

PlayStation®Edge Geometry Library Reference

Table of Contents

Preface.....	4
About This Document.....	5
Data Types for Runtime.....	6
EdgeGeomSpuConfigInfo	7
EdgeGeomPpuConfigInfo	11
EdgeGeomBlendShapeInfo	14
EdgeGeomViewportInfo	15
EdgeGeomLocalToWorldMatrix	16
EdgeGeomSharedBufferInfo	17
EdgeGeomRingBufferInfo	18
EdgeGeomOutputBufferInfo.....	20
EdgeGeomAllocationInfo	22
EdgeGeomLocation	23
EdgeGeomCullingResults	24
EdgeGeomVertexStreamDescription	25
EdgeGeomCustomVertexFormatInfo	27
EdgeGeomCustomTransformVertexesForCullCallbackInfo.....	28
EdgeGeomSpuContext	29
Functions for SPU Runtime	30
edgeGeomInitialize	31
edgeGeomValidateBufferOrder	33
edgeGeomGetSpuConfigInfo	35
edgeGeomGetViewportInfo.....	36
edgeGeomGetLocalToWorldMatrix	37
edgeGeomGetUniformTable	38
edgeGeomGetUniformTableByAttribute.....	39
edgeGeomGetUniformTables.....	40
edgeGeomGetUniformTableCount.....	41
edgeGeomAssignUniformTable	42
edgeGeomUnassignUniformTable	43
edgeGeomGetPositionUniformTable.....	44
edgeGeomSetPositionUniformTable	45
edgeGeomGetTransformUniformTable	46
edgeGeomGetNormalUniformTable.....	47
edgeGeomSetNormalUniformTable	48
edgeGeomGetTangentUniformTable.....	49
edgeGeomSetTangentUniformTable	50
edgeGeomGetBinormalUniformTable	51
edgeGeomSetBinormalUniformTable	52
edgeGeomGetIndexTable	53
edgeGeomGetIndexCount	54
edgeGeomSetIndexCount.....	55
edgeGeomGetVertexCount.....	56
edgeGeomSetVertexCount	57

edgeGeomGetFreePtr	58
edgeGeomSetFreePtr	59
edgeGeomIsAllocatedFromRingBuffer	60
edgeGeomGetOutputVertexStride	61
edgeGeomDecompressVertexes	62
edgeGeomDecompressIndexes	66
edgeGeomProcessBlendShapes	67
edgeGeomNormalizeUniformTable	68
edgeGeomSkinVertexes	69
edgeGeomTransformVertexes	70
edgeGeomCullOccludedTriangles	71
edgeGeomCullTriangles	72
edgeGeomCalculateDefaultOutputSize	73
edgeGeomAllocateOutputSpace	74
edgeGeomUseOutputSpace	76
edgeGeomOutputIndexes	77
edgeGeomCompressVertexes	78
edgeGeomOutputVertexes	80
edgeGeomBeginCommandBufferHole	81
edgeGeomSetVertexDataArrays	82
edgeGeomEndCommandBufferHole	83
Callback Functions	84
EdgeGeomGetOutputVertexStrideCallback	85
EdgeGeomDecompressVertexStreamCallback	86
EdgeGeomBlendVertexStreamCallback	87
EdgeGeomCompressVertexStreamCallback	88
EdgeGeomSetVertexDataArraysCallback	89
EdgeGeomTransformVertexesForCullCallback	90
Constants	91
Triangle Culling Mode	92
Skinning Mode	93
Index Data Type	94
Skinning Matrix Format	95
Geometry Configuration Flags	96
Output Mode	97

Preface

About This Document

Purpose

This document provides an API reference for the geometry component of the Edge library. Use this component to efficiently perform skinning, coordinate transformation, and triangle culling through the SPUs.

Audience and Prerequisites

This document was written for PlayStation®3 developers who want to write high-performance applications for the PlayStation®3. It is assumed that such developers have familiarity with the following:

- C and C++
- PlayStation®3 hardware
- SCE standard library functions

Related Documentation

In combination with this reference, the following documents provide complete usage and reference information about the Edge library:

- *PlayStation®Edge Library Overview*
- *PlayStation®Edge Geometry Library: Quick Start*
- *PlayStation®Edge Geometry Library for Offline Tool: Reference*
- *PlayStation®Edge Animation Library Reference*
- *PlayStation®Edge Animation Library for Offline Tool: Reference*
- *PlayStation®Edge Zlib Library Reference*
- *PlayStation®Edge LZMA Library Reference*
- *PlayStation®Edge LZO Library Reference*
- *PlayStation®Edge DXT Library Reference*
- *PlayStation®Edge Post Library Reference*

Typographic Conventions

This document uses the following typographic conventions:

Convention	Meaning
<code>fixed-width font</code>	Indicates programming code and literals, such as processing instructions, register names, data types, events, and file names. Also indicates function, structure, and macro names.
<code>fixed-width font + bold</code>	In structure/function definitions only, indicates names of structures and functions.
<i>fixed-width font + italics</i>	Indicates arguments, parameters, and variables.
blue + underlined text	Indicates a hyperlink (blue displays in color printers or online only).

Data Types for Runtime

EdgeGeomSpuConfigInfo

Structure that stores process configuration for SPU.

Definition

```
#include <edgegeom_structs.h>
struct __attribute__((__aligned__(16))) EdgeGeomSpuConfigInfo{
    uint8_t flagsAndUniformTableCount;
    uint8_t commandBufferHoleSize;
    uint8_t inputVertexFormatId;
    uint8_t secondaryInputVertexFormatId;
    uint8_t outputVertexFormatId;
    uint8_t unused;
    uint8_t indexesFlavorAndSkinningFlavor;
    uint8_t skinningMatrixFormat;
    uint16_t numVertexes;
    uint16_t numIndexes;
    uint32_t indexesOffset;
};
```

Members

<i>flagsAndUniformTableCount</i>	Processing flags are stored in the upper 4 bits, and the number of uniform tables - 1 is stored in the lower 4 bits. See " Geometry Configuration Flags ".
<i>commandBufferHoleSize</i>	Maximum size of command buffer hole needed for output (in qword).
<i>inputVertexFormatId</i>	Format ID of the primary input vertex stream, from the <code>EDGE_GEOM_SPU_VERTEX_FORMAT_*</code> enum. A value of <code>0xFF</code> indicates a custom (non-built-in) format.
<i>secondaryInputVertexFormatId</i>	Format ID of the secondary input vertex stream, from the <code>EDGE_GEOM_SPU_VERTEX_FORMAT_*</code> enum. If the segment's secondary vertex stream size is 0, this field can take any value. A value of <code>0xFF</code> indicates a custom (non-built-in) format.
<i>outputVertexFormatId</i>	Format ID of the output vertex stream, from the <code>EDGE_GEOM_RSX_VERTEX_FORMAT_*</code> enum. A value of <code>0xFF</code> indicates a custom (non-built-in) format.
<i>unused</i>	Unused data field.
<i>indexesFlavorAndSkinningFlavor</i>	Flavors of the input index stream and skinning calculation. See " Skinning Mode " and " Index Data Type ".
<i>skinningMatrixFormat</i>	Format ID of the skinning matrices, if any, from the <code>EDGE_GEOM_MATRIX_*</code> enum. If skinning is not performed, this field will be ignored.
<i>numVertexes</i>	Number of vertexes.
<i>numIndexes</i>	Number of indexes. After calling edgeGeomInitialize() , the user should access the number of indexes by calling edgeGeomGetIndexCount() .
<i>indexesOffset</i>	Most users should set this field to <code>0xFFFFFFFF</code> . See below for a full explanation.

Description

The `EdgeGeomSpuConfigInfo` data structure is a static structure of configuration information. This configuration information is required to initialize an Edge Geometry process. For presegmented geometry, this structure should be generated by the offline tools.

The lower four bits of `flagsAndUniformTableCount` indicate the number of uniform tables that `libedgegeomtool` needs to allocate during the process. This count is equal to the number of vertex attributes - 1 (to allow a count of 1 - 16 to fit in 4 bits). However, several Edge operations (including skinning, culling, occlusion and blend shapes) require an additional uniform table for scratch space. If performing any of these operations, the uniform table count must be increased by one. The upper four bits are reserved for processing flags; the following flags are currently defined:

Macro Name	Value	Usage
<code>EDGE_GEOM_FLAG_STATIC_GEOMETRY_FAST_PATH</code>	<code>0x10</code>	If set, the following Edge functions will return immediately: <code>edgeGeomSkinVertexes()</code> , <code>edgeGeomProcessBlendShapes()</code> , <code>edgeGeomCompressVertexes()</code> , <code>edgeGeomOutputVertexes()</code> .
<code>EDGE_GEOM_FLAG_INCLUDES_EXTRA_UNIFORM_TABLE</code>	<code>0x80</code>	Must be set if this segment uses skinning, culling, occlusion, or blend shapes because these operations require an additional uniform table for intermediate data processing.

`commandBufferHoleSize` indicates the maximum size of command buffer hole needed for outputting the graphic commands (in qwords). This should be calculated in the tools using `edgeGeomGetCommandBufferHoleSize()`.

The various format ID fields are indexes into the following lists of built-in vertex formats. A custom format has an ID of -1 (`0xFF`). In this case, processing is handled by a user-provided callback or, if present, a stream description structure generated in the offline tools and uploaded as part of the input data to the SPU.

The primary and secondary vertex format IDs reference the following list of built-in formats:

Input Layout 0: `EDGE_GEOM_SPU_VERTEX_FORMAT_F32c3`

Byte offset	Format	# of Components	Attribute ID
0	F32	3	<code>EDGE_GEOM_ATTRIBUTE_USAGE_POSITION</code>

Input Layout 1: `EDGE_GEOM_SPU_VERTEX_FORMAT_F32c3_X11Y11Z10N_X11Y11Z10N`

Byte offset	Format	# of Components	Attribute ID
0	F32	3	<code>EDGE_GEOM_ATTRIBUTE_USAGE_POSITION</code>
12	X11Y11Z10N	3	<code>EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL</code>
16	X11Y11Z10N	3	<code>EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT</code>

Input Layout 2: `EDGE_GEOM_SPU_VERTEX_FORMAT_F32c3_X11Y11Z10N_I16Nc4`

Byte offset	Format	# of Components	Attribute ID
0	F32	3	<code>EDGE_GEOM_ATTRIBUTE_USAGE_POSITION</code>
12	X11Y11Z10N	3	<code>EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL</code>
16	I16N	4	<code>EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT</code>

Input Layout 3: `EDGE_GEOM_SPU_VERTEX_FORMAT_F32c3_X11Y11Z10N_X11Y11Z10N_X11Y11Z10N`

Byte offset	Format	# of Components	Attribute ID
0	F32	3	<code>EDGE_GEOM_ATTRIBUTE_USAGE_POSITION</code>
12	X11Y11Z10N	3	<code>EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL</code>

Byte offset	Format	# of Components	Attribute ID
16	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT
20	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_BINORMAL

Input Layout 4: EDGE_GEOM_SPU_VERTEX_FORMAT_FIXED_UNITVEC_UNITVEC

Byte offset	Format	# of Components	Attribute ID
0	SW FIXED	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION
6	SW UNIT VEC	4	EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL
9	SW UNIT VEC	4	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT

Input Layout 5: EDGE_GEOM_SPU_VERTEX_FORMAT_FIXED_UNITVEC_UNITVEC_UNITVEC

Byte offset	Format	# of Components	Attribute ID
0	SW FIXED	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION
6	SW UNIT VEC	4	EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL
9	SW UNIT VEC	4	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT
12	SW UNIT VEC	4	EDGE_GEOM_ATTRIBUTE_USAGE_BINORMAL

Input Layout 6: EDGE_GEOM_SPU_VERTEX_FORMAT_EMPTY

Byte offset	Format	# of Components	Attribute ID
No attributes contained			

The output format ID references the following list of built-in formats:

Output Layout 0: EDGE_GEOM_RSX_VERTEX_FORMAT_F32c3

Byte offset	Format	# of Components	Attribute ID
0	F32	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION

Output Layout 1: EDGE_GEOM_RSX_VERTEX_FORMAT_F32c3_X11Y11Z10N_X11Y11Z10N

Byte offset	Format	# of Components	Attribute ID
0	F32	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION
12	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL
16	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT

Output Layout 2: EDGE_GEOM_RSX_VERTEX_FORMAT_F32c3_X11Y11Z10N_I16Nc4

Byte offset	Format	# of Components	Attribute ID
0	F32	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION
12	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL
16	I16N	4	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT

Output Layout 3: EDGE_GEOM_RSX_VERTEX_FORMAT_F32c3_X11Y11Z10N_X11Y11Z10N_X11Y11Z10N

Byte offset	Format	# of Components	Attribute ID
0	F32	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION
12	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL
16	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT
20	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_BINORMAL

Output Layout 4: EDGE_GEOM_RSX_VERTEX_FORMAT_EMPTY

Byte offset	Format	# of Components	Attribute ID
No attributes contained			

The upper 4 bits of the *indexesFlavorAndSkinningFlavor* indicate the type of index data stream whereas the lower 4 bits indicate which skinning process should be performed on this vertex data chunk.

$$indexesFlavorAndSkinningFlavor = ((IndexFlavor \& 0xF) \ll 4) | (SkinningFlavor \& 0xF)$$

For possible values for index and skinning flavor, see “[Index Data Type](#)” and “[Skinning Mode](#)”.

skinningMatrixFormat specifies the format of skinning matrices, if any. For possible values, see “[Skinning Matrix Format](#)”.

numVertexes stores the number of vertexes and *numIndexes* stores the number of indexes for this particular job. Both are usually generated by offline tools. After calling [edgeGeomInitialize\(\)](#), the user should access the number of indexes by calling [edgeGeomGetIndexCount\(\)](#).

indexesOffset stores the RSX™ I/O offset to the index data. This information can be used when there is no change to the original input index data, that is, when no index data compression and no culling have been performed, and the original index buffer can be rendered in its unmodified state. You can use the index buffers generated by Edge (by far the more common use case), by setting the *indexesOffset* field to -1 (0xFFFFFFFF).

See Also

[edgeGeomInitialize](#)

EdgeGeomPpuConfigInfo

Structure that stores configuration info for each segment to set up.

Definition

```
#include <edgegeom_structs.h>
struct __attribute__((aligned(16))) EdgeGeomPpuConfigInfo
{
    EdgeGeomSpuConfigInfo spuConfigInfo;
    voidptr32_t indexes;
    uint16_t indexesSizes[2];
    voidptr32_t spuVertexes[2];
    uint16_t spuVertexesSizes[6];
    voidptr32_t rsxOnlyVertexes;
    uint32_t rsxOnlyVertexesSize;
    uint16_t skinMatricesByteOffsets[2];
    uint16_t skinMatricesSizes[2];
    uint16_t skinIndexesAndWeightsSizes[2];
    uint16_t shapeSize;
    uint16_t shapeFixedOffsetSize;
    voidptr32_t skinIndexesAndWeights;
    uint32_t ioBufferSize;
    uint32_t scratchSize;
    uint32_t numBlendShapes;
    uint16_t *blendShapeSizes;
    uint32_t *blendShapes;
    uint32_t fixedOffsetsSize[2];
    voidptr32_t fixedOffsets[2];
    voidptr32_t spuInputStreamDescs[2];
    voidptr32_t spuOutputStreamDesc;
    voidptr32_t rsxOnlyStreamDesc;
    uint16_t spuInputStreamDescSizes[2];
    uint16_t spuOutputStreamDescSize;
    uint16_t rsxOnlyStreamDescSize;
};
```

Members

<i>spuConfigInfo</i>	The SPU configuration info.
<i>indexes</i>	A pointer to the index data for this segment.
<i>indexesSizes[2]</i>	The DMA sizes for the index data of this segment.
<i>spuVertexes[2]</i>	Pointers to the two SPU input streams of vertex data for this segment.
<i>spuVertexesSizes[6]</i>	The DMA sizes for the two streams of SPU input vertex data for this segment.
	The first three elements in the array relate to the first stream and the second three elements in the array relate to the second stream.
<i>rsxOnlyVertexes</i>	Pointer to this segment's RSX™-only vertex stream. These attributes should be sent directly to the RSX™ whereas the SPU input streams are sent through the SPUs.
<i>rsxOnlyVertexesSize</i>	Size (in bytes) of this segment's RSX™-only vertex stream.
<i>skinMatricesByteOffsets[2]</i>	The byte offsets from the base matrix pointer of the two sections of matrices to upload.
<i>skinMatricesSizes[2]</i>	The sizes of the two sections of matrices to upload.
<i>skinIndexesAndWeightsSizes[2]</i>	The DMA sizes of the indexes and weights data.
<i>shapeSize</i>	The DMA size of the blend shape data.

<i>shapeFixedOffsetsSize</i>	The DMA size of the fixed point compression offsets.
<i>skinIndexesAndWeights</i>	A pointer to the interlaced skinning indexes and weights data.
<i>ioBufferSize</i>	This SPURS job's required I/O Buffer size.
<i>scratchSize</i>	This SPURS job's required scratch size in quadwords
<i>numBlendShapes</i>	Number of blend shapes defined for this segment. The <i>blendShapes</i> and <i>blendShapeSizes</i> member arrays have this many elements.
<i>blendShapeSizes</i>	A pointer to an array containing the sizes (in bytes) of each blend shape buffer for this segment. This array has <i>numBlendShapes</i> elements.
<i>blendShapes</i>	An array of pointers to each blend shape buffer for this segment. Unused entries will be NULL. This array has <i>numBlendShapes</i> elements.
<i>fixedOffsetsSize[2]</i>	The DMA size of the fixed-point offset data for each vertex stream.
<i>fixedOffsets[2]</i>	A pointer to the fixed-point offset data for each vertex stream, used to bias fixed-point attributes back to their original range during decompression.
<i>spuInputStreamDescs</i>	Pointers to the optional stream description structures for the two SPU input vertex stream. Each entry will only be non-NULL if the corresponding stream is using a custom vertex format.
<i>spuOutputStreamDesc</i>	A pointer to the SPU output stream description structure for the secondary input vertex stream. Will only be non-NULL if the SPU output stream is using a custom vertex format.
<i>rsxOnlyStreamDesc</i>	A pointer to the optional stream description structure for the RSX™-only vertex stream. Will only be non-NULL if the <i>rsxVertexesSize</i> is non-zero.
<i>spuInputStreamDescSizes</i>	Sizes (in bytes) of the SPU input stream descriptions.
<i>spuOutputStreamDescSize</i>	Size (in bytes) of the SPU output stream description.
<i>rsxOnlyStreamDescSize</i>	Size (in bytes) of the RSX™-only stream description.

Description

This structure is used to store information for a runtime PPU process to initialize SPU jobs and should be generated by offline tools.

The structure represents two different vertex data streams and two different skinning matrices data streams. It also includes two different fixed point offset tables.

For example, the third DMA tag for the second vertex data stream could be:

```
dmaList[SECOND_VERT_ST_TAG_3] =
    (ppuInfo->spuVertexes[1]+ ppuInfo->spuVertexesSizes[3]+
    ppuInfo->spuVertexesSizes[4]) | (ppuInfo->spuVertexesSizes[5] << 32);
```

Note that there are only one Indexes data stream and one *skinIndexesAndWeight* data stream in this structure even though both of them have two entries to describe the size of buffer. For example, the second DMA tag for the index data stream could be:

```
dmaList[INDEX_ST_TAG_2] =
    (ppuInfo->indexes + ppuInfo->indexesSizes [0]) |
    (ppuInfo->indexesSizes [1] << 32);
```

Note: [EdgeGeomPpuConfigInfo](#) is not used directly by the Edge Geometry runtime. It is provided for convenience to encapsulate all of the data that could possibly be needed to create an Edge Geometry job, and is used by the Edge sample programs for this purpose. Users are encouraged to create an Edge transport structure that best serves their needs.

See Also

[EdgeGeomSpuConfigInfo](#)

EdgeGeomBlendShapeInfo

Structure that stores the information of a Blend Shape.

Definition

```
#include <edgegeom_structs.h>
struct __attribute__((__aligned__(16))) EdgeGeomBlendShapeInfo
{
    uint64_t dmaTag;
    float alpha;
    uint32_t padding;
};
```

Members

<i>dmaTag</i>	A DMA tag that is set up to DMA the blend shape data
<i>alpha</i>	The percent to blend in this blend shape
<i>padding</i>	Structure padding (currently unused)

Description

The `EdgeGeomBlendShapeInfo` structure contains the information of a shape data that is needed by the runtime SPU process to perform the blend shape operation. If there is no blend shape to apply, the structure is not necessary.

The field *dmaTag* is used to issue a List DMA GET by the runtime SPU process to collect the blend shape data. This can be constructed as follows:

```
shapeInfo.dmaTag = ppuInfo->blendShapes[TARGET_SHAPE] |
(ppuInfo->blendShapeSizes[TARGET_SHAPE]<<32)
```

See Also

[EdgeGeomPpuConfigInfo](#), [edgeGeomProcessBlendShapes](#)

EdgeGeomViewportInfo

Structure that passes the Viewport setting for a triangle culling operation.

Definition

```
#include <edgegeom_structs.h>
struct __attribute__((__aligned__(16))) EdgeGeomViewportInfo
{
    uint16_t scissorArea[4];
    float depthRange[2];
    float viewProjectionMatrix[16];
    float viewportScales[4];
    float viewportOffsets[4];
    uint8_t sampleFlavor;
    uint8_t pad[3];
};
```

Members

<i>scissorArea[4]</i>	The parameters that are passed to <code>cellGcmSetScissor()</code> .
<i>depthRange[2]</i>	The minimum and maximum parameters that are passed to <code>cellGcmSetViewport()</code> .
<i>viewProjectionMatrix[16]</i>	A matrix that transforms the vertex data from world space to device coordinates.
<i>viewportScales[4]</i>	The scale values that are passed to <code>cellGcmSetViewport()</code> .
<i>viewportOffsets[4]</i>	The offset values that are passed to <code>cellGcmSetViewport()</code> .
<i>sampleFlavor</i>	The type of MSAA to use: CELL_GCM_SURFACE_CENTER_1 CELL_GCM_SURFACE_DIAGONAL_CENTERED_2 CELL_GCM_SURFACE_SQUARE_CENTERED_4 CELL_GCM_SURFACE_SQUARE_ROTATED_4 This should be set to the same value that your target surface has. See the <i>libgcm Reference</i> for further information about MSAA.
<i>pad[3]</i>	Padding for byte alignment.

Description

The `EdgeGeomViewportInfo` data structure contains information necessary to perform triangle culling. If no triangle culling is being performed via inspection of the *cullingFlavor*, which should be set as part of SPU Input User Data, then this structure is not necessary.

EdgeGeomLocalToWorldMatrix

Structure that passes the Local-to-World Matrix to the SPU to perform coordinate transformation.

Definition

```
#include <edgegeom_structs.h>
struct __attribute__((__aligned__(16))) EdgeGeomLocalToWorldMatrix
{
    float matrixData[12];
};
```

Members

matrixData A 4x3 matrix to do the local to world transformation for a certain geometry segment

Description

The local-to-world matrix, typically referred to as the model matrix, is a 4x3 matrix that transforms the object from its local space into world space. This matrix relieves that PPU from some computation because the PPU does not have to compute the model-view-projection matrix in its entirety and can instead give its parts to the SPU. If triangle culling is not required, then this structure is not necessary.

EdgeGeomSharedBufferInfo

Structure that contains information for shared buffer control used by all SPUs.

Definition

```
#include <edgegeom_structs.h>
struct __attribute__((__aligned__(128))) EdgeGeomSharedBufferInfo
{
    uint32_t startEa;
    uint32_t startOffset;
    uint32_t endEa;
    uint32_t currentEa;
    uint32_t locationId;
    uint32_t failedAllocSize;
    uint32_t pad1[2];
    uint32_t pad2[24];
};
```

Members

<i>startEa</i>	Effective Address of shared buffer start
<i>startOffset</i>	I/O offset of the shared buffer start
<i>endEa</i>	Address of buffer end (start + size)
<i>currentEa</i>	Effective Address of the beginning of buffer free area
<i>locationId</i>	Location of the buffer CELL_GCM_LOCATION_MAIN or CELL_GCM_LOCATION_LOCAL
<i>failedAllocSize</i>	Total size of allocations that could not be fulfilled per frame
<i>pad1[2]</i>	Padding for byte alignment
<i>pad2[24]</i>	Padding for byte alignment

Description

The `EdgeGeomSharedBufferInfo` structure describes a simple output buffer that is shared between all SPUs by using atomic locks. When it is full, subsequent allocations fail. A buffer is considered empty or nonexistent if its *startEa* and *endEa* fields are the same.

This structure is mainly used in double buffer, single buffer and hybrid buffer schemes. For more information about how to set up this structure for each type of buffer scheme, see the “[EdgeGeomOutputBufferInfo](#)” section.

When the SPU fails to allocate free output space from a shared buffer, the value of *failedAllocSize* is incremented by the requested allocation size. Applications can monitor this value to determine whether their buffer is too small, and (if so) how much larger it needs to be.

See Also

[EdgeGeomOutputBufferInfo](#)

EdgeGeomRingBufferInfo

Structure contains information for ring buffer control used by one SPU.

Definition

```
#include <edgegeom_structs.h>
struct __attribute__((__aligned__(16))) EdgeGeomRingBufferInfo
{
    uint32_t startEa;
    uint32_t startOffset;
    uint32_t endEa;
    uint32_t currentEa;
    uint32_t locationId;
    uint32_t rsxLabelEa;
    uint32_t cachedFree;
    uint32_t pad[25];
};
```

Members

<i>startEa</i>	Effective Address of the ring buffer start
<i>startOffset</i>	I/O offset of the buffer start
<i>endEa</i>	Address of buffer end (start + size)
<i>currentEa</i>	Address of the beginning of buffer free area
<i>locationId</i>	The location of the buffer; should be either CELL_GCM_LOCATION_MAIN or CELL_GCM_LOCATION_LOCAL
<i>rsxLabelEa</i>	Effective Address of RSX™ label used for synchronization
<i>cachedFree</i>	Cached free pointer (must be initialized to 0)
<i>pad[25]</i>	Padding for byte alignment

Description

The `EdgeGeomRingBufferInfo` structure describes an effectively infinite output buffer that is synchronized with the RSX™, so that it knows when it is safe to loop back on itself. If an allocation does not fit, the SPU blocks until the RSX™ consumes enough data to free up the required space. A buffer is considered empty or nonexistent if its *startEa* and *endEa* fields are the same.

This structure is mainly used in ring buffer and hybrid buffer schemes. For more information about how to set up this structure for each of the different buffer schemes, see the “[EdgeGeomOutputBufferInfo](#)” section.

The *rsxLabelEa* field provides the address in local memory of the RSX™ label used by the SPU to monitor the RSX™ consumption of the buffer data. Each SPU needs its own dedicated RSX™ label.

Notes

The RSX™ label IDs 0 to 63 are reserved by the OS; the label IDs 64 to 255 might be available. However, in the absence of a centralized RSX™ label-reserving system, there is currently no way to be certain which labels are actually in use by other parts of the same application. Read the *libgcm Overview* document for more detailed information about RSX™ labels.

If an EdgeGeom job uses more than one ring buffer, the command buffer hole size specified in the `SpuConfigInfo` *must be updated* to have an additional 16 bytes for each additional ring buffer (in other words, nothing need be added if only one ring buffer is used), where the commands to write each ring buffer’s RSX™ label will be placed.

See Also

[EdgeGeomOutputBufferInfo](#)

EdgeGeomOutputBufferInfo

Structure that describes the application's output buffering scheme.

Definition

```
#include <edgegeom_structs.h>
struct __attribute__((__aligned__(128))) EdgeGeomOutputBufferInfo
{
    EdgeGeomSharedBufferInfo sharedInfo;
    EdgeGeomRingBufferInfo ringInfo[6];
};
```

Members

<i>sharedInfo</i>	Represents a simple shared buffer. Shared by all SPU.
<i>ringInfo[6]</i>	Represents 6 ring buffers, one per SPU addressable by SPURS.

Description

This structure describes the application's output buffering scheme. It contains two components; one [EdgeGeomSharedBufferInfo](#) object, and an array of six [EdgeGeomRingBufferInfo](#) objects (one for each addressable SPU). A buffer is considered empty or nonexistent if its *startEa* and *endEa* fields are the same.

There are four different buffering schemes supported: double buffer, single buffer, ring buffer, and hybrid buffer.

Pseudo Code for Implementing Double Buffer:

```
static EdgeGeomOutputBufferInfo info;
// Frame 0
memset(&info, 0, sizeof(info));
info.sharedInfo.startEa = (uint32_t)buffer[0];
info.sharedInfo.endEa = (uint32_t)buffer[0] + sizeof(buffer[0]);
info.sharedInfo.currentEa = info.sharedInfo.startEa;
info.sharedInfo.locationId = CELL_GCM_LOCATION_MAIN;
cellGcmAddressToOffset(buffer, &info.sharedInfo.startOffset);
//...
// Frame 1
memset(&info, 0, sizeof(info));
info.sharedInfo.startEa = (uint32_t)buffer[1];
info.sharedInfo.endEa = (uint32_t)buffer[1] + sizeof(buffer[1]);
info.sharedInfo.currentEa = info.sharedInfo.startEa;
info.sharedInfo.locationId = CELL_GCM_LOCATION_MAIN;
cellGcmAddressToOffset(buffer, &info.sharedInfo.startOffset);
```

Pseudo Code for Implementing Single Buffer:

This buffer must be reset every frame before use by the SPU.

```
static EdgeGeomOutputBufferInfo info;
memset(&info, 0, sizeof(info));
info.sharedInfo.startEa = (uint32_t)buffer;
info.sharedInfo.endEa = (uint32_t)buffer + sizeof(buffer);
info.sharedInfo.currentEa = info.sharedInfo.startEa;
info.sharedInfo.locationId = CELL_GCM_LOCATION_MAIN;
cellGcmAddressToOffset(buffer, &info.sharedInfo.startOffset);
```

Pseudo Code for Implementing Ring Buffer:

```
static EdgeGeomOutputBufferInfo info;
memset(&info, 0, sizeof(info));
uint32_t labels = (uint32_t)cellGcmGetLabelAddress(64);
for(uint32_t i=0;i<6;++i)
{
    info.ringInfo[i].startEa = (uint32_t)buffer + i*sizeof(buffer)/6;
    info.ringInfo[i].endEa = info.ringInfo[i].startEa + sizeof(buffer)/6;
    info.ringInfo[i].currentEa = info.ringInfo[i].startEa;
    info.ringInfo[i].locationId = CELL_GCM_LOCATION_MAIN;
    info.ringInfo[i].rsxLabelEa = labels + i*16;
    info.ringInfo[i].cachedFree = 0;
    cellGcmAddressToOffset(
        (void*)info.ringInfo[i].startEa,
        &info.ringInfo[i].startOffset);
    (uint32_t*)labels[i*4] = info.ringInfo[i].endEa;
}
```

Pseudo Code for Implementing Hybrid Buffer:

Implementing hybrid buffers is as simple as setting both a single buffer and a ring buffer up. The runtime intelligently knows what to do given the presence of any structures that are set up. There is no wrong way to use the system. This scheme is recommended.

```
static EdgeGeomOutputBufferInfo info;
memset(&info, 0, sizeof(info));
info.sharedInfo.startEa = (uint32_t)sbuffer;
info.sharedInfo.endEa = (uint32_t)sbuffer + sizeof(sbuffer);
info.sharedInfo.currentEa = info.sharedInfo.startEa;
info.sharedInfo.locationId = CELL_GCM_LOCATION_MAIN;
cellGcmAddressToOffset(sbuffer, &info.sharedInfo.startOffset);
uint32_t labels = (uint32_t)cellGcmGetLabelAddress(64);
for(uint32_t i=0;i<6;++i)
{
    info.ringInfo[i].startEa = (uint32_t)rbuffer + i*sizeof(rbuffer)/6;
    info.ringInfo[i].endEa = info.ringInfo[i].startEa + sizeof(rbuffer)/6;
    info.ringInfo[i].currentEa = info.ringInfo[i].startEa;
    info.ringInfo[i].locationId = CELL_GCM_LOCATION_MAIN;
    info.ringInfo[i].rsxLabelEa = labels + i*16;
    info.ringInfo[i].cachedFree = 0;
    cellGcmAddressToOffset(
        (void*)info.ringInfo[i].startEa,
        &info.ringInfo[i].startOffset);
    (uint32_t*)labels[i*4] = info.ringInfo[i].endEa;
}
```

See Also

[edgeGeomAllocateOutputSpace](#), [EdgeGeomAllocationInfo](#)

EdgeGeomAllocationInfo

Structure that describes the allocation reserved by [edgeGeomAllocateOutputSpace\(\)](#).

Definition

```
#include <edgegeom_structs.h>
struct __attribute__((__aligned__(16))) EdgeGeomAllocationInfo
{
    uint32_t ea;
    uint32_t offset;
    uint32_t location;
    uint32_t endEa;
    uint32_t rsxLabelEa;
    bool isVertexData;
};
```

Members

<i>ea</i>	Effective address of the allocation
<i>offset</i>	RSX™ offset of the allocation
<i>location</i>	RSX™ location of the allocation CELL_GCM_LOCATION_MAIN or CELL_GCM_LOCATION_LOCAL
<i>endEa</i>	End of the allocation (ring buffers only)
<i>rsxLabelEa</i>	Effective address of an RSX™ label (ring buffers only)
<i>isVertexData</i>	True if passed to edgeGeomOutputIndexes() or edgeGeomOutputVertexes()

Description

This structure describes the allocation reserved by a call to the [edgeGeomAllocateOutputSpace\(\)](#) application's output buffering scheme. It contains six components, two of which are only used when using ring buffers.

location refers to whether the allocation resides in main memory or in RSX™ local memory.

isVertexData and *rsxLabelEa* together are used to determine whether it is necessary to invalidate the vertex cache due to cached data being stored in space allocated from a ring buffer.

See Also

[edgeGeomAllocateOutputSpace](#), [edgeGeomIsAllocatedFromRingBuffer](#), [edgeGeomOutputIndexes](#), [edgeGeomOutputVertexes](#), [edgeGeomEndCommandBufferHole](#)

EdgeGeomLocation

Structure that describes where in memory something is located from both the CELL and RSX™ perspectives.

Definition

```
#include <edgegeom_structs.h>
struct __attribute__((__aligned__(16))) EdgeGeomLocation
{
    uint32_t ea;
    uint32_t offset;
    uint32_t location;
};
```

Members

<i>ea</i>	Effective address of the allocation
<i>offset</i>	RSX™ offset of the allocation
<i>location</i>	RSX™ location of the allocation CELL_GCM_LOCATION_MAIN or CELL_GCM_LOCATION_LOCAL

Description

location refers to whether the allocation resides in main memory or in RSX™ local memory.

See Also

[edgeGeomOutputVertexes](#), [edgeGeomOutputIndexes](#), [edgeGeomSetVertexDataArrays](#)

EdgeGeomCullingResults

Structure that stores the number of triangles culled by each test case.

Definition

```
#include <edgegeom_structs.h>
struct __attribute__((__aligned__(16)))
EdgeGeomCullingResults
{
    uint16_t numOccludedTriangles;
    uint16_t numOutsideFrustumTriangles;
    uint16_t numNoPixelTriangles;
    uint16_t numBackFacingTriangles;
    uint16_t totalNumCulledTriangles;
    uint16_t pad[3];
}
```

Members

<i>numOccludedTriangles</i>	The number of triangles culled by occlusion testing
<i>numOutsideFrustumTriangles</i>	The number of triangles outside of the view frustum
<i>numNoPixelTriangles</i>	The number of triangles that do not pass the sample point tests
<i>numBackFacingTriangles</i>	The number of back-facing triangles
<i>totalNumCulledTriangles</i>	The total number of culled triangles
<i>pad</i>	Padding for byte alignment

Description

This structure is provided for profiling purposes. When used in conjunction with the C versions of the occlusion and triangle culling functions, this structure will be updated to contain the culled triangle counts per test. The value of *totalNumCulledTriangles* is the total number of triangles culled, which is not necessarily the sum of the other members of the structure because some triangles may fail multiple tests. Triangles that are occlusion-culled will not be considered for the other tests.

For performance purposes, the culling results will not be tallied unless both of the following conditions are true:

- The C implementations of the culling functions are used (see `edgegeom_config.h`).
- `EDGE_GEOM_DEBUG` is defined.

Also note that the Edge culling functions will only increment the counters; it is the user's responsibility to initialize the counters to zero before culling, if desired.

See Also

[edgeGeomCullTriangles](#), [edgeGeomCullOccludedTriangles](#)

EdgeGeomVertexStreamDescription

Structure stores custom flavor information, including callback functions and vertex stream descriptions.

Definition

```
#include <edgegeom_structs.h>
struct EdgeGeomAttributeBlock
{
    uint8_t offset;
    uint8_t format;
    uint8_t componentCount;
    uint8_t edgeAttributeId;
    uint8_t size;
    uint8_t rsxRegister;
    uint8_t fixedBlockOffset;
    uint8_t padding;
};
struct EdgeGeomAttributeFixedBlock
{
    uint8_t integer0;
    uint8_t mantissa0;
    uint8_t integer1;
    uint8_t mantissa1;
    uint8_t integer2;
    uint8_t mantissa2;
    uint8_t integer3;
    uint8_t mantissa3;
};
union EdgeGeomGenericBlock
{
    EdgeGeomAttributeBlock attributeBlock;
    EdgeGeomAttributeFixedBlock attributeBlock;
};
struct EdgeGeomVertexStreamDescription
{
    uint8_t numAttributes;
    uint8_t stride;
    uint8_t numBlocks;
    uint8_t padding[5];
    EdgeGeomGenericBlock blocks[0];
};
```

Members

<i>numAttributes</i>	The number of vertex attributes present in this vertex stream.
<i>stride</i>	The size (in bytes) of a single vertex in the stream. This size may be greater than the sum of all the individual attribute sizes, due to padding within and after each vertex.
<i>numBlocks</i>	The number of entries in the <i>blocks</i> array.
<i>padding</i>	Unused.
<i>blocks</i>	A variable-sized array of 8-byte attribute description blocks which “hangs” off the end of the structure. By convention, the first <i>numAttributes</i> entries in this array are <i>EdgeGeomAttributeBlock</i> objects (one for each vertex attribute in the stream). The remaining (<i>numBlocks</i> - <i>numAttributes</i>) entries are <i>EdgeGeomAttributeFixedBlock</i> objects (one for each fixed-point vertex attribute in the stream).

Description

This structure describes the makeup of a vertex stream, including information about each vertex attribute contained in the stream. It consists of an 8-byte header block, followed by a variable-size array of 8-byte attribute data blocks. The header block contains the stream's vertex stride and vertex attribute count, along with the number of 8-byte blocks in the variable-size blocks array.

The *blocks* array contains two types of objects. The first *numAttributes* entries are *EdgeGeomAttributeBlock* objects, each of which describes one vertex attribute. The order of these attribute blocks is undefined, and not necessarily related to the order of attributes within each vertex. Each attribute block contains information about the attribute's byte offset, data format, component count, Edge attribute ID, size in bytes, and vertex program slot index. If the attribute's type is fixed-point, the attribute block also contains a byte offset (from the beginning of the *blocks* array) to the associated *EdgeGeomAttributeFixedBlock*. These blocks start at *blocks[numAttributes]*, and contain detailed precision information for each component of a fixed point attribute.

Note that the *blocks* array technically has 0 entries, according to the structure definition. The blocks "hang" off the end of the structure, guaranteeing that each [EdgeGeomVertexStreamDescription](#) object is contiguous in memory. As a side effect, `sizeof(EdgeGeomVertexStreamDescription)` is a largely meaningless value (it only considers the size of the header block, and will always return 8 bytes). The formula for the true size of a stream description is:

```
size = streamDesc.numBlocks*8 + 8
```

EdgeGeomCustomVertexFormatInfo

Structure stores custom flavor information, including callback functions and vertex stream descriptions.

Definition

```
#include <edgegeom.h>
struct EdgeGeomCustomVertexFormatInfo
{
    EdgeGeomVertexStreamDescription *inputStreamDescA;
    EdgeGeomVertexStreamDescription *inputStreamDescB;
    EdgeGeomVertexStreamDescription *outputStreamDesc;
    EdgeGeomVertexStreamDescription *blendStreamDesc;
    EdgeGeomDecompressVertexStreamCallback decompressInputCallbackA;
    EdgeGeomDecompressVertexStreamCallback decompressInputCallbackB;
    EdgeGeomBlendVertexStreamCallback decompressBlendCallback;
    EdgeGeomGetOutputVertexStrideCallback outputStrideCallback;
    EdgeGeomCompressVertexStreamCallback compressOutputCallback;
    EdgeGeomSetVertexDataArraysCallback setVertexDataArraysCallback;
};
```

Members

<i>inputStreamDescA</i>	Pointer to the primary input stream description structure.
<i>inputStreamDescB</i>	Pointer to the secondary input stream description structure. Should be NULL if there is no secondary vertex stream.
<i>outputStreamDesc</i>	Pointer to the output stream description structure.
<i>blendStreamDesc</i>	Pointer to the blend shape delta stream description structure.
<i>decompressInputCallbackA</i>	Callback function to decompress primary input vertex stream.
<i>decompressInputCallbackB</i>	Callback function to decompress secondary input vertex stream.
<i>decompressBlendCallback</i>	Callback function to decompress blend shape stream data.
<i>outputStrideCallback</i>	Callback function to calculate the stride length of vertex data in custom output flavor.
<i>compressOutputCallback</i>	Callback function to compress output vertex data stream.
<i>setVertexDataArraysCallback</i>	Callback function to generate graphics commands to setup vertex data stream and fill the command buffer hole.

Description

This container structure should be filled out by the user with all relevant custom vertex format information. Edge supports two different methods for compressing and decompressing vertex streams: stream description structures and custom callback functions. If a segment does not use custom formats (that is, its [EdgeGeomSpuConfigInfo](#) structure contains only built-in format IDs), then all fields of this structure will be ignored and can take any value (though NULL is safest).

Stream description structures are generated by the tools for custom vertex formats. The Edge runtime parses the stream description and compresses/decompresses the stream appropriately.

Alternatively, if even more flexibility is required, users can use *decompressInputCallback*, *decompressBlendCallback* and *compressOutputCallback* to overload the appropriate functions with their custom SPU code that is linked in with the Edge Geometry SPU elf. When providing a *compressOutputCallback*, it is necessary to also provide an *outputStrideCallback* and a *setVertexDataArraysCallback*.

If both a callback and a stream description are provided for a particular operation (input, output, or blending), the callback takes precedence.

EdgeGeomCustomTransformVertexesForCullCallbackInfo

Structure that stores a custom [EdgeGeomTransformVertexesForCullCallback](#) and a pointer to user data.

Definition

```
#include <edgegeom.h>
struct EdgeGeomCustomTransformVertexesForCullCallbackInfo
{
    EdgeGeomTransformVertexesForCullCallback transformCallback;
    void *transformCallbackUserData;
};
```

Members

<i>transformCallback</i>	Callback for transforming vertices
<i>transformCallbackUserData</i>	Pointer to any user data needed by the callback

Description

This structure contains a callback and a pointer to user data needed for a custom vertex transform to be performed prior to triangle culling. A pointer to the structure should be provided to [edgeGeomInitialize\(\)](#).

See Also

[edgeGeomInitialize](#), [edgeGeomTransformVertexes](#), [EdgeGeomTransformVertexesForCullCallback](#)

EdgeGeomSpuContext

Structure that stores SPU context-specific data

Definition

```
#include <edgegeom_structs.h>
struct EdgeGeomSpuContext
{
    /* omitted */
};
```

Description

This structure contains SPU context-specific data. The application must instantiate this structure and pass it to [edgeGeomInitialize\(\)](#), where it will be initialized.

See Also

[edgeGeomInitialize](#)

Functions for SPU Runtime

edgeGeomInitialize

Initializes the runtime SPU geometry processing.

Definition

```
#include <edgegeom.h>
void edgeGeomInitialize (
    EdgeGeomSpuContext *ctx
    EdgeGeomSpuConfigInfo *spuConfigInfo,
    void *scratchBuffer,
    uint32_t scratchBufferSize,
    void *ioBuffer,
    uint32_t ioBufferSize,
    uint32_t dmaTag,
    const EdgeGeomViewportInfo *inViewportInfo = 0,
    const EdgeGeomLocalToWorldMatrix *inLocalToWorldMatrix = 0,
    const EdgeGeomCustomVertexFormatInfo *customFormatInfo = 0,
    const EdgeGeomCustomTransformVertexesForCullCallbackInfo
        *customTransformInfo = 0,
    uint32_t gcmControlEa = 0
)
```

Arguments

<i>ctx</i>	A pointer to the SPU context.
<i>spuConfigInfo</i>	A pointer to the configuration information. This data is copied into the context.
<i>scratchBuffer</i>	A pointer to the beginning of the SPURS scratch buffer.
<i>scratchBufferSize</i>	Size of the SPURS scratch buffer in bytes.
<i>ioBuffer</i>	A pointer to the beginning of the SPURS I/O buffer.
<i>ioBufferSize</i>	Size of the SPURS I/O buffer in bytes.
<i>dmaTag</i>	The DMA tag that is used for data transfers.
<i>inViewportInfo</i>	A pointer to a viewport info structure. This data is copied into the context.
<i>inLocalToWorldMatrix</i>	A pointer to a local to world matrix. This data is copied into the context.
<i>customFormatInfo</i>	A pointer to the custom vertex format info. This data is copied into the context.
<i>customTransformInfo</i>	A pointer to a callback and user data for a custom vertex transformation, to replace the default transform used by triangle culling.
<i>gcmControlEa</i>	The effective address in main memory of the GCM control structure.

Return Values

None.

Description

This function initializes the Edge Geometry library and should be called before any processing operations are executed.

Basically this function:

- Copies the [EdgeGeomViewportInfo](#) to the context
- Copies the [EdgeGeomLocalToWorldMatrix](#) to the context

- Copies [EdgeGeomSpuConfigInfo](#) to the context
- Copies the Fixed Offsets to the context
- Sets up pointers to the vertex uniform tables
- Sets up the user/custom callback
- Sets the *free pointer* to the end of the I/O buffer

For more information about the free pointer and Edge Geometry SPU local storage management, see the *PlayStation®Edge Library Overview*.

edgeGeomValidateBufferOrder

Validates the order of buffers as they have been arranged in the DMA list.

Definition

```
#include <edgegeom.h>
uint32_t edgeGeomValidateBufferOrder (
    const void *pOutputStreamDesc,
    const void *pIndexes,
    const void *pSkinMatrices,
    const void *pSkinWeights,
    const void *pVertexesA,
    const void *pVertexesB,
    const void *pViewportInfo,
    const void *pLocalToWorld,
    const void *pSpuConfigInfo,
    const void *pFixedOffsetsA,
    const void *pFixedOffsetsB,
    const void *pInputStreamDescA,
    const void *pInputStreamDescB
)
```

Arguments

<i>pOutputStreamDesc</i>	A pointer to the stream description structure representing the configuration of output data stream.
<i>pIndexes</i>	A pointer to the input index data.
<i>pSkinMatrices</i>	A pointer to the input skinning matrices.
<i>pSkinWeights</i>	A pointer to the input skinning weights and indexes.
<i>pVertexesA</i>	A pointer to the first vertex data stream.
<i>pVertexesB</i>	A pointer to the second vertex data stream.
<i>pViewportInfo</i>	A pointer to the viewport info structure.
<i>pLocalToWorld</i>	A pointer to the local to world matrix.
<i>pSpuConfigInfo</i>	A pointer to the SPU configuration info structure.
<i>pFixedOffsetsA</i>	A pointer to the fixed-point offset data, used to bias fixed-point attributes back to their original range during decompression. This is the data for the first vertex stream.
<i>pFixedOffsetsB</i>	A pointer to the fixed-point offset data, used to bias fixed-point attributes back to their original range during decompression. This is the data for the second vertex stream.
<i>pInputStreamDescA</i>	A pointer to the stream description structure representing the configuration of input data stream A.
<i>pInputStreamDescB</i>	A pointer to the stream description structure representing the configuration of input data stream B.

Return Values

A return value of 0 indicates no errors or warnings.

Otherwise this function returns a logical OR of codes for error conditions encountered. The following table shows a list of error conditions and codes:

Macro	Value	Description
EDGE_GEOM_VALIDATE_WARNING_VIEWPORT_INFO	0x00000001	Performance warning
EDGE_GEOM_VALIDATE_WARNING_LOCAL_TO_WORLD	0x00000002	Performance warning
EDGE_GEOM_VALIDATE_WARNING_SPU_CONFIG_INFO	0x00000004	Performance warning

Macro	Value	Description
EDGE_GEOM_VALIDATE_WARNING_FIXED_OFFSETS	0x00000008	Performance warning
EDGE_GEOM_VALIDATE_WARNING_CUSTOM_INPUT_FORMAT	0x00000010	Performance warning
EDGE_GEOM_VALIDATE_WARNING_FAST_PATH	0x00000020	Performance warning
EDGE_GEOM_VALIDATE_ERROR_SKINNING_MATRICES	0x00010000	Will cause fatal error
EDGE_GEOM_VALIDATE_ERROR_SKINNING_WEIGHTS	0x00020000	Will cause fatal error
EDGE_GEOM_VALIDATE_ERROR_INDEXES	0x00040000	Will cause fatal error
EDGE_GEOM_VALIDATE_ERROR_CUSTOM_OUTPUT_FORMAT	0x00080000	Will cause fatal error
EDGE_GEOM_VALIDATE_ERROR_FAST_PATH	0x00100000	Will cause fatal error

Description

This function checks the validity of the buffer ordering. In some cases, a misplaced buffer is a fatal error; in others, it is simply a performance warning. For more information on the correct buffer ordering, see the *PlayStation®Edge Library Overview*.

The following macros can be used to quickly check whether any errors or warnings were reported by the function.

Macro	Value
EDGE_GEOM_VALIDATE_ERROR_MASK	0xFFFF0000
EDGE_GEOM_VALIDATE_WARNING_MASK	0x0000FFFF

Notes

This function will only validate buffer ordering if `EDGE_GEOM_DEBUG` is defined when Edge Geometry is built.

edgeGeomGetSpuConfigInfo

Gets the local copy of the SPU configuration information.

Definition

```
#include <edgegeom.h>
EdgeGeomSpuConfigInfo *edgeGeomGetSpuConfigInfo (
    EdgeGeomSpuContext *ctx
)
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

Returns a pointer to the context's SPU configuration info.

Description

This function returns a pointer to the context's copy of the [EdgeGeomSpuConfigInfo](#) passed into initialization.

edgeGeomGetViewportInfo

Gets the local copy of the viewport information.

Definition

```
#include <edgegeom.h>
EdgeGeomViewportInfo *edgeGeomGetViewportInfo (
    EdgeGeomSpuContext *ctx
)
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

Returns the context's copy of the viewport information.

Description

Returns a pointer to the context's copy of the [EdgeGeomViewportInfo](#) passed into initialization.

edgeGeomGetLocalToWorldMatrix

Gets the local copy of the local-to-world matrix.

Definition

```
#include <edgegeom.h>
EdgeGeomLocalToWorldMatrix *edgeGeomGetLocalToWorldMatrix (
    EdgeGeomSpuContext *ctx
);
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

Returns the context's copy of the local-to-world matrix.

Description

Returns a pointer to the context's copy of the [EdgeGeomLocalToWorldMatrix](#) passed into initialization.

edgeGeomGetUniformTable

Gets the specified uniform table.

Definition

```
#include <edgegeom.h>
qword *edgeGeomGetUniformTable (
    EdgeGeomSpuContext *ctx,
    uint32_t index
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>index</i>	The uniform table index

Return Values

Returns the specified uniform table in scratch memory. The index is an absolute index into the uniform table array. To locate the uniform table for a particular vertex attribute, use [edgeGeomGetUniformTableByAttribute\(\)](#).

Description

The uniform tables will be placed starting at the beginning of the scratch buffer passed to [edgeGeomInitialize\(\)](#).

The actual initialization of the data in these tables happens inside the function [edgeGeomDecompressVertexes\(\)](#).

The order of the uniform tables depends on the vertex data input flavor. Tables will be assigned to attributes in the order in which they are decompressed.

Notes

For the runtime performance, there is no error check inside the function.

See Also

[edgeGeomInitialize](#), [edgeGeomDecompressVertexes](#), [edgeGeomGetUniformTableByAttribute](#)

edgeGeomGetUniformTableByAttribute

Gets the specified uniform table.

Definition

```
#include <edgegeom.h>
qword *edgeGeomGetUniformTableByAttribute (
    EdgeGeomSpuContext *ctx,
    EdgeGeomAttributeId attrId
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>attrId</i>	The attribute ID whose uniform table should be returned

Return Values

Returns the uniform table in scratch memory which corresponds to the specified vertex attribute. To access the uniform table at a specific absolute index, use [edgeGeomGetUniformTable\(\)](#).

Description

The uniform tables will be placed starting at the beginning of the scratch buffer passed to [edgeGeomInitialize\(\)](#).

The actual initialization of the data in these tables happens inside the function [edgeGeomDecompressVertexes\(\)](#).

Notes

For the runtime performance, there is no error check inside the function.

See Also

[edgeGeomInitialize](#), [edgeGeomDecompressVertexes](#), [edgeGeomGetUniformTableByAttribute](#)

edgeGeomGetUniformTables

Gets the internal array of uniform table pointers.

Definition

```
#include <edgegeom.h>
qword *edgeGeomGetUniformTables (
    EdgeGeomSpuContext *ctx
);
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

Returns a pointer to the array of uniform table pointers, which is a `qword*[16]` array.

Description

This function returns a pointer to a `qword*[16]` array. Each of the elements of the array points to a uniform table. See the “[edgeGeomGetUniformTable](#)” section for details about uniform tables.

See Also

[edgeGeomGetUniformTable](#)

edgeGeomGetUniformTableCount

Gets the number of uniform tables.

Definition

```
#include <edgegeom.h>
uint32_t edgeGeomGetUniformTableCount (
    EdgeGeomSpuContext *ctx
)
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

Number of uniform tables in scratch memory.

Description

This function returns the number of uniform tables, which is equal to `(spuConfigInfo.flagsAndUniformTableCount & 0xF) + 1`.

See Also

[edgeGeomGetUniformTables](#)

edgeGeomAssignUniformTable

Assigns the next available (unassigned) uniform table to the specified vertex attribute ID.

Definition

```
#include <edgegeom.h>
uint32_t edgeGeomAssignUniformTable (
    EdgeGeomSpuContext *ctx,
    EdgeGeomAttributeId attrId
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>attrId</i>	The vertex attribute ID to assign to the next available uniform table

Return Values

If successful, the return value is the index of the newly-assigned uniform table (which will always be 0..16). If no available uniform tables remain, the function returns -1.

Description

This function finds the first available uniform table and assigns it to the specified vertex attribute ID. It is not normally necessary to call this function; Edge Geometry automatically sets up all relevant uniform tables during the vertex decompression stage. However, if the user is somehow generating new vertex attributes on the SPU at runtime (for example, creating a binormal table by crossing the normal with the tangent), it is necessary to call this function to assign a table to the binormal attribute.

Notes

This function will not allocate new uniform tables; the number of tables is fixed at initialization, according to the *flagsAndUniformTableCount* member of [EdgeGeomSpuConfigInfo](#). It may be necessary to unassign unused uniform tables before new ones can be mapped.

See Also

[edgeGeomUnassignUniformTable](#)

edgeGeomUnassignUniformTable

Unmaps the specified vertex attribute uniform table, making it available to be reassigned to a different attribute.

Definition

```
#include <edgegeom.h>
uint32_t edgeGeomUnassignUniformTable (
    EdgeGeomSpuContext *ctx,
    EdgeGeomAttributeId attrId
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>attrId</i>	The ID of the vertex attribute whose uniform table should be unassigned

Return Values

If successful, the return value is the index of the newly-unassigned uniform table (which will always be 0..16). If the specified attribute ID was not found, the function returns -1.

Description

This function locates the uniform table associated with the specified vertex attribute ID, and marks it as unused. This allows users to subsequently reassign the table to a new vertex attribute using [edgeGeomAssignUniformTable\(\)](#).

See Also

[edgeGeomAssignUniformTable](#)

edgeGeomGetPositionUniformTable

Gets the uniform table associated with position data.

Definition

```
#include <edgegeom.h>
qword *edgeGeomGetPositionUniformTable (
    EdgeGeomSpuContext *ctx
)
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

Returns a pointer to the position uniform table in scratch memory.

Description

This function returns a pointer to vertex position uniform table.

If the input data has been correctly decompressed, the data in this table is in `vec_float4` format.

Notes

This function is only valid if called after vertex decompression or after the position uniform table has been explicitly set with [edgeGeomSetPositionUniformTable\(\)](#).

See Also

[edgeGeomSetPositionUniformTable](#)

edgeGeomSetPositionUniformTable

Sets the uniform table associated with position data.

Definition

```
#include <edgegeom.h>
void edgeGeomSetPositionUniformTable (
    EdgeGeomSpuContext *ctx
    qword *table
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>table</i>	Pointer to specified position uniform table

Return Values

None.

Description

You can call this function to set the context's position uniform table pointer to the specified place.

This is normally unnecessary because the pointer will be set correctly by the Edge runtime in nearly all circumstances.

The internal position uniform table pointer is used by other functions in the library. For example, [edgeGeomProcessBlendShapes\(\)](#), [edgeGeomCullOccludedTriangles\(\)](#), [edgeGeomCullTriangles\(\)](#), and other functions that reference vertex position data.

See Also

[edgeGeomGetPositionUniformTable](#)

edgeGeomGetTransformUniformTable

Gets the uniform table associated with transformed positions.

Definition

```
#include <edgegeom.h>
qword *edgeGeomGetTransformUniformTable (
    EdgeGeomSpuContext *ctx
)
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

Returns the uniform table associated with transformed positions.

Description

This function returns the uniform table that contains transformed positions.

This table is only valid after a call to an [edgeGeomCullOccludedTriangles\(\)](#) or [edgeGeomCullTriangles\(\)](#) function when the culling flavor is not set to `EDGE_GEOM_CULL_NONE`.

Notes

This transformed position table should always be the last uniform table.

The data contained in this uniform table is normally only valid after [edgeGeomCullOccludedTriangles\(\)](#) or [edgeGeomCullTriangles\(\)](#).

See Also

[edgeGeomCullOccludedTriangles](#), [edgeGeomCullTriangles](#)

edgeGeomGetNormalUniformTable

Gets the uniform table associated with normals.

Definition

```
#include <edgegeom.h>
qword *edgeGeomGetNormalUniformTable (
    EdgeGeomSpuContext *ctx
)
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

Returns the uniform table associated with normal data.

Description

This function returns a pointer to the uniform table that contains normals.

This normal uniform table should be ready after calling function [edgeGeomDecompressVertexes\(\)](#).

Notes

This is only valid after vertex decompression or an explicit call to [edgeGeomSetNormalUniformTable\(\)](#).

See Also

[edgeGeomSetNormalUniformTable](#), [edgeGeomDecompressVertexes](#)

edgeGeomSetNormalUniformTable

Sets the uniform table associated with vertex normals.

Definition

```
#include <edgegeom.h>
void edgeGeomSetNormalUniformTable (
    EdgeGeomSpuContext *ctx
    qword *table
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>table</i>	A pointer to the uniform table

Return Values

None.

Description

This function sets the context's normal uniform table pointer. You can use this to set the table to your own normals. This is normally unnecessary because the pointer will be set correctly by the Edge runtime in nearly all circumstances.

See Also

[edgeGeomGetNormalUniformTable](#)

edgeGeomGetTangentUniformTable

Gets the uniform table associated with vertex tangents.

Definition

```
#include <edgegeom.h>
qword *edgeGeomGetTangentUniformTable (
    EdgeGeomSpuContext *ctx
)
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

Returns the uniform table associated with vertex tangents.

Description

This function returns a pointer to the uniform table that contains tangent data.

Notes

The data contained in the returned pointer is only valid after vertex decompression or an explicit call to [edgeGeomSetTangentUniformTable\(\)](#).

See Also

[edgeGeomSetTangentUniformTable](#)

edgeGeomSetTangentUniformTable

Sets the uniform table associated with vertex tangents.

Definition

```
#include <edgegeom.h>
void edgeGeomSetTangentUniformTable (
    EdgeGeomSpuContext *ctx,
    qword *table
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>table</i>	A pointer to the uniform table

Return Values

None.

Description

This function sets the context's tangent table pointer to point to specified tangent table.
This is normally unnecessary because the pointer will be set correctly by the Edge runtime in nearly all circumstances.

See Also

[edgeGeomGetTangentUniformTable](#)

edgeGeomGetBinormalUniformTable

Gets the uniform table associated with vertex bi-normals.

Definition

```
#include <edgegeom.h>
qword *edgeGeomGetBinormalUniformTable (
    EdgeGeomSpuContext *ctx
)
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

Returns the uniform table associated with vertex bi-normals.

Description

This function returns the uniform table that contains bi-normal data.

Notes

The returned pointer is only valid after vertex decompression or an explicit call to [edgeGeomSetBinormalUniformTable\(\)](#).

See Also

[edgeGeomSetBinormalUniformTable](#)

edgeGeomSetBinormalUniformTable

Sets the uniform table associated with vertex bi-normals.

Definition

```
#include <edgegeom.h>
void edgeGeomSetBinormalUniformTable (
    EdgeGeomSpuContext *ctx,
    qword *table
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>table</i>	A pointer to the uniform table

Return Values

None.

Description

This function sets the context's bi-normal table pointer to point to the specified bi-normal table.

This is normally unnecessary because the pointer will be set correctly by the Edge runtime in nearly all circumstances.

See Also

[edgeGeomGetBinormalUniformTable](#)

edgeGeomGetIndexTable

Returns a pointer to the decompressed index table.

Definition

```
#include <edgegeom.h>
uint16_t *edgeGeomGetIndexTable (
    EdgeGeomSpuContext *ctx
)
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

Pointer to the decompressed index table. If this function is called before [edgeGeomDecompressIndexes\(\)](#), the return value will be undefined.

Notes

The number of elements in the index table can always be retrieved by calling [edgeGeomGetIndexCount\(\)](#). In addition, The number of elements *after* occlusion culling is returned by [edgeGeomCullOccludedTriangles\(\)](#), and the number of elements *after* triangle culling is returned by [edgeGeomCullTriangles\(\)](#).

See Also

[EdgeGeomSpuConfigInfo](#), [edgeGeomCullTriangles](#), [edgeGeomCullOccludedTriangles](#)

edgeGeomGetIndexCount

Returns the number of indexes in the index table.

Definition

```
#include <edgegeom.h>
uint16_t edgeGeomGetIndexCount (
    EdgeGeomSpuContext *ctx
)
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

The number of indexes in the index table.

Description

This function provides user access to the internal count of indexes in the index table. This number is updated by both [edgeGeomCullOccludedTriangles\(\)](#) and [edgeGeomCullTriangles\(\)](#). The initial value is copied from the [EdgeGeomSpuConfigInfo](#) provided to [edgeGeomInitialize\(\)](#). Any user changes to the index count must be propagated to the Edge internals via a call to [edgeGeomSetIndexCount\(\)](#).

See Also

[EdgeGeomSpuConfigInfo](#), [edgeGeomInitialize](#), [edgeGeomCullTriangles](#),
[edgeGeomCullOccludedTriangles](#), [edgeGeomSetIndexCount](#)

edgeGeomSetIndexCount

Sets the number of indexes in the index table.

Definition

```
#include <edgegeom.h>
void edgeGeomSetIndexCount (
    EdgeGeomSpuContext *ctx,
    uint16_t count
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>count</i>	The new index count

Return Values

None.

Description

This function provides user access to set the internal count of indexes in the index table. The user must update the index count with this function after any user processing that modifies the index count. The current index count can be accessed through [edgeGeomGetIndexCount\(\)](#).

See Also

[edgeGeomGetIndexCount](#)

edgeGeomGetVertexCount

Returns the number of vertexes in the vertex table.

Definition

```
#include <edgegeom.h>
uint16_t edgeGeomGetVertexCount (
    EdgeGeomSpuContext *ctx
)
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

The number of vertexes in the vertex table.

Description

This function provides user access to the internal count of vertexes in the vertex table. The initial value is copied from the [EdgeGeomSpuConfigInfo](#) provided to [edgeGeomInitialize\(\)](#). Any user changes to the vertex count must be propagated to the Edge internals via a call to [edgeGeomSetVertexCount\(\)](#).

See Also

[EdgeGeomSpuConfigInfo](#), [edgeGeomInitialize](#), [edgeGeomSetVertexCount](#)

edgeGeomSetVertexCount

Sets the number of vertexes in the vertex table.

Definition

```
#include <edgegeom.h>
void edgeGeomSetVertexCount (
    EdgeGeomSpuContext *ctx,
    uint16_t count
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>count</i>	The new vertex count

Return Values

None.

Description

This function provides user access to set the internal count of vertexes in the index table. The user must update the vertex count with this function after any user processing that modifies the vertex count. The current vertex count can be accessed through [edgeGeomGetVertexCount\(\)](#).

See Also

[edgeGeomGetVertexCount](#)

edgeGeomGetFreePtr

Gets a pointer to the end of used SPURS I/O buffer memory.

Definition

```
#include <edgegeom.h>
void *edgeGeomGetFreePtr (
    EdgeGeomSpuContext *ctx
)
```

Arguments

<i>ctx</i>	A pointer to the context
------------	--------------------------

Return Values

Returns a pointer to the end of used SPURS I/O buffer memory.

Description

This function returns the current free pointer. See the *PlayStation®Edge Library Overview* for more information about the free pointer.

This free pointer is pointing to the “start of free space” in the Input/Output buffer.

For example, after an [edgeGeomDecompressVertexes\(\)](#) function call, it is usually unnecessary to keep compressed input vertex data in the I/O buffer. So the free pointer is set to the start of the input vertex buffer at the end of function [edgeGeomDecompressVertexes\(\)](#).

Notes

This is only valid after [edgeGeomInitialize\(\)](#).

See Also

[edgeGeomInitialize](#), [edgeGeomDecompressVertexes](#)

edgeGeomSetFreePtr

Sets the free pointer to the specified address.

Definition

```
#include <edgegeom.h>
void edgeGeomSetFreePtr (
    EdgeGeomSpuContext *ctx
    const void *ptr
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>ptr</i>	The pointer to set

Return Values

None.

Description

This function sets the current free pointer to point to a specific buffer.

Notes

The buffer pointed by this pointer is used internally by the library. For more information about the free space, see the “[edgeGeomGetFreePtr](#)” section and the *PlayStation®Edge Library Overview*.

See Also

[edgeGeomGetFreePtr](#)

edgeGeomIsAllocatedFromRingBuffer

Tests whether output space is allocated from a ring buffer.

Definition

```
#include <edgegeom.h>
bool edgeGeomIsAllocatedFromRingBuffer (
    EdgeGeomAllocationInfo *info
)
```

Arguments

info Pointer to an allocation info structure

Return Values

Returns true if space pointed to by *info* is allocated from a ring buffer. Returns false otherwise.

Description

This function determines whether output space is allocated from a ring buffer. This is useful in determining whether cached data on the RSX™ should be invalidated.

Notes

This function does not need to be called in the default Edge Geometry usage pattern. Edge internally manages invalidation of the vertex cache when ring buffers are in use. However, if your code allocates vertex or texture data for output, use this function to determine whether a ring buffer is used and thus whether the corresponding cache should be invalidated.

See Also

[edgeGeomAllocateOutputSpace](#)

edgeGeomGetOutputVertexStride

Gets the stride size for the specified output flavor.

Definition

```
#include <edgegeom.h>
uint32_t edgeGeomGetOutputVertexStride (
    EdgeGeomSpuContext *ctx,
    uint32_t outputFormatId
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>outputFormatId</i>	The ID of the RSX™ output vertex format for which to retrieve the stride

Return Values

Returns the size in bytes of a single output vertex. Returns 0 if the output format does not exist internally and the output stride callback function is not registered on [EdgeGeomSpuContext](#).

Description

If the specified output vertex format exists internally, this function returns the appropriate stride for this output flavor. Otherwise, the function checks if the output stride callback function, [EdgeGeomGetOutputVertexStrideCallback\(\)](#), is set, and if so, calls it. If the callback function does not exist, the function returns 0.

edgeGeomDecompressVertexes

Decompresses input vertex data according to the input vertex format IDs specified in the `EdgeGeomSpuConfigInfo`.

Definition

```
#include <edgegeom.h>
void edgeGeomDecompressVertexes (
    EdgeGeomSpuContext *ctx,
    const void *vertexesA,
    const void *fixedOffsetsA,
    const void *vertexesB,
    const void *fixedOffsetsB
)
```

Arguments

<i>ctx</i>	A pointer to the context.
<i>vertexesA</i>	Pointer to the first input vertex data stream.
<i>fixedOffsetsA</i>	Pointer to the fixed offsets buffer used when decompressing the first vertex stream.
<i>vertexesB</i>	Pointer to second input vertex data stream.
<i>fixedOffsetsB</i>	This could be 0 if there is no second input vertex stream. Pointer to the fixed offsets buffer used when decompressing the second vertex stream.

Return Values

None.

Description

This function will first extract the input format ID from the context's [EdgeGeomSpuConfigInfo\(\)](#) variable, which is specified as an argument in [edgeGeomInitialize\(\)](#) function.

The function sets the free pointer to the start of the input vertex buffer after it finishes the decompression.

The *fixedOffsets** arguments refer to tables of fixed point offsets provided by the tools for this segment. They contain four floating-point offsets per fixed point attribute in the input flavor. These offsets are added to the decompressed fixed-point attribute values after their conversion back to floats, to get the values back to the correct original range.

To add support for a new statically-linked input vertex format, you need to add a new case block to the switch statement in the internal function [edgeGeomDecompressVertexes\(\)](#). The macros used inside these case blocks (defined in `edgegeom_decompress.h`) allow the clear and concise implementation of near-optimal decompression code.

Note: The `EDGE_GEOM_DECOMPRESS` macros rely heavily on compile-time constants, so that the optimizing compiler can pre-compute as much as possible. For this reason, it is necessary that all arguments to these macros be compile-time constants (either numerical literals or `#define` constants. In practice this shouldn't be an issue, because the parameter values should be known at compile time.

Within the case block, the first macro that must be called is `EDGE_GEOM_DECOMPRESS_INIT_GLOBAL`. The macro's single parameter is the total vertex stride, in bytes.

Next, list the attributes to be decompressed. The attribute types supported by the decompression macros are:

Attribute Type	Description
F32	32-bit floating point.
F16	16-bit floating point.
U8	8-bit signed integer.
I16	16-bit signed integer.
U8N	8-bit unsigned integer, normalized to represent the range $[0.0 \dots 1.0]$.
I16N	16-bit signed integer, normalized to represent the range $[-1.0 \dots 1.0]$.
X11Y11Z10N	32-bit packed 3-component vector. Basically an I11N for X, an I11N for Y and an I10N for Z, packed into one 32-bit value.
FIXED_POINT	Edge-specific format that encodes each vector component as a fixed-point value; see the sections about data compression in the “The Specifics of Edge Geometry Runtime” chapter of the <i>PlayStation®Edge Library Overview</i> for more information.
UNIT_VECTOR	Edge-specific format that packs a unit vector into 24 bits; see the sections about data compression in the “The Specifics of Edge Geometry Runtime” chapter of the <i>PlayStation®Edge Library Overview</i> for more information.

For each attribute, call the appropriate macro from the list below in the order in which the attributes appear within the vertex:

- `EDGE_GEOM_DECOMPRESS_INIT_F32(attrId, attrOffset, attrComponentCount)`
- `EDGE_GEOM_DECOMPRESS_INIT_F16(attrId, attrOffset, attrComponentCount)`
- `EDGE_GEOM_DECOMPRESS_INIT_U8(attrId, attrOffset, attrComponentCount)`
- `EDGE_GEOM_DECOMPRESS_INIT_I16(attrId, attrOffset, attrComponentCount)`
- `EDGE_GEOM_DECOMPRESS_INIT_U8N(attrId, attrOffset, attrComponentCount)`
- `EDGE_GEOM_DECOMPRESS_INIT_I16N(attrId, attrOffset, attrComponentCount)`
- `EDGE_GEOM_DECOMPRESS_INIT_X11Y11Z10N(attrId, attrOffset, attrComponentCount)`
- `EDGE_GEOM_DECOMPRESS_INIT_FIXED_POINT(attrId, attrOffset, attrComponentCount, integerBits1, mantissaBits1, integerBits2, mantissaBits2, integerBits3, mantissaBits3, integerBits4, mantissaBits4)`
- `EDGE_GEOM_DECOMPRESS_INIT_UNIT_VECTOR(attrId, attrOffset, attrComponentCount)`

The above macros take the following parameters:

<code>attrId</code>	A numerical value that uniquely identifies this attribute, from the <code>EDGE_GEOM_ATTRIBUTE_ID_*</code> enumeration.
<code>attrOffset</code>	The byte offset of the attribute within the vertex.
<code>attrComponentCount</code>	The number of components in the attribute. Must be in the range [1-4].

In addition, `EDGE_GEOM_DECOMPRESS_INIT_FIXED_POINT` takes eight additional parameters that describe the bit depths of the integer and mantissa components of each component. The total bits used by the attribute must be divisible by 8. For unused components, use 0 for both bit depths.

Once all the attributes have been initialized, you must call `EDGE_GEOM_DECOMPRESS_LOAD_COMMON()` before entering the main do-while loop, which loops over each vertex in the input stream. For the loop condition, use `!EDGE_GEOM_DECOMPRESS_LOOP_DONE()`.

Inside the loop, first call `EDGE_GEOM_DECOMPRESS_LOOP_START()` to set up the loop body.

Next, decompress each attribute. For each attribute, call the appropriate macro from the following list (the `attrNum` parameter has the same meaning as it did previously):

- `EDGE_GEOM_DECOMPRESS_LOOP_F32(attrId)`
- `EDGE_GEOM_DECOMPRESS_LOOP_F16(attrId)`
- `EDGE_GEOM_DECOMPRESS_LOOP_U8(attrId)`

- `EDGE_GEOM_DECOMPRESS_LOOP_I16(attrId)`
- `EDGE_GEOM_DECOMPRESS_LOOP_U8N(attrId)`
- `EDGE_GEOM_DECOMPRESS_LOOP_I16N(attrId)`
- `EDGE_GEOM_DECOMPRESS_LOOP_X11Y11Z10N(attrId)`
- `EDGE_GEOM_DECOMPRESS_LOOP_FIXED_POINT(attrId)`
- `EDGE_GEOM_DECOMPRESS_LOOP_UNIT_VECTOR(attrId)`

Finish the loop with a call to `EDGE_GEOM_DECOMPRESS_LOOP_END()`.

After the loop, you must finalize the decompression process by calling the appropriate macro for each attribute type. The following list provides the finalization macro for each attribute type (the *attrId* parameter has the same meaning as before):

- `EDGE_GEOM_DECOMPRESS_FINALIZE_F32(attrId)`
- `EDGE_GEOM_DECOMPRESS_FINALIZE_F16(attrId)`
- `EDGE_GEOM_DECOMPRESS_FINALIZE_U8(attrId)`
- `EDGE_GEOM_DECOMPRESS_FINALIZE_I16(attrId)`
- `EDGE_GEOM_DECOMPRESS_FINALIZE_U8N(attrId)`
- `EDGE_GEOM_DECOMPRESS_FINALIZE_I16N(attrId)`
- `EDGE_GEOM_DECOMPRESS_FINALIZE_X11Y11Z10N(attrId)`
- `EDGE_GEOM_DECOMPRESS_FINALIZE_FIXED_POINT(attrId)`
- `EDGE_GEOM_DECOMPRESS_FINALIZE_UNIT_VECTOR(attrId)`

The following is a complete example for an input stream with a vertex stride of 30 bytes containing five attributes: a 3-component fixed-point position (X is 12.20, Y is 12.20, Z is 4.20), a 3-component X11Y11Z10N normal, a 4-component I16N tangent (the w is a flip factor), a unit-vector binormal and a 2-component I16N texture coordinate:

```
case MY_NEW_INPUT_FORMAT:
{
    EDGE_GEOM_DECOMPRESS_INIT_GLOBAL(30);

    EDGE_GEOM_DECOMPRESS_INIT_FIXED_POINT(EDGE_GEOM_ATTRIBUTE_ID_POSITION, 0, 3,
12,20, 12,20, 4,20, 0,0);
    EDGE_GEOM_DECOMPRESS_INIT_X11Y11Z10N(EDGE_GEOM_ATTRIBUTE_ID_NORMAL,
11, 3);
    EDGE_GEOM_DECOMPRESS_INIT_I16N(EDGE_GEOM_ATTRIBUTE_ID_TANGENT, 15, 4);
    EDGE_GEOM_DECOMPRESS_INIT_UNIT_VECTOR(EDGE_GEOM_ATTRIBUTE_ID_BINORMAL, 23,
3);
    EDGE_GEOM_DECOMPRESS_INIT_F16(EDGE_GEOM_ATTRIBUTE_ID_UV0, 26, 2);

    EDGE_GEOM_DECOMPRESS_LOAD_COMMON();

    do
    {
        EDGE_GEOM_DECOMPRESS_LOOP_START();

        EDGE_GEOM_DECOMPRESS_LOOP_FIXED_POINT(
EDGE_GEOM_ATTRIBUTE_ID_POSITION);
        EDGE_GEOM_DECOMPRESS_LOOP_X11Y11Z10N(
EDGE_GEOM_ATTRIBUTE_ID_NORMAL);
        EDGE_GEOM_DECOMPRESS_LOOP_I16N(EDGE_GEOM_ATTRIBUTE_ID_TANGENT);
        EDGE_GEOM_DECOMPRESS_LOOP_UNIT_VECTOR(
EDGE_GEOM_ATTRIBUTE_ID_BINORMAL);
        EDGE_GEOM_DECOMPRESS_LOOP_F16(EDGE_GEOM_ATTRIBUTE_ID_UV0);
```



```
        EDGE_GEOM_DECOMPRESS_LOOP_END();
    }
    while(!EDGE_GEOM_DECOMPRESS_LOOP_DONE());

    EDGE_GEOM_DECOMPRESS_FINALIZE_FIXED_POINT(EDGE_GEOM_ATTRIBUTE_ID
_POSITION);
    EDGE_GEOM_DECOMPRESS_FINALIZE_X11Y11Z10N(EDGE_GEOM_ATTRIBUTE_ID_NORMAL);
    EDGE_GEOM_DECOMPRESS_FINALIZE_I16N(EDGE_GEOM_ATTRIBUTE_ID_TANGENT);
    EDGE_GEOM_DECOMPRESS_FINALIZE_UNIT_VECTOR(EDGE_GEOM_ATTRIBUTE_ID
_BINORMAL);
    EDGE_GEOM_DECOMPRESS_FINALIZE_F16(EDGE_GEOM_ATTRIBUTE_ID_UV0);

    break;
}
```

See Also

[EdgeGeomSpuConfigInfo](#)

edgeGeomDecompressIndexes

Decompresses the input index data stream.

Definition

```
#include <edgegeom.h>
void edgeGeomDecompressIndexes (
    EdgeGeomSpuContext *ctx,
    const void *indexes
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>indexes</i>	Pointer to input index data stream

Return Values

None.

Description

This function extracts the *indexesFlavorAndSkinningFlavor* format from the context's [EdgeGeomSpuConfigInfo](#), which is specified as an argument in [edgeGeomInitialize\(\)](#) function. Then, this function decompresses input index data and overwrites the decompressed data on the original input index buffer.

Because the size of this new index buffer is equal to or be bigger than the original one, this function assumes that the free pointer points to the end of the input index buffer when it is called.

After calling this function the free space pointer is set to the end of the decompressed index buffer.

See Also

[EdgeGeomSpuConfigInfo](#)

edgeGeomProcessBlendShapes

Performs the blend shape operation.

Definition

```
#include <edgegeom.h>
void edgeGeomProcessBlendShapes (
    EdgeGeomSpuContext *ctx,
    uint32_t numShapes,
    uint32_t shapeInfosEa
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>numShapes</i>	Number of shapes to be blended with
<i>shapeInfosEa</i>	The effective address of the shape information data (EdgeGeomBlendShapeInfo)

Return Values

None.

Description

This function performs the blend shape process on the data in the vertex uniform tables.

For each shape, this function:

- Uses DMA in the [EdgeGeomBlendShapeInfo](#) from *shapeInfosEa* to the free pointer
- Issues a List DMA using the element in the [EdgeGeomBlendShapeInfo](#).*dmaTag* with barrier
- Waits for the above DMAs to finish
- Calls [edgeGeomDecompressVertexes\(\)](#) to decompress data with shape flavor and blend uniform tables by [EdgeGeomBlendShapeInfo](#).*alpha*
- Increments *shapeInfosEa* for next shape

See Also

[EdgeGeomBlendShapeInfo](#), [edgeGeomDecompressVertexes](#)

edgeGeomNormalizeUniformTable

Normalizes the specified uniform table.

Definition

```
#include <edgegeom.h>
void edgeGeomNormalizeUniformTable(
    qword *pUniform,
    int32_t vertexCount
)
```

Arguments

<i>pUniform</i>	Target uniform table to normalize
<i>vertexCount</i>	Number of vertexes

Return Values

None.

Description

This function normalizes the vector formed by the X, Y and Z components of each quadword of the specified uniform table, taking special care to preserve the W value of each vertex.

edgeGeomSkinVertexes

Performs skinning operation.

Definition

```
#include <edgegeom.h>
void edgeGeomSkinVertexes (
    EdgeGeomSpuContext *ctx,
    void *matrices,
    uint32_t matrixCount,
    void *indexesAndWeights
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>matrices</i>	Pointer to skin matrix data (4x3 floats)
<i>matrixCount</i>	Number of matrices
<i>indexesAndWeights</i>	Input indexes and weights data stream

Return Values

None.

Description

This function skins the uniform tables using the given the *indexesFlavorAndSkinningFlavor* as defined in the [EdgeGeomSpuConfigInfo\(\)](#).

Internally, before actually skinning vertexes, function performs a special transformation on the input skin matrices. This is the reason for the *matrixCount* argument, and you must be aware of this internal transform if you want to use the skin matrices data after calling this function.

At the end of this function, the free pointer is set to the beginning of the skin matrices, *matrices*.

edgeGeomTransformVertexes

Transforms specified uniform table.

Definition

```
#include <edgegeom.h>
void edgeGeomTransformVertexes (
    uint32_t numVertexes,
    void *inVertexes,
    void *outVertexes,
    void *matrix
)
```

Arguments

<i>numVertexes</i>	Number of vertexes
<i>inVertexes</i>	Input uniform table
<i>outVertexes</i>	Output transformed uniform table
<i>matrix</i>	4x4 matrix used to transform <i>inVertexes</i>

Return Values

None.

Description

This function multiplies the passed in 4x4 matrix with the specified uniform table.

Notes

This operation is safe to perform in place. For example, it is valid to have *inVertexes* be the same pointer as *outVertexes*.

edgeGeomCullOccludedTriangles

Culls triangles hidden from view by occluders.

Definition

```
#include <edgegeom.h>
uint32_t edgeGeomCullOccludedTriangles (
    EdgeGeomSpuContext *ctx,
    const uint32_t occludersCount,
    const uint32_t occludersEA,
    int32_t indexBias = 0,
    EdgeGeomCullingResults *detailedResults = 0
)
```

Arguments

<i>ctx</i>	A pointer to context.
<i>occludersCount</i>	The number of occluders (8 or less).
<i>occludersEA</i>	Effective address where the occlude data is located in main memory.
<i>indexBias</i>	Bias indices to be zero-based.
<i>detailedResults</i>	(Optional.) Keep track of how many triangles failed each individual culling test, for profiling purposes. This will be ignored unless the C implementation of the culling function is used and <code>EDGE_GEOM_DEBUG</code> is defined.

Return Values

Returns the number of visible indexes.

Description

If *occludersCount* is non-zero, this function will do the following:

- DMA the occluders into local memory
- Transform each occluder and each vertex into screen space
- Check each vertex to determine if it is hidden by the occluders
- Cull any triangle whose vertices are all behind at least one occluder in screen space
- Return the number of visible triangle indexes

Each occluder is defined by 16 floats representing four vertices. Each occluder's data is an array of 16 floats. This function supports a maximum of 8 occluders.

indexBias should be set such that, when added to the first index value, the result will be 0. Index biasing is provided for use with index and vertex data that is loaded as a subset of larger geometry (which may thus not be zero-indexed).

If an [EdgeGeomCullingResults](#) structure was passed, the debug-build C version of this function will increment the occlusion and culled members of the results structure for each occluded triangle.

Notes

This function should be called before [edgeGeomCullTriangles\(\)](#).

edgeGeomCullTriangles

Culls triangles with the specified culling flavor.

Definition

```
#include <edgegeom.h>
uint32_t edgeGeomCullTriangles (
    EdgeGeomSpuContext *ctx,
    uint32_t cullingFlavor,
    int32_t indexBias = 0,
    EdgeGeomCullingResults *detailedResults = 0
)
```

Arguments

<i>ctx</i>	A pointer to the context.
<i>cullingFlavor</i>	Flavor of culling operation. See “ Triangle Culling Mode ”.
<i>indexBias</i>	Bias indices to be zero-based.
<i>detailedResults</i>	(Optional.) Keep track of how many triangles failed each individual culling test, for profiling purposes. This will be ignored unless the C implementation of the culling function is used and <code>EDGE_GEOM_DEBUG</code> is defined.

Return Values

Returns the number of visible indexes.

Description

If the *cullingFlavor* is not set to `EDGE_GEOM_CULL_NONE`, this function:

- Multiplies the [EdgeGeomViewportInfo](#).*viewProjectionMatrix* by the [EdgeGeomLocalToWorldMatrix](#)
- Transforms positions to projection space and divides the resulting *x* and *y* coordinates by the *w* coordinate
- Applies viewport scales and offsets to transform positions into clip space
- Performs pre-computations for culling tests, and stores results in *Z* and *W* components
- Stores the transformed positions *X* and *Y* and the culling test values *Z* and *W* into the transform uniform table
- Culls the triangles with the culling flavor specified and writes out the output indexes to the current output pointer
- Sets the free pointer to I/O Buffer start plus the number of visible indexes times two rounded up to the next multiple of 16 bytes
- Returns the number of visible triangle indexes

If the *cullingFlavor* is set to `EDGE_GEOM_CULL_NONE`, this function returns the number of indexes specified in the [EdgeGeomSpuConfigInfo](#) structure. If [EdgeGeomSpuConfigInfo](#).*indexesOffset* is not set to -1, it sets the free pointer to the beginning of the index data. If the *indexesOffset* is -1, the free space pointer is not changed.

indexBias should be set such that, when added to the first index value, the result will be 0. Index biasing is provided for use with index and vertex data that is loaded as a subset of larger geometry, (which may thus not be zero-indexed).

If an [EdgeGeomCullingResults](#) structure was passed, the debug-build C version of this function will increment the members of the results structure for each tested triangles.

edgeGeomCalculateDefaultOutputSize

Calculates the output size used in standard operation.

Definition

```
#include <edgegeom.h>
uint32_t edgeGeomCalculateDefaultOutputSize (
    EdgeGeomSpuContext *ctx,
    uint32_t numIndexes
)
```

Arguments

<i>ctx</i>	A pointer to the context.
<i>numIndexes</i>	The number of visible indexes. Typically, this is the value returned by edgeGeomCullOccludedTriangles() or edgeGeomCullTriangles() .

Return Values

Returns the standard allocation size to pass to [edgeGeomAllocateOutputSpace\(\)](#).

Description

Calculates the output size used in standard operation.

See Also

[edgeGeomAllocateOutputSpace](#), [edgeGeomCullTriangles](#), [edgeGeomCullOccludedTriangles](#)

edgeGeomAllocateOutputSpace

Allocates output space on main/local memory.

Definition

```
#include <edgegeom.h>
bool edgeGeomAllocateOutputSpace (
    EdgeGeomSpuContext *ctx,
    uint32_t outputBufferInfoEa,
    uint32_t allocSize,
    EdgeGeomAllocationInfo *outInfo,
    uint32_t spuId
)
```

Arguments

<i>ctx</i>	A pointer to the context.
<i>outputBufferInfoEa</i>	Effective address of the EdgeGeomOutputBufferInfo structure.
<i>allocSize</i>	The size in bytes to allocate from the buffer information.
<i>outInfo</i>	The allocation information; filled out if the function succeeds.
<i>spuId</i>	The ID of this SPU. Usually retrieved from <code>cellSpursGetCurrentSpuId()</code> , which is described in the <i>libspurs Core Reference</i> .

Return Values

Returns true if the allocation is completed successfully.

Returns false otherwise.

Description

This function tries to allocate a free space in the main/local memory that can be used by the library to output indexes, vertexes and the command buffer.

If (*outputBufferInfoEa* & 0x2) is set, this function assumes there is a static output buffer with address equal to *outputBufferInfoEa* & 0xFFFFFFFF. *outputBufferInfoEa* should be masked using either `EDGE_GEOM_DIRECT_OUTPUT_TO_MAIN_MEMORY` or `EDGE_GEOM_DIRECT_OUTPUT_TO_LOCAL_MEMORY`.

Otherwise, this function tries to dynamically allocate space from a shared buffer or a ring buffer depending on the setting in [EdgeGeomOutputBufferInfo](#). As described in the “[EdgeGeomOutputBufferInfo](#)” section, a buffer is considered empty or nonexistent if its *startEa* and *endEa* fields are the same.

If only the shared buffer is not empty, the single buffer output scheme is used.

If only the ring buffer is not empty, the ring buffer output scheme is used.

Finally, if both buffers are not empty, the ring buffer output scheme is used, with the shared buffer providing overflow space when the SPU would otherwise have to wait for the RSX™.

When the ring-buffer scheme is used, because the function internally waits for RSX™ to consume certain memory, it does not return until the allocation is successfully completed.

spuId specifies which ring buffer (specifically, the one corresponding to the current SPU) to allocate from.

For more information about output-buffer schemes, see the *PlayStation®Edge Library Overview*.

Notes

If the allocation fails, any remaining processes in the library are dropped. This might cause a graphic error and crash the RSX™.

edgeGeomUseOutputSpace

Uses allocated space and updates the corresponding [EdgeGeomAllocationInfo](#) data.

Definition

```
#include <edgegeom.h>
inline void edgeGeomUseOutputSpace(
    EdgeGeomAllocationInfo *info,
    uint32_t size,
    uint32_t *ea,
    uint32_t *offset
)
```

Arguments

<i>info</i>	The structure to update after use
<i>size</i>	The amount of output space to use
<i>ea</i>	The effective address at the start of the used space
<i>offset</i>	The RSX™ offset at the start of the used space

Return Values

None.

Description

This function is provided to handle management of allocated space when the space is used for multiple, separate pieces of output data. A single allocation and this function should be used in place of multiple allocations from the same ring buffer, which can cause deadlocks. This function updates the starting effective address and RSX™ offset of the remaining space in the allocated region.

edgeGeomOutputIndexes

Outputs indexes from I/O buffer to main/local memory.

Definition

```
#include <edgegeom.h>
void edgeGeomOutputIndexes (
    EdgeGeomSpuContext *ctx,
    uint32_t numIndexes,
    EdgeGeomAllocationInfo *info,
    EdgeGeomLocation *outLoc
)
```

Arguments

<i>ctx</i>	A pointer to the context.
<i>numIndexes</i>	The number of visible indexes. Typically, this is the value returned by edgeGeomCullOccludedTriangles() , edgeGeomCullTriangles() .
<i>info</i>	Allocation information, indicates where to place the indexes.
<i>outLoc</i>	The location of the indexes.

Return Values

None.

Description

This function will DMA the indexes to the output data address and then increment the output data address by the number of visible indexes times two rounded up to the next multiple of 16 bytes.

After a call to this function, the *ea* and offset of *info* will be incremented by the appropriate amount used. This is so you can make a single large allocation, but have it contain different types of data.

Notes

If the user generates a command buffer and uses the command-buffer-hole type of synchronization, this function must be called before calling [edgeGeomBeginCommandBufferHole\(\)](#).

See Also

[edgeGeomCullTriangles](#), [edgeGeomCullOccludedTriangles](#)

edgeGeomCompressVertexes

Compresses data in uniform tables into the output vertex format specified in [EdgeGeomSpuConfigInfo](#).

Definition

```
#include <edgegeom.h>
void *edgeGeomCompressVertexes (
    EdgeGeomSpuContext *ctx
)
```

Arguments

ctx A pointer to the context

Return Values

Returns a pointer to compressed vertex data

Description

This function compresses uniform tables into the output vertex format and writes the output vertexes to the output pointer. After the compression is done, the function increments the free pointer by the output format's stride size multiplied by the number of vertexes rounded up to the next multiple of 128 bytes.

Notes

To add support for a new statically linked output vertex format, add a new case block to the switch statement in [edgeGeomCompressVertexes\(\)](#). The macros used inside these case blocks (defined in `edgegeom_compress.h`) allow the clear and concise implementation of near-optimal compression code.

Note: The `EDGE_GEOM_COMPRESS` macros rely heavily on compile-time constants, so that the optimizing compiler can pre-compute as much as possible. For this reason, all arguments to these macros *must* be compile-time constants (either numerical literals or `#define` constants). In practice this should not be an issue, because the parameter values should be known at compile time.

Within the case block, the first macro that must be called is `EDGE_GEOM_COMPRESS_INIT_GLOBAL()`. The macro's single parameter is the total vertex stride, in bytes.

Next, list the attributes to be compressed. The attribute types supported by the compression macros are:

Attribute Type	Description
F32	32-bit floating point.
F16	16-bit floating point.
U8	8-bit signed integer.
I16	16-bit signed integer.
U8N	8-bit unsigned integer, normalized to represent the range [0.0..1.0].
I16N	16-bit signed integer, normalized to represent the range [-1.0..1.0].
X11Y11Z10N	32-bit packed 3-component vector. Basically an I11N for X, an I11N for Y and an I10N for Z, packed into one 32-bit value.

For each attribute, call the appropriate macro from the list below in the order in which the attributes appear within the vertex.

- `EDGE_GEOM_COMPRESS_INIT_F32(attrId, attrOffset, attrSize)`
- `EDGE_GEOM_COMPRESS_INIT_F16(attrId, attrOffset, attrSize)`

- `EDGE_GEOM_COMPRESS_INIT_U8(attrId, attrOffset, attrSize)`
- `EDGE_GEOM_COMPRESS_INIT_I16(attrId, attrOffset, attrSize)`
- `EDGE_GEOM_COMPRESS_INIT_U8N(attrId, attrOffset, attrSize)`
- `EDGE_GEOM_COMPRESS_INIT_I16N(attrId, attrOffset, attrSize)`
- `EDGE_GEOM_COMPRESS_INIT_X11Y11Z10N(attrId, attrOffset, attrSize)`

The above macros take the following parameters:

<i>attrId</i>	A numerical value that uniquely identifies this attribute, from the <code>EDGE_GEOM_ATTRIBUTE_ID_*</code> enumeration
<i>attrOffset</i>	The byte offset of the attribute within the vertex
<i>attrSize</i>	The size of the attribute in bytes

Next, insert a do-while loop with `!EDGE_GEOM_COMPRESS_LOOP_DONE()` as the loop condition.

Inside the loop, first call `EDGE_GEOM_COMPRESS_LOOP_START()` to set up the loop body.

Next, compress each attribute. For each attribute, call the appropriate macro from the following list (the *attrId* parameter has the same meaning as it did previously):

- `EDGE_GEOM_COMPRESS_LOOP_F32(attrId)`
- `EDGE_GEOM_COMPRESS_LOOP_F16(attrId)`
- `EDGE_GEOM_COMPRESS_LOOP_U8(attrId)`
- `EDGE_GEOM_COMPRESS_LOOP_I16(attrId)`
- `EDGE_GEOM_COMPRESS_LOOP_U8N(attrId)`
- `EDGE_GEOM_COMPRESS_LOOP_I16N(attrId)`
- `EDGE_GEOM_COMPRESS_LOOP_X11Y11Z10N(attrId)`

Finish the loop with a call to `EDGE_GEOM_COMPRESS_LOOP_END()`.

The following is a full example for an output stream with a vertex stride of 32 bytes containing five attributes: a 3-component F32 position, a 3-component X11Y11Z10N normal, a 4-component I16N tangent (the w is a flip factor), a 3-component X11Y11Z10N binormal and a 2-component I16N texture coordinate:

```
case MY_NEW_OUTPUT_FORMAT:
{
    EDGE_GEOM_COMPRESS_INIT_GLOBAL(32);

    EDGE_GEOM_COMPRESS_INIT_F32(EDGE_GEOM_ATTRIBUTE_ID_POSITION, 0, 12);
    EDGE_GEOM_COMPRESS_INIT_X11Y11Z10N(EDGE_GEOM_ATTRIBUTE_ID_NORMAL, 12, 4);
    EDGE_GEOM_COMPRESS_INIT_I16N(EDGE_GEOM_ATTRIBUTE_ID_TANGENT, 16, 8);
    EDGE_GEOM_COMPRESS_INIT_X11Y11Z10N(EDGE_GEOM_ATTRIBUTE_ID_BINORMAL, 24, 4);
    EDGE_GEOM_COMPRESS_INIT_F16(EDGE_GEOM_ATTRIBUTE_ID_UV0, 28, 4);
    do
    {
        EDGE_GEOM_COMPRESS_LOOP_START();

        EDGE_GEOM_COMPRESS_LOOP_F32(EDGE_GEOM_ATTRIBUTE_ID_POSITION);
        EDGE_GEOM_COMPRESS_LOOP_X11Y11Z10N(EDGE_GEOM_ATTRIBUTE_ID_NORMAL);
        EDGE_GEOM_COMPRESS_LOOP_I16N(EDGE_GEOM_ATTRIBUTE_ID_TANGENT);
        EDGE_GEOM_COMPRESS_LOOP_X11Y11Z10N(EDGE_GEOM_ATTRIBUTE_ID_BINORMAL);
        EDGE_GEOM_COMPRESS_LOOP_F16(EDGE_GEOM_ATTRIBUTE_ID_UV0);

        EDGE_GEOM_COMPRESS_LOOP_END();
    }
    while(!EDGE_GEOM_COMPRESS_LOOP_DONE());
    break;
}
```

edgeGeomOutputVertexes

Outputs the compressed vertex data from I/O buffer to main/local memory.

Definition

```
#include <edgegeom.h>
void edgeGeomOutputVertexes (
    EdgeGeomSpuContext *ctx,
    EdgeGeomAllocationInfo *info,
    EdgeGeomLocation *outLoc
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>info</i>	Allocation information, indicates where to place the vertexes
<i>outLoc</i>	The location of the indexes is stored here

Return Values

None.

Description

This function uses DMA to output vertexes to the output data address and then increments the output data address by the output vertex stride as specified by the output format flavor multiplied by the number of vertexes rounded up to the next multiple of 128 bytes.

After this function has been called, the *ea* and *offset* of *info* are incremented by the appropriate amount used. This is so you can make a single large allocation, but have it contain different types of data.

Notes

If you generate a command buffer and use the command-buffer-hole type of synchronization, this function must be called before calling [edgeGeomBeginCommandBufferHole\(\)](#).

See Also

[edgeGeomBeginCommandBufferHole](#)

edgeGeomBeginCommandBufferHole

Begins command buffer generation to fill the command buffer hole in main memory.

Definition

```
#include <edgegeom.h>
void edgeGeomBeginCommandBufferHole (
    EdgeGeomSpuContext *ctx,
    CellGcmContextData *gcmCtx,
    uint32_t holeEa
)
```

Arguments

<i>ctx</i>	A pointer to the context.
<i>gcmCtx</i>	Context data filled out by this function; it contains the appropriate information for use in writing RSX™ commands. See the <i>libgcm Reference</i> for further information.
<i>holeEa</i>	Effective address to the command buffer needed to be filled.

Return Values

None.

Description

Begins command buffer generation to fill the command buffer hole. Subsequent commands occur in the *gcmCtx* context, which is aligned to correspond with the command buffer resident in main memory.

See Also

[EdgeGeomSpuConfigInfo](#)

edgeGeomSetVertexDataArrays

Inserts commands that set the vertex data formats and offsets for the RSX™.

Definition

```
#include <edgegeom.h>
void edgeGeomSetVertexDataArrays (
    EdgeGeomSpuContext *ctx,
    CellGcmContextData *gcmCtx,
    EdgeGeomLocation *vtxLoc
)
```

Arguments

<i>ctx</i>	A pointer to the context.
<i>gcmCtx</i>	The context into which the commands are placed. See the <i>libgcm Reference</i> for further information.
<i>vtxLoc</i>	Provides the vertexes locations so the command buffer hole can be correctly filled.

Return Values

None.

Description

Inserts commands, into the specified command context, that set the vertex data formats and offsets for the RSX™. If the output vertex format specified in the [EdgeGeomSpuConfigInfo](#) is not one of the predefined formats and an [EdgeGeomSetVertexDataArraysCallback\(\)](#) has been provided in the [EdgeGeomVertexStreamDescription](#) structure, the callback should fill in the vertex data commands.

See Also

[EdgeGeomLocation](#), [EdgeGeomSetVertexDataArraysCallback](#), [edgeGeomOutputVertexes](#), [EdgeGeomVertexStreamDescription](#)

edgeGeomEndCommandBufferHole

Fills the command buffer hole and uses DMA to write the results to the corresponding buffer in main memory.

Definition

```
#include <edgegeom.h>
void edgeGeomEndCommandBufferHole (
    EdgeGeomSpuContext *ctx,
    CellGcmContextData *gcmCtx,
    uint32_t holeEa,
    EdgeGeomAllocationInfo *infos,
    uint32_t numInfos
)
```

Arguments

<i>ctx</i>	A pointer to the context.
<i>gcmCtx</i>	The context where the commands that fill the command buffer hole are stored.
<i>holeEa</i>	Effective address to the command buffer to fill.
<i>infos</i>	An array of allocation information. Each allocation must be specified as a separate entry in the array. This can be NULL when not using ring buffering.
<i>numInfos</i>	The number of elements in the <i>infos</i> array. This can be zero when not using ring buffering.

Return Values

None.

Description

Fills the command buffer hole in the SPU local store, writing a Skip NOP for the unused space. Uses DMA to write the completed command buffer to the corresponding space in main memory. If ring buffers are used, this function also writes the commands to update the RSX™ labels for each of the elements in the [EdgeGeomAllocationInfo](#) array after the output data has been used.

See Also

[EdgeGeomSpuConfigInfo](#), [EdgeGeomAllocationInfo](#), [edgeGeomAllocateOutputSpace](#)

Callback Functions

EdgeGeomGetOutputVertexStrideCallback

Callback function that provides size of stride for custom output vertex format.

Definition

```
#include <edgegeom_structs.h>
typedef uint32_t (*EdgeGeomGetOutputVertexStrideCallback) (
    EdgeGeomSpuContext *ctx,
    uint32_t outputFormatId
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>outputFormatId</i>	A custom output format ID

Return Values

Stride size of the specified RSX™ vertex format

Description

[EdgeGeomGetOutputVertexStrideCallback\(\)](#) is called whenever the function [edgeGeomGet\(\)](#) encounters a custom output format.

Use the [EdgeGeomVertexStreamDescription](#) structure to set this function pointer.

See Also

[edgeGeomGet](#)

EdgeGeomDecompressVertexStreamCallback

Callback function that decompresses vertex data stream with specified custom input vertex format.

Definition

```
#include <edgegeom_structs.h>
typedef uint32_t (*EdgeGeomDecompressVertexStreamCallback) (
    EdgeGeomSpuContext *ctx,
    const void *vertexes,
    uint32_t numVertexes,
    const void *fixedOffsets,
    uint32_t inputFormatId
)
```

Arguments

<i>ctx</i>	A pointer to the context.
<i>vertexes</i>	Pointer to the input vertex data stream to be decompressed.
<i>numVertexes</i>	Number of vertexes.
<i>fixedOffsets</i>	Pointer to table of fixed point offsets used during decompression of the vertex stream. Can be NULL if there are no fixed point attributes in the stream.
<i>inputFormatId</i>	The stream's vertex format ID.

Return Values

Not used.

Description

This callback function is called from [edgeGeomDecompressVertexes\(\)](#) function if the input vertex data stream is compressed using a custom input vertex format.

Use the [EdgeGeomVertexStreamDescription](#) structure to set this function pointer.

See Also

[edgeGeomDecompressVertexes](#)

EdgeGeomBlendVertexStreamCallback

Callback function that performs blend shape operation with a custom vertex delta format.

Definition

```
#include <edgegeom_structs.h>
typedef uint32_t (*EdgeGeomBlendVertexStreamCallback) (
    EdgeGeomSpuContext *ctx,
    const void *vertexes,
    uint32_t numVertexes,
    const void *fixedOffsets,
    uint32_t deltaFormatId,
    float alpha
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>vertexes</i>	Pointer to input vertex delta stream, to be blended with the previously decompressed vertex data
<i>numVertexes</i>	Number of vertexes
<i>fixedOffsets</i>	Pointer to the fixed offsets buffer used to data decompression
<i>deltaFormatId</i>	A custom vertex delta format ID
<i>alpha</i>	The percent to blend in this blend shape

Return Values

Not used.

Description

This callback function is called from the function [edgeGeomProcessBlendShapes\(\)](#) to perform blend shape operation with a custom vertex delta format. The result data must be written into uniform tables obtained by calling [edgeGeomGetUniformTables\(\)](#).

Use the [EdgeGeomVertexStreamDescription](#) structure to set this function pointer.

See Also

[edgeGeomProcessBlendShapes](#)

EdgeGeomCompressVertexStreamCallback

Callback function that compresses vertex data into a custom output format.

Definition

```
#include <edgegeom_structs.h>
typedef uint32_t (*EdgeGeomCompressVertexStreamCallback) (
    EdgeGeomSpuContext *ctx,
    uint32_t numVertexes,
    void *outVertexes,
    uint32_t outputFormatId
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>numVertexes</i>	Number of vertexes
<i>outVertexes</i>	A free space in the I/O buffer for outputting compressed data
<i>flavor</i>	A custom output vertex format ID

Return Values

Not used.

Description

This callback function is called from function [edgeGeomCompressVertexes\(\)](#) to compress data in vertex uniform tables to a custom output format.

Use the [EdgeGeomVertexStreamDescription](#) structure to set this function pointer.

See Also

[edgeGeomCompressVertexes](#), [EdgeGeomVertexStreamDescription](#)

EdgeGeomSetVertexDataArraysCallback

Callback function that generates command buffer for custom output flavor.

Definition

```
#include <edgegeom_structs.h>
typedef uint32_t (*EdgeGeomSetVertexDataArraysCallback) (
    EdgeGeomSpuContext *ctx,
    uint32_t outputFormatId,
    CellGcmContextData *gcmCtx,
    uint32_t outputLocation,
    uint32_t vertexOffset
)
```

Arguments

<i>ctx</i>	A pointer to the context.
<i>outputFormatId</i>	A custom vertex output format ID.
<i>gcmCtx</i>	The context into which the commands are placed. See the <i>libgcm Reference</i> for further information.
<i>outputLocation</i>	The location of final vertex data CELL_GCM_LOCATION_MAIN or CELL_GCM_LOCATION_LOCAL.
<i>vertexOffset</i>	The I/O offset of the start address of vertex data on main/local memory.

Return Values

Not used.

Description

This callback function is called from function [edgeGeomSetVertexDataArrays\(\)](#) to generate the command buffer for a custom output vertex data format.

This can be accomplished by calling `cellGcmSetVertexDataArray()` for each attribute.

Use the [EdgeGeomVertexStreamDescription](#) structure to set this function pointer.

Notes

The command buffer that is output by this function is transferred via DMA to main memory to fill the command-buffer hole. So be careful about the way the command buffer is generated.

For more information about the command-buffer hole, see the *PlayStation®Edge Library Overview*.

See Also

[edgeGeomBeginCommandBufferHole](#), [edgeGeomEndCommandBufferHole](#),
[edgeGeomSetVertexDataArrays](#), [EdgeGeomVertexStreamDescription](#)

EdgeGeomTransformVertexesForCullCallback

Callback function to transform vertex positions prior to triangle culling.

Definition

```
#include <edgegeom.h>
typedef void (*EdgeGeomTransformVertexesForCullCallback) (
    EdgeGeomSpuContext *ctx,
    void *userData
)
```

Arguments

<i>ctx</i>	A pointer to the context
<i>userData</i>	A pointer to user data

Return Values

None.

Description

If provided, [EdgeGeomTransformVertexesForCullCallback\(\)](#) will be called in place of the default vertex transformation function, [edgeGeomTransformVertexes\(\)](#).

The culling function expects the transformed vertices to have the following format:

- x component – transformed x coordinate
- y component – transformed y coordinate
- z component – vertex frustum test results. Bit 31-26: less than min z, less than min x, less than min y, greater than max x, greater than max y, greater than max z.
- w component – Bit 32: set if transformed w is positive. Bit 30-16: quantized x. bit 15-0: quantized y.

Constants

Triangle Culling Mode

Constants expressing the various different combinations of triangle culling tests to perform.

Definition

Macro	Value	Description
EDGE_GEOM_CULL_NONE	0	Do not perform triangle culling.
EDGE_GEOM_CULL_FRUSTUM	1	Perform frustum and pixel center triangle culling.
EDGE_GEOM_CULL_BACKFACES_AND_FRUSTUM	2	Perform back face, frustum, and pixel center triangle culling.
EDGE_GEOM_CULL_FRONTFACES_AND_FRUSTUM	3	Perform front face, frustum, and pixel center triangle culling.

Description

This is passed to [edgeGeomCullTriangles\(\)](#) in the *cullingFlavor* argument.

Skinning Mode

Constants expressing the various different types of matrix palette skinning to perform.

Definition

Macro	Value	Description
EDGE_GEOM_SKIN_NONE	0	Do not perform skinning.
EDGE_GEOM_SKIN_NO_SCALING	1	Unit matrix skinning.
EDGE_GEOM_SKIN_UNIFORM_SCALING	2	Perform skinning.
EDGE_GEOM_SKIN_NON_UNIFORM_SCALING	3	Perform skinning and compute cofactor matrices for normal transformation.
EDGE_GEOM_SKIN_SINGLE_BONE_NO_SCALING	4	Single bone unit matrix skinning.
EDGE_GEOM_SKIN_SINGLE_BONE_UNIFORM_SCALING	5	Single bone skinning.
EDGE_GEOM_SKIN_SINGLE_BONE_NON_UNIFORM_SCALING	6	Single bone skinning. Computes cofactor matrices for normal transformation.

Description

Used by *indexesFlavorAndSkinningFlavor* in [EdgeGeomSpuConfigInfo](#).

Index Data Type

Constants expressing the various different types of index data that is used in the segment.

Definition

Macro	Value	Description
EDGE_GEOM_INDEXES_U16_TRIANGLE_LIST_CW	0	16-bit index data with clockwise triangle ordering
EDGE_GEOM_INDEXES_U16_TRIANGLE_LIST_CCW	1	16-bit index data with counter-clockwise triangle ordering
EDGE_GEOM_INDEXES_SW_TRIANGLE_LIST_CW	2	Software-compressed index data with clockwise triangle ordering
EDGE_GEOM_INDEXES_SW_TRIANGLE_LIST_CCW	3	Software-compressed index data with counter-clockwise triangle ordering

Description

Used by *indexesFlavorAndSkinningFlavor* in [EdgeGeomSpuConfigInfo](#).

Skinning Matrix Format

Constants expressing the supported skinning matrix formats.

Definition

Macro	Value	Description
EDGE_GEOM_MATRIX_3x4_ROW_MAJOR	0	This is Edge's native matrix type, and the most efficient in terms of memory usage. It is the top three rows of a "DirectX-style" 4x4 matrix; the fourth row is always [0,0,0,1] and does not need to be stored explicitly.
EDGE_GEOM_MATRIX_4x4_ROW_MAJOR	1	Specifies "DirectX-style" row-major 4x4 matrices. This is provided primarily for convenience; obviously, 4x4 matrices use 33% more memory than 3x4 matrices.
EDGE_GEOM_MATRIX_4x4_COLUMN_MAJOR	2	Specifies "OpenGL-style" column-major 4x4 matrices. This is provided primarily for convenience; obviously, 4x4 matrices use 33% more memory than 3x4 matrices.

Description

Used by *skinningMatrixFormat* in [EdgeGeomSpuConfigInfo](#).

Geometry Configuration Flags

Constants expressing the various different configurations of the geometry pipeline. These flags are stored in the *flagsAndUniformTableCount* member of [EdgeGeomSpuConfigInfo](#).

Definition

Macro	Value	Description
EDGE_GEOM_FLAG_STATIC_GEOMETRY_FAST_PATH	0x10	This indicates that the input geometry will not be modified by the SPU job, which allows certain Edge functions to return immediately without performing unnecessary work.
EDGE_GEOM_FLAG_INCLUDES_EXTRA_UNIFORM_TABLE	0x80	This flag must be set by the tools if the segment will use either culling or custom blend shape flavors. Both of these operations require an additional uniform table to store temporary data, and the size of this table must be accounted for in the Edge partitioner when the segment is created.

Description

Used by the upper four bits of *flagsAndUniformTableCount* in [EdgeGeomSpuConfigInfo](#).

Output Mode

Constants indicating use of static output buffer in main or video memory. These should be masked into the output buffer info effective address passed into [edgeGeomAllocateOutputSpace\(\)](#).

Definition

Macro	Value	Description
EDGE_GEOM_DIRECT_OUTPUT_TO_LOCAL_MEMORY	0x2	This tells Edge to output directly to the address in video memory given to edgeGeomAllocateOutputSpace() .
EDGE_GEOM_DIRECT_OUTPUT_TO_MAIN_MEMORY	0x3	This tells Edge to output directly to the address in main memory given to edgeGeomAllocateOutputSpace() .

Description

This exists in [EdgeGeomSpuConfigInfo\(\)](#).