;
```

Description

The opaque structure for shader patcher internals.

SceGxmShaderPatcherId

The opaque structure for a registered program, exposed as an ID.

Definition

```
#include <gxm/shader_patcher.h>
typedef struct SceGxmRegisteredProgram *SceGxmShaderPatcherId;
```

Description

The opaque structure for a registered program, exposed as an ID.

SceGxmShaderPatcherParams

The initialization parameters for the shader patcher.

Definition

```
#include <gxm/shader_patcher.h>
typedef struct SceGxmShaderPatcherParams {
    void *userData;
    SceGxmShaderPatcherHostAllocCallback hostAllocCallback;
    SceGxmShaderPatcherHostFreeCallback hostFreeCallback;
    SceGxmShaderPatcherBufferAllocCallback bufferAllocCallback;
    SceGxmShaderPatcherBufferFreeCallback bufferFreeCallback;
    void *bufferMem;
    uint32_t bufferMemSize;
    SceGxmShaderPatcherUsseAllocCallback vertexUsseAllocCallback;
    SceGxmShaderPatcherUsseFreeCallback vertexUsseFreeCallback;
    void *vertexUsseMem;
    uint32_t vertexUsseMemSize;
    uint32_t vertexUsseOffset;
    SceGxmShaderPatcherUsseAllocCallback fragmentUsseAllocCallback;
    SceGxmShaderPatcherUsseFreeCallback fragmentUsseFreeCallback;
    void *fragmentUsseMem;
    uint32_t fragmentUsseMemSize;
    uint32_t fragmentUsseOffset;
} SceGxmShaderPatcherParams;
```

Members

userData	The user data parameter to pass to memory callbacks.
hostAllocCallback	The callback for host memory allocations. This cannot be `NULL`.
hostFreeCallback	The callback for host memory deallocations. This cannot be `NULL`.
bufferAllocCallback	The callback for GPU buffer memory allocations. This callback can be `NULL` if a pre-allocated region for buffers is provided, in which case the shader patcher will manage a heap internally using this region.
bufferFreeCallback	The callback for GPU buffer memory deallocations. This callback can be `NULL` if a pre-allocated region for buffers is provided, in which case the shader patcher will manage a heap internally using this region.
bufferMem	The pre-allocated memory region for GPU buffers. This region must be `NULL` if buffer allocation callbacks are provided.
bufferMemSize	The size of the pre-allocated memory region for GPU buffers. This size must be zero if buffer allocation callbacks are provided.
vertexUsseAllocCallback	The callback for vertex USSE code memory allocations. This callback can be `NULL` if a pre-allocated region for vertex USSE code is provided, in which case the shader patcher will manage a heap internally using this region.
vertexUsseFreeCallback	The callback for vertex USSE code memory deallocations. This callback can be `NULL` if a pre-allocated region for vertex USSE code is provided, in which case the shader patcher will manage a heap internally using this region.
vertexUsseMem	The pre-allocated memory region for vertex USSE code. This region must be `NULL` if vertex USSE allocation callbacks are provided.

vertexUsseMemSize	The size of the pre-allocated memory region for vertex USSE code. This size must be zero if vertex USSE allocation callbacks are provided.
vertexUsseOffset	The USSE offset of the pre-allocated memory region for vertex USSE code. This field is ignored if vertex USSE allocation callbacks are provided.
fragmentUsseAllocCallback	The callback for fragment USSE code memory allocations. This callback can be `NULL` if a pre-allocated region for fragment USSE code is provided, in which case the shader patcher will manage a heap internally using this region.
fragmentUsseFreeCallback	The callback for fragment USSE code memory deallocations. This callback can be `NULL` if a pre-allocated region for fragment USSE code is provided, in which case the shader patcher will manage a heap internally using this region.
fragmentUsseMem	The pre-allocated memory region for fragment USSE code. This region must be `NULL` if fragment USSE allocation callbacks are provided.
fragmentUsseMemSize	The size of the pre-allocated memory region for fragment USSE code. This size must be zero if fragment USSE allocation callbacks are provided.
fragmentUsseOffset	The USSE offset of the pre-allocated memory region for fragment USSE code. This field is ignored if fragment USSE allocation callbacks are provided.

Description

The initialization parameters for the shader patcher. Host memory must always be provided using the callbacks *hostAllocCallback* and *hostFreeCallback*.

For the remaining categories of callback, either an alloc/free callback pair must be provided, or a static allocation. When a static allocation is provided, the shader patcher will internally create a heap on that allocation, implementing the callbacks itself.

If the base addresses and sizes of the vertex USSE memory and fragment USSE memory match, then the allocation is treated as a shared heap. For this case, the memory must be mapped as both vertex and fragment USSE code using `sceGxmMapVertexUsseMemory()` and `sceGxmMapFragmentUsseMemory()`, which allows for vertex and fragment code to share a memory budget. If the base addresses or sizes are different, then the heaps will be independent.

Functions

sceGxmShaderPatcherAddRefFragmentProgram

Increases the reference count of the fragment program.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherAddRefFragmentProgram(
    SceGxmShaderPatcher *shaderPatcher,
    SceGxmFragmentProgram *fragmentProgram
);
```

Arguments

[in,out] *shaderPatcher*	A pointer to the shader patcher.
[in,out] *fragmentProgram*	A pointer to the fragment program whose reference count should be increased.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because one or more of the pointers supplied was `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because the program was not in the cache.

Description

Increases the reference count of the fragment program. The fragment program must have been created by the shader patcher.

sceGxmShaderPatcherAddRefVertexProgram

Increases the reference count of the vertex program.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherAddRefVertexProgram(
    SceGxmShaderPatcher *shaderPatcher,
    SceGxmVertexProgram *vertexProgram
);
```

Arguments

[in,out] `shaderPatcher`	A pointer to the shader patcher.
[in,out] `vertexProgram`	A pointer to the vertex program whose reference count should be increased.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because one or more of the pointers supplied was `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because the program was not in the cache.

Description

Increases the reference count of the vertex program. The vertex program must have been created by the shader patcher.

sceGxmShaderPatcherCreate

Creates a new shader patcher.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherCreate(
    const SceGxmShaderPatcherParams *params,
    SceGxmShaderPatcher **shaderPatcher
);
```

Arguments

[in] *params*	The shader patcher parameters, including host alloc/free callbacks. This structure does not need to persist after this function returns.
[out] *shaderPatcher*	A pointer to storage for a shader patcher pointer. The host allocation callback is used to allocate memory for the shader patcher internal structures.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because the input parameter was `NULL`.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed because some allocated memory did not have the required alignment.
`SCE_GXM_ERROR_OUT_OF_HOST_MEMORY`	The operation failed because a host memory allocation failed.

Description

Creates a new shader patcher.

sceGxmShaderPatcherCreateFragmentProgram

Finds or creates a final fragment program.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherCreateFragmentProgram(
    SceGxmShaderPatcher *shaderPatcher,
    SceGxmShaderPatcherId programId,
    SceGxmOutputRegisterFormat outputFormat,
    SceGxmMultisampleMode multisampleMode,
    const SceGxmBlendInfo *blendInfo,
    const SceGxmProgram *vertexProgram,
    SceGxmFragmentProgram **fragmentProgram
);
```

Arguments

[in,out] *shaderPatcher*	A pointer to the shader patcher.
[in] *programId*	The ID for a program registered with this shader patcher.
[in] *outputFormat*	The format for the fragment program `COLOR0` output.
[in] *multisampleMode*	The multisample mode.
[in] *blendInfo*	A pointer to the blend info structure or `NULL`. This structure is copied by the function and therefore does not need to persist after the call.
[in] *vertexProgram*	A pointer to the vertex program or `NULL`. This structure does not need to persist after the call.
[out] *fragmentProgram*	A pointer to storage for a fragment program pointer.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_UNSUPPORTED`	Blending or masking is enabled for an unsupported output register format.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because one or more of the pointers supplied was `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed because some allocated memory did not have the required alignment.
`SCE_GXM_ERROR_OUT_OF_HOST_MEMORY`	The operation failed because a host memory allocation failed.
`SCE_GXM_ERROR_OUT_OF_FRAGMENT_USSE_MEMORY`	The operation failed because a fragment USSE memory allocation failed.

Description

Finds or creates a final fragment program. The fragment program is constructed by appending the shader compiler output with output register conversion and blending code.

When this program is no longer needed, it should be released by calling `sceGxmShaderPatcherReleaseFragmentProgram()`. The caller is responsible for ensuring that the GPU is no longer using this program before it is released.

Blending/masking is only supported for the output register formats SCE_GXM_OUTPUT_REGISTER_FORMAT_UCHAR4 or SCE_GXM_OUTPUT_REGISTER_FORMAT_HALF4. If the *blendInfo* structure has an active blend or non-trivial mask when using unsupported output register format, the error code SCE_GXM_ERROR_UNSUPPORTED will be returned.

The parameter *vertexProgram* is only required to remap texture coordinates. If all vertex programs that will be used with this fragment program write a contiguous range of texture coordinates starting at TEXCOORD0 (or do not write texture coordinates), then the *vertexProgram* parameter may be NULL. Any vertex program that uses texture coordinates that either start at a non-zero binding (such as TEXCOORD1) or form a non-contiguous range must be explicitly linked by passing the program as the *vertexProgram* parameter.

sceGxmShaderPatcherCreateMaskUpdateFragmentProgram

Creates a mask update fragment program.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherCreateMaskUpdateFragmentProgram(
    SceGxmShaderPatcher *shaderPatcher,
    SceGxmFragmentProgram **fragmentProgram
);
```

Arguments

[in,out] *shaderPatcher*	A pointer to the shader patcher.
[out] *fragmentProgram*	A pointer to the storage for a fragment program pointer.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because one or more of the pointers supplied was `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed because an allocation did not have the required alignment.
`SCE_GXM_ERROR_OUT_OF_HOST_MEMORY`	The operation failed because a host allocation failed.

Description

Creates a mask update fragment program. If the shader patcher has already created a mask update fragment program, then the existing program is used after its reference count has been incremented.

When this program is no longer needed, it should be released by calling `sceGxmShaderPatcherReleaseFragmentProgram()`. The caller is responsible for ensuring that the GPU is no longer using this program before it is released.

The purpose of a mask update fragment is to clear or set the mask bit that is part of a depth/stencil surface of format `SCE_GXM_DEPTH_STENCIL_FORMAT_DF32M` or `SCE_GXM_DEPTH_STENCIL_FORMAT_DF32M_S8`. A mask update fragment program has a pass type of `SCE_GXM_PASS_TYPE_MASK_UPDATE`, does not require any shader resources such as uniform buffers or textures, and will automatically disable the fragment program while it is set on the libgxm context.

When a mask update fragment program is used with a draw call, the hardware behaviour changes as follows:

- The depth test is bypassed. All pixels or samples covered by the geometry are considered to pass the depth test.
- Depth writes do not occur.
- The stencil test is bypassed. All pixels or samples covered by the geometry are considered to pass the stencil test.
- Stencil operations do not occur.

- The meaning of the current stencil function is changed. If the current stencil function has 0 in bit 25, such as SCE_GXM_STENCIL_FUNC_NEVER, the mask bit is cleared for all pixels or samples covered by the geometry. If the current stencil function has 1 in bit 25, such as SCE_GXM_STENCIL_FUNC_ALWAYS, the mask bit is set for all pixels or samples covered by the geometry.

The mask bit can be used to implement a 2D viewport or scissoring. This is achieved by drawing 2D geometry to clear and set the mask bit to the desired shape.

sceGxmShaderPatcherCreateVertexProgram

Finds or creates a final vertex program.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherCreateVertexProgram(
    SceGxmShaderPatcher *shaderPatcher,
    SceGxmShaderPatcherId programId,
    const SceGxmVertexAttribute *attributes,
    uint32_t attributeCount,
    const SceGxmVertexStream *streams,
    uint32_t streamCount,
    SceGxmVertexProgram **vertexProgram
);
```

Arguments

[in,out] *shaderPatcher*	A pointer to the shader patcher.
[in] *programId*	The ID for a program registered with this shader patcher.
[in] *attributes*	A pointer to the vertex attributes. These structures are copied by the function and therefore do not need to persist after the call.
[in] *attributeCount*	The number of attributes.
[in] *streams*	A pointer to the vertex streams. These structures are copied by the function and therefore do not need to persist after the call.
[in] *streamCount*	The number of vertex streams.
[out] *vertexProgram*	A pointer to storage for a vertex program pointer.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because one or more of the pointers supplied was `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_GXM_ERROR_INVALID_ALIGNMENT`	The operation failed because some allocated memory did not have the required alignment.
`SCE_GXM_ERROR_OUT_OF_HOST_MEMORY`	The operation failed because a host memory allocation failed.
`SCE_GXM_ERROR_OUT_OF_VERTEX_USSE_MEMORY`	The operation failed because a vertex USSE memory allocation failed.

Description

Finds or creates a final vertex program. The vertex program is constructed by pairing the offline program with the vertex attribute types and stream configuration.

When this program is no longer needed it should be released by calling `sceGxmShaderPatcherReleaseVertexProgram()`. The caller is responsible for ensuring that the GPU is no longer using this program before it is released.

sceGxmShaderPatcherDestroy

Destroys the shader patcher.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherDestroy(
    SceGxmShaderPatcher *shaderPatcher
);
```

Arguments

[in,out] `shaderPatcher`	The shader patcher to be destroyed.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_PROGRAM_IN_USE`	The operation failed because a program is still in use.

Description

Destroys the shader patcher. All registered programs will be unregistered. It is an error to destroy the shader patcher before all vertex and fragment programs have been released.

sceGxmShaderPatcherForceUnregisterProgram

Unregisters a compiled program with the patcher and forces all vertex or fragment programs for the ID to be released.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherForceUnregisterProgram(
    SceGxmShaderPatcher *shaderPatcher,
    SceGxmShaderPatcherId programId
);
```

Arguments

[in,out] *shaderPatcher*	The shader patcher the ID was registered with.
[in] *programId*	The ID of the program this operation must force to unregister.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because the input parameter was `NULL`.

Description

Unregisters a compiled program with the patcher and forces all vertex or fragment programs for the ID to be released. This function will force all vertex or fragment programs that were created from the ID to be destroyed even if their reference counts are still non-zero. Callers should ensure that no vertex or fragment programs for this ID are currently in use by the GPU. Once this function has completed successfully, vertex or fragment programs for this ID are invalid and should no longer be used.

sceGxmShaderPatcherGetBufferMemAllocated

Gets the total amount of GPU buffer memory currently allocated by the shader patcher.

Definition

```
#include <gxm/shader_patcher.h>
uint32_t sceGxmShaderPatcherGetBufferMemAllocated(
    const SceGxmShaderPatcher *shaderPatcher
);
```

Arguments

[in] *shaderPatcher*	A pointer to the shader patcher.

Return Values

The amount of buffer memory in bytes that is currently allocated.

Description

Gets the total amount of GPU buffer memory currently allocated by the shader patcher.

sceGxmShaderPatcherGetFragmentProgramRef Count

Retrieves the reference count of the fragment program.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherGetFragmentProgramRefCount(
    const SceGxmShaderPatcher *shaderPatcher,
    const SceGxmFragmentProgram *fragmentProgram,
    uint32_t *refCount
);
```

Arguments

[in] *shaderPatcher*	A pointer to the shader patcher.
[in] *fragmentProgram*	A pointer to the fragment program whose reference count should be retrieved.
[out] *refCount*	The storage for the reference count.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because the program was not in the cache.

Description

Retrieves the reference count of the fragment program. The fragment program must have been created by the shader patcher.

sceGxmShaderPatcherGetFragmentUsseMem Allocated

Gets the total amount of fragment USSE memory currently allocated by the shader patcher.

Definition

```
#include <gxm/shader_patcher.h>
uint32_t sceGxmShaderPatcherGetFragmentUsseMemAllocated(
   const SceGxmShaderPatcher *shaderPatcher
);
```

Arguments

[in] *shaderPatcher*	A pointer to the shader patcher.

Return Values

The amount of fragment USSE memory in bytes that is currently allocated.

Description

Gets the total amount of fragment USSE memory currently allocated by the shader patcher.

sceGxmShaderPatcherGetHostMemAllocated

Gets the total amount of host memory currently allocated by the shader patcher.

Definition

```
#include <gxm/shader_patcher.h>
uint32_t sceGxmShaderPatcherGetHostMemAllocated(
    const SceGxmShaderPatcher *shaderPatcher
);
```

Arguments

[in] *shaderPatcher*	A pointer to the shader patcher.

Return Values

The amount of host memory in bytes that is currently allocated.

Description

Gets the total amount of host memory currently allocated by the shader patcher.

sceGxmShaderPatcherGetProgramFromId

Gets the program for the given program ID.

Definition

```
#include <gxm/shader_patcher.h>
const SceGxmProgram *sceGxmShaderPatcherGetProgramFromId(
    SceGxmShaderPatcherId programId
);
```

Arguments

[in] *programId*	The ID to get the program for.

Return Values

The compiled program for the given ID or `NULL` if the *programId* parameter was `NULL`.

Description

Gets the program for the given program ID.

sceGxmShaderPatcherGetUserData

Gets the user data pointer associated with this shader patcher.

Definition

```
#include <gxm/shader_patcher.h>
void *sceGxmShaderPatcherGetUserData(
    const SceGxmShaderPatcher *shaderPatcher
);
```

Arguments

[in] *shaderPatcher*	The shader patcher to query.

Return Values

The user data pointer for this shader patcher.

Description

Gets the user data pointer associated with this shader patcher.

sceGxmShaderPatcherGetVertexProgramRefCount

Retrieves the reference count of the vertex program.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherGetVertexProgramRefCount(
    const SceGxmShaderPatcher *shaderPatcher,
    const SceGxmVertexProgram *vertexProgram,
    uint32_t *refCount
);
```

Arguments

[in] `shaderPatcher`	A pointer to the shader patcher.
[in] `vertexProgram`	A pointer to the vertex program whose reference count should be retrieved.
[out] `refCount`	The storage for the reference count.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because one or more of the pointers supplied was `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because the program was not in the cache.

Description

Retrieves the reference count of the vertex program. The vertex program must have been created by the shader patcher.

sceGxmShaderPatcherGetVertexUsseMemAllocated

Gets the total amount of vertex USSE memory currently allocated by the shader patcher.

Definition

```
#include <gxm/shader_patcher.h>
uint32_t sceGxmShaderPatcherGetVertexUsseMemAllocated(
    const SceGxmShaderPatcher *shaderPatcher
);
```

Arguments

[in] *shaderPatcher*	A pointer to the shader patcher.

Return Values

The amount of vertex USSE memory in bytes that is currently allocated.

Description

Gets the total amount of vertex USSE memory currently allocated by the shader patcher.

sceGxmShaderPatcherRegisterProgram

Registers a compiled program to the shader patcher returning an ID.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherRegisterProgram(
    SceGxmShaderPatcher *shaderPatcher,
    const SceGxmProgram *programHeader,
    SceGxmShaderPatcherId *programId
);
```

Arguments

[in,out] *shaderPatcher*	The shader patcher to register with.
[in] *programHeader*	The compiled program header to register.
[out] *programId*	Storage for the program ID.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because the input parameter was NULL.
SCE_GXM_ERROR_INVALID_ALIGNMENT	The operation failed because some allocated memory did not have the required alignment.
SCE_GXM_ERROR_OUT_OF_HOST_MEMORY	The operation failed because a host memory allocation failed.
SCE_GXM_ERROR_OUT_OF_BUFFER_MEMORY	The operation failed because a buffer memory allocation failed.

Description

Registers a compiled program to the shader patcher returning an ID. The caller should take care to only create one ID per compiled program to avoid duplication.

Final vertex and fragment programs may be created from the ID by calling `sceGxmShaderPatcherCreateVertexProgram()` or `sceGxmShaderPatcherCreateFragmentProgram()`.

The compiled program must persist in memory until all vertex or fragment programs have been released and the ID has been unregistered using `sceGxmShaderPatcherUnregisterProgram()`.

sceGxmShaderPatcherReleaseFragmentProgram

Decreases the reference count of the fragment program.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherReleaseFragmentProgram(
    SceGxmShaderPatcher *shaderPatcher,
    SceGxmFragmentProgram *fragmentProgram
);
```

Arguments

[in,out] *shaderPatcher*	A pointer to the shader patcher.
[in,out] *fragmentProgram*	A pointer to the fragment program whose reference count is to be decremented.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed because one or more of the pointers supplied was `NULL`.
`SCE_GXM_ERROR_INVALID_VALUE`	The operation failed because the program was not in the cache.

Description

Decreases the reference count of the fragment program. The fragment program must have been created by the shader patcher. If the reference count of the program hits zero the program is destroyed. Therefore callers should ensure that the GPU has finished using this program before calling this function.

sceGxmShaderPatcherReleaseVertexProgram

Decreases the reference count of the vertex program.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherReleaseVertexProgram(
    SceGxmShaderPatcher *shaderPatcher,
    SceGxmVertexProgram *vertexProgram
);
```

Arguments

[in,out] *shaderPatcher*	A pointer to the shader patcher.
[in,out] *vertexProgram*	A pointer to the vertex program whose reference count is to be decremented.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because one or more of the pointers supplied was NULL.
SCE_GXM_ERROR_INVALID_VALUE	The operation failed because the program was not in the cache.

Description

Decreases the reference count of the vertex program. The vertex program must have been created by the shader patcher. If the reference count of the program hits zero the program is destroyed. Therefore callers should ensure that the GPU has finished using this program before calling this function.

sceGxmShaderPatcherSetUserData

Sets a new user data pointer for this shader patcher.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherSetUserData(
    SceGxmShaderPatcher *shaderPatcher,
    void *userData
);
```

Arguments

[in,out] *shaderPatcher*	The shader patcher to update.
[in] *userData*	The new user data pointer.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because the input parameter was NULL.

Description

Sets a new user data pointer for this shader patcher. The user data pointer is passed to all memory allocation and deallocation callbacks.

sceGxmShaderPatcherUnregisterProgram

Unregisters a compiled program with the patcher.

Definition

```
#include <gxm/shader_patcher.h>
SceGxmErrorCode sceGxmShaderPatcherUnregisterProgram(
    SceGxmShaderPatcher *shaderPatcher,
    SceGxmShaderPatcherId programId
);
```

Arguments

[in,out] *shaderPatcher*	The shader patcher the ID was registered with.
[in] *programId*	The ID to unregister.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_GXM_ERROR_INVALID_POINTER	The operation failed because the input parameter was NULL.
SCE_GXM_ERROR_PROGRAM_IN_USE	The operation failed because the program is still in use.

Description

Unregisters a compiled program with the patcher. This function may only be called after all vertex or fragment programs that were created from the ID have been released.

Callback Functions

SceGxmShaderPatcherBufferAllocCallback

The shader patcher buffer memory allocation callback.

Definition

```
#include <gxm/shader_patcher.h>
typedef void * (*SceGxmShaderPatcherBufferAllocCallback)(
    void *userData,
    uint32_t size
);
```

Arguments

userData	A pointer to user data for optional use by the callback.
size	The size in bytes of memory to allocate.

Return Values

A pointer to the allocated memory, or `NULL` if none can be allocated. The pointer must be aligned to at least 4 bytes.

Description

The shader patcher buffer memory allocation callback. This callback is called to allocate memory for buffers required by vertex or fragment programs, and is called when the program is registered using `sceGxmShaderPatcherRegisterProgram()`. These buffers can be used for:

- Constant literals loaded from memory by the USSE. Usually the shader compiler will be able to remap literals to secondary attribute registers, but if pressure is high enough then literals must sometimes be loaded from memory directly.
- Spilled values or indexable temps. If the shader compiler needs to spill temp registers to memory then this buffer needs to be allocated.

Any buffers allocated using this callback will be shared by all vertex or fragment programs created from the registered program. The memory allocated by this callback must have been mapped with both `SCE_GXM_MEMORY_ATTRIB_READ` and `SCE_GXM_MEMORY_ATTRIB_WRITE` access using `sceGxmMapMemory()`.

SceGxmShaderPatcherBufferFreeCallback

The shader patcher buffer memory deallocation callback.

Definition

```
#include <gxm/shader_patcher.h>
typedef void (*SceGxmShaderPatcherBufferFreeCallback)(
   void *userData,
   void *mem
);
```

Arguments

userData	A pointer to user data for optional use by the callback.
mem	A pointer to the memory to be freed.

Return Values

None

Description

The shader patcher buffer memory deallocation callback. This callback is called to deallocate memory for buffers required by vertex or fragment programs. It is called when the program is unregistered using `sceGxmShaderPatcherUnregisterProgram()`.

SceGxmShaderPatcherHostAllocCallback

The shader patcher host memory allocation callback.

Definition

```
#include <gxm/shader_patcher.h>
typedef void * (*SceGxmShaderPatcherHostAllocCallback)(
    void *userData,
    uint32_t size
);
```

Arguments

userData	A pointer to user data for optional use by the callback.
size	The size in bytes of the memory to allocate.

Return Values

A pointer to the allocated memory, or `NULL` if none can be allocated. The pointer must be aligned to at least 4 bytes.

Description

The shader patcher host memory allocation callback. This should return standard cached CPU memory for shader patcher data structures, such as that returned from libc malloc.

SceGxmShaderPatcherHostFreeCallback

The shader patcher host memory free callback.

Definition

```
#include <gxm/shader_patcher.h>
typedef void (*SceGxmShaderPatcherHostFreeCallback)(
    void *userData,
    void *mem
);
```

Arguments

userData	A pointer to user data for optional use by the callback.
mem	A pointer to the memory to be freed.

Return Values

None

Description

The shader patcher host memory free callback.

SceGxmShaderPatcherUsseAllocCallback

The shader patcher vertex or fragment USSE allocation callback.

Definition

```
#include <gxm/shader_patcher.h>
typedef void * (*SceGxmShaderPatcherUsseAllocCallback)(
    void *userData,
    uint32_t size,
    uint32_t *usseOffset
);
```

Arguments

userData	A pointer to user data for optional use by the callback.
size	The size in bytes of the memory to allocate.
usseOffset	The storage for the USSE offset of the allocation.

Return Values

A pointer to the allocated memory, or `NULL` if none can be allocated. The pointer must be aligned to at least `SCE_GXM_USSE_ALIGNMENT` bytes.

Description

The shader patcher vertex or fragment USSE allocation callback. This callback is called to allocate memory for USSE code required by the vertex or fragment programs. It is called when a new program needs to be created during `sceGxmShaderPatcherCreateVertexProgram()` or `sceGxmShaderPatcherCreateFragmentProgram()`.

Any buffers allocated using this callback will be shared by all vertex or fragment programs created from the registered program. The memory allocated by this callback must have been mapped as vertex or fragment USSE code, depending on whether the callback is the vertex USSE callback or fragment USSE callback respectively.

Note that it is possible to map memory as both vertex and fragment USSE code, and the shader patcher can be configured to use a shared heap for vertex and fragment USSE code. See the documentation for `SceGxmShaderPatcherParams` for more details.

SceGxmShaderPatcherUsseFreeCallback

The shader patcher vertex or fragment USSE deallocation callback.

Definition

```
#include <gxm/shader_patcher.h>
typedef void (*SceGxmShaderPatcherUsseFreeCallback)(
    void *userData,
    void *mem
);
```

Arguments

userData	A pointer to user data for optional use by the callback.
mem	A pointer to the memory to be freed.

Return Values

None

Description

The shader patcher vertex or fragment USSE deallocation callback. This callback is called to deallocate memory for USSE code required by the vertex or fragment programs, and is called when a program hits a reference count of zero and is destroyed during `sceGxmShaderPatcherReleaseVertexProgram()` or `sceGxmShaderPatcherReleaseFragmentProgram()`.