

PlayStation®Edge Geometry Library for Offline Tool: Reference

Table of Content

Preface.....	4
About This Document.....	5
Data Types for Tools.....	6
EdgeGeomSegmentFormat	7
EdgeGeomScene	9
EdgeGeomSegment.....	12
EdgeGeomKCacheOptimizerUserData.....	15
EdgeGeomKCacheOptimizerHillclimberUserData	16
EdgeGeomPartitionerInput.....	17
EdgeGeomPartitionElementCounts	19
EdgeGeomPartitionerOutput.....	20
EdgeGeomSpuVertexAttributeDefinition	21
EdgeGeomSpuVertexFormat	23
EdgeGeomRsxVertexAttributeDefinition	24
EdgeGeomRsxVertexFormat	25
libedgegeomtool Functions.....	26
edgeGeomPartitioner	27
edgeGeomAlloc.....	28
edgeGeomFree	29
edgeGeomSetAllocFunc	30
edgeGeomSetFreeFunc.....	31
edgeGeomGetCommandBufferHoleSize	32
edgeGeomGetScratchBufferSizeInQwords	33
edgeGeomGetSpuVertexFormat.....	34
edgeGeomGetRsxVertexFormat.....	35
edgeGeomGetSpuVertexFormatId.....	36
edgeGeomGetRsxVertexFormatId.....	37
edgeGeomSpuVertexFormatIsValid	38
edgeGeomRsxVertexFormatIsValid	39
edgeGeomGetSpuVertexAttributeSize.....	40
edgeGeomGetRsxVertexAttributeSize.....	41
edgeGeomGetAttributeSlotIndex	42
edgeGeomSetAttributeSlotIndex.....	43
edgeGeomComputeTriangleCentroids	44
edgeGeomGetBlendedVertexes	45
edgeGeomBlendShapeAffectsSegment	47
edgeGeomMakeSpuConfigInfo	49
edgeGeomMakeIndexBuffer	51
edgeGeomMakeSpuVertexBuffer.....	52
edgeGeomMakeRsxVertexBuffer.....	54
edgeGeomMakeSkinningBuffer	56
edgeGeomMakeBlendShapeBuffer	58
edgeGeomMergeIdenticalVertexes.....	60
edgeGeomPartitionSceneIntoSegments.....	61

edgeGeomFreeSegmentData	63
edgeGeomTriangulatePolygons	64
edgeGeomMakeSpuStreamDescription	65
edgeGeomMakeRsxStreamDescription	66
edgeGeomKCacheOptimizer	67
edgeGeomKCacheOptimizerHillclimber	69
Callback Functions	70
EdgeGeomVertexCacheOptimizerFunc	71
EdgeGeomCustomPartitionDataSizeFunc	72
Enumerations	73
EdgeGeomIndexesFlavor	74
EdgeGeomSkinningFlavor	75
EdgeGeomMatrixFormat	76
EdgeGeomCullingFlavor	77

Preface

About This Document

Purpose

This document provides an API reference for the offline-tool geometry component of the Edge library, libedgegeomtool. Integrate this component into your asset-pipeline to efficiently manage the data and to maintain high performance in the runtime.

The Edge geometry compiler tool, edgegeomcompiler, is a sample tool that demonstrates how libedgegeomtool can be used in an asset pipeline. Specifically, it demonstrates how to take data in COLLADA™ format and pass it through the Edge library, producing a binary file that can be loaded into the runtime sample. Due to the usage of FCollada, edgeanimcompiler now only supports Windows.

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 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 Tools

EdgeGeomSegmentFormat

Container structure for all data pertaining to the format of the segments generated by the Edge tools.

Definition

```
#include <libedgegeomtool_wrap.h>
struct EdgeGeomSegmentFormat
{
    EdgeGeomSpuVertexFormat *m_spuInputVertexFormats[2];
    EdgeGeomSpuVertexFormat *m_spuInputVertexDeltaFormat;
    EdgeGeomRsxVertexFormat *m_spuOutputVertexFormat;
    EdgeGeomRsxVertexFormat *m_rsxOnlyVertexFormat;
    EdgeGeomSkinningFlavor m_skinType;
    EdgeGeomIndexesFlavor m_indexesType;
    EdgeGeomMatrixFormat m_skinMatrixFormat;
};
```

Members

<i>m_spuInputVertexFormats</i>	Pointers to the vertex formats of the two SPU input vertex streams. The first entry must always exist; the second can be NULL if there will only be one SPU input stream.
<i>m_spuInputVertexDeltaFormat</i>	Pointer to the vertex format to be used by blend shape vertex delta streams. Can be NULL if the scene contains no blend shape data.
<i>m_spuOutputVertexFormat</i>	Pointer to the vertex format of the SPU output vertex stream. Required to be non-NULL.
<i>m_rsxOnlyVertexFormat</i>	Pointer to the vertex format of the segment's RSX™-only vertex stream. This stream should contain attributes that will not be processed by the SPU.
<i>m_skinType</i>	Defines the skinning algorithm that will be used for the scene. For maximum efficiency, choose the simplest algorithm that can correctly be applied to your scene data.
<i>m_indexesType</i>	Determines the format of the runtime input index/triangle stream.
<i>m_skinMatrixFormat</i>	Determines the format of skinning matrices, if skinning is enabled.

Description

This structure bundles together all the information needed to convert a raw geometry data (such as an [EdgeGeomScene](#) object) into a format suitable for processing by the Edge Geometry runtime.

The following values can be set for *m_indexesType*.

Enum	Value	Description
kIndexesU16TriangleListCW	0	16-bit indexed triangle list with clockwise ordering
kIndexesU16TriangleListCCW	1	16-bit indexed triangle list with counter-clockwise ordering
kIndexesCompressedTriangleListCW	2	Compressed indexed triangle list with clockwise ordering (SPU only)
kIndexesCompressedTriangleListCCW	3	Compressed indexed triangle list with counter-clockwise ordering (SPU only)

The following values can be set to *m_skinMatrixFormat*.

Enum	Value	Description
kMatrix3x4RowMajor	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.
kMatrix4x4RowMajor	1	Specifies "DirectX-style" row-major 4x4 matrices. This is provided primarily for convenience; obviously, 4x4 matrices use 33% more memory than 3x4 matrices.
kMatrix4x4ColumnMajor	2	Specifies "OpenGL-style" column-major 4x4 matrices. This is provided primarily for convenience; obviously, 4x4 matrices use 33% more memory than 3x4 matrices.

The caller is responsible for all memory management associated with this structure.

See Also

[EdgeGeomScene](#), [EdgeGeomSegment](#)

EdgeGeomScene

Container structure for all data associated with the input scene loaded by the Edge Geometry tools, which will be partitioned and converted into runtime segment data.

Definition

```
#include <libedgegeomtool_wrap.h>
struct EdgeGeomScene
{
    // Triangle data
    uint32_t m_numTriangles;
    uint32_t *m_triangles;
    int32_t *m_materialIdPerTriangle;

    // Main vertex data
    uint32_t m_numVertexes;
    uint32_t m_numFloatsPerVertex;
    float m_vertexes;
    uint8_t m_numVertexAttributes;
    uint16_t *m_vertexAttributeIndexes;
    EdgeGeomAttributeId *m_vertexAttributeIds;

    //Blend shape data
    uint32_t m_numBlendShapes;
    uint32_t m_numFloatsPerDelta;
    float *m_vertexDeltas;
    uint8_t m_numBlendedAttributes;
    uint16_t *m_blendedAttributeIndexes;
    EdgeGeomAttributeId *m_blendedAttributeIds;

    // Skinning data
    int32_t *m_matrixIndexesPerVertex;
    float *m_skinningWeightsPerVertex;
};
```

Members

<i>m_numTriangles</i>	The number of triangles in the scene.
<i>m_triangles</i>	The scene's triangle data, represented as three 32-bit integers per triangle. The array must contain at least <i>m_numTriangles*3</i> entries.
<i>m_materialIdPerTriangle</i>	An array of material IDs for each triangle in the scene. A material ID can be anything – a pointer to a shader, a hash of the shader's name, a simple integer index – as long as it is unique across all materials. The array must contain at least <i>m_numTriangles</i> entries.
<i>m_numVertexes</i>	The number of vertexes in the scene.
<i>m_numFloatsPerVertex</i>	The stride (in 32-bit floats) of each vertex in the <i>m_vertexes</i> array.
<i>m_vertexes</i>	The scene's vertex data, represented as an interleaved stream of vertex attributes such that each vertex is presented as a contiguous block of data. All attributes are represented here in full 32-bit precision. The array must contain at least (<i>m_numVertexes</i> * <i>m_numFloatsPerVertex</i>) entries.
<i>m_numVertexAttributes</i>	The number of vertex attributes in each vertex in the primary vertex stream.

<code>m_vertexAttributeIndexes</code>	Array of indexes of all vertex attribute within each vertex's block of float data within <code>m_vertexes</code> . The array must contain at least <code>m_numVertexAttributes</code> entries, which must be listed in the same order as the entries of the <code>m_vertexAttributeIds</code> array.
<code>m_vertexAttributeIds</code>	Array of attribute IDs for all vertex attributes. The array must contain at least <code>m_numVertexAttributes</code> entries, which must be listed in the same order as the entries of the <code>m_vertexAttributeIndexes</code> array.
<code>m_numBlendShapes</code>	The number of blend shapes in the scene. If zero, the following blend-shape-related values and arrays can be left undefined.
<code>m_numFloatsPerDelta</code>	The stride (in 32-bit floats) of each vertex delta in the <code>m_vertexDeltas</code> array.
<code>m_vertexDeltas</code>	The scene's blend shape data, represented as an interleaved stream of vertex attributes such that each delta is presented as a contiguous block of data. Deltas for all vertexes in the scene are provided for each blend shape in a contiguous block of memory (with <code>m_numVertexes*m_numFloatsPerDelta</code> entries per shape). All attributes are represented here in full 32-bit precision. The array must contain at least $(m_numBlendShapes * m_numVertexes * m_numFloatsPerDelta)$ entries.
<code>m_numBlendedAttributes</code>	The number of vertex attributes in each vertex delta in the blend shape stream. Not all attributes in the main scene must be blended, but each attribute in the delta stream <i>must</i> be present in the main vertex stream.
<code>m_blendedAttributeIndexes</code>	Array of indexes of all blended vertex attribute within each vertex delta's block of float data within <code>m_vertexDeltas</code> . The array must contain at least <code>m_numBlendedAttributes</code> entries, which must be listed in the same order as the entries of the <code>m_blendedAttributeIds</code> array.
<code>m_blendedAttributeIds</code>	Array of attribute IDs for all blended vertex attributes. The array must contain at least <code>m_numBlendedAttributes</code> entries, which must be listed in the same order as the entries of the <code>m_blendedAttributeIndexes</code> array.
<code>m_matrixIndexesPerVertex</code>	Contains the indexes into the scene's bone array of the four skinning bone matrices that influence this vertex. If a vertex has fewer than four bone influences, the remaining indexes should be set to -1. The array must have <code>m_numVertexes*4</code> entries.
<code>m_skinningWeightsPerVertex</code>	Contains the weights of each vertex's four skinning bones. Each weight ranges from 0.0 to 1.0, and a vertex's four weights must sum to exactly 1.0. The weights for unused bones should be set to 0.0. The array must have <code>m_numVertexes*4</code> entries.

Description

This structure bundles together all of the input scene's geometry data to the Edge tools, including triangles, vertexes, blend shapes, and skinning data. The user is responsible for all memory management associated with this structure.

A properly formatted example of triangle data, showing three triangles and two materials, is as follows:

`m_triangles:`

0	1	2	2	1	3	3	2	4
---	---	---	---	---	---	---	---	---

m_materialIdPerTriangle:

0	0	1
---	---	---

m_numTriangles:

3

The storage format for *m_vertexes* is as follows, where the same number indicates the same vertex and the slices within each vertex represent the vertex's attributes:

0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	5	5	5	5	.
																								.
																								.

The storage format for *m_vertexDeltas* is similar, except that there will be one complete set of vertex data per blend shape, typically with a different number of elements per vertex than the base vertex attributes array:

0	0	0	1	1	1	2	2	2	2	3	3	3	4	4	4	5	5	5	...
0	0	0	1	1	1	2	2	2	2	3	3	3	4	4	4	5	5	5	...
0	0	0	1	1	1	2	2	2	2	3	3	3	4	4	4	5	5	5	...

EdgeGeomSegment

Container structure for all final runtime data produced by the Edge tools, in PlayStation®3-native big-endian format, ready to write to disk.

Definition

```
#include <libedgegeomtool_wrap.h>
struct EdgeGeomSegment
{
    uint32_t m_materialId;

    uint8_t *m_spuConfigInfo;

    uint8_t *m_indexes;
    uint16_t m_indexesSizes[2];

    uint8_t *m_spuVertexes[2];
    uint16_t m_spuVertexesSizes[6];
    uint32_t m_fixedOffsetsSizes[2];
    uint32_t *m_fixedOffsetPtrs[2];

    uint8_t *m_rsxOnlyVertexes;
    uint32_t m_rsxOnlyVertexesSize;

    uint8_t *m_spuInputStreamDescriptions[2];
    uint16_t m_spuInputStreamDescriptionSizes[2];
    uint8_t *m_spuOutputStreamDescription;
    uint16_t *m_spuOutputStreamDescriptionSize;
    uint8_t *m_rsxOnlyStreamDescription;
    uint16_t *m_rsxOnlyStreamDescriptionSize;

    uint16_t m_skinMatricesByteOffsets[2];
    uint16_t m_skinMatricesSizes[2];
    uint8_t *m_skinIndexesAndWeights;
    uint16_t m_skinIndexesAndWeightsSizes[2];

    uint32_t m_ioBufferSize;
    uint32_t m_scratchSize;

    uint32_t m_numBlendShapes;
    uint16_t *m_blendShapeSizes;
    uint8_t **m_blendShapes;
};
```

Members

<i>m_materialId</i>	The material ID for all the triangles in this segment. This value is drawn from the <i>m_materialIdPerTriangle</i> array passed in as part of the EdgeGeomScene object.
<i>m_spuConfigInfo</i>	Pointer to the segment's <i>EdgeGeomSpuConfigInfo</i> structure.
<i>m_indexes</i>	Pointer to the segment's index buffer.
<i>m_indexesSizes</i>	DMA tag sizes for the index buffer.
<i>m_spuVertexes</i>	Pointer to the segment's primary and secondary SPU input vertex buffers. The second element can be NULL if only one SPU vertex stream is needed.

<i>m_spuVertexesSizes</i>	DMA tag sizes for the SPU input vertex buffers (three entries per vertex stream). Unused entries should be set to zero.
<i>m_fixedOffsetsSizes</i>	Size (in bytes) of the primary and secondary fixed point offset tables. If unused, set to zero.
<i>m_fixedOffsetPtrs</i>	Pointers to the tables of fixed point offsets for each of the two input vertex streams. These values are added fixed-point values to bring them back to their original range. If unused, set to NULL.
<i>m_rsxOnlyVertexes</i>	Pointer to the segment's optional RSX™-only vertex stream. These attributes will not be sent through the SPU's NULL if unused.
<i>m_rsxOnlyVertexesSize</i>	Size (in bytes) of the RSX™-only vertex stream. If unused, this will be zero.
<i>m_spuInputStreamDescriptions</i>	Pointers to the two SPU input vertex stream descriptions. The second entry can be NULL if only one vertex stream is used.
<i>m_spuInputStreamDescription Sizes</i>	Sizes (in bytes) of the two SPU input vertex stream descriptions.
<i>m_spuOutputStreamDescription</i>	Pointer to the SPU output vertex stream description.
<i>m_spuOutputStreamDescription Size</i>	Size (in bytes) of the SPU output vertex stream description.
<i>m_rsxOnlyStreamDescription</i>	Pointer to the stream description for the optional RSX™-only vertex stream. Will only be non-NULL if and only if this segment has an RSX™-only vertex stream.
<i>m_rsxOnlyStreamDescription Size</i>	Size (in bytes) of the RSX™-only stream description. Will be zero if unused.
<i>m_skinMatricesByteOffsets</i>	Byte offsets to the start of the two skinning matrix ranges used by this segment.
<i>m_skinMatricesSizes</i>	Sizes (in bytes) of the two skinning matrix ranges used by this segment.
<i>m_skinIndexesAndWeights</i>	Pointer to this segment's skinning index/weights buffer.
<i>m_skinIndexesAndWeightsSizes</i>	Sizes (in bytes) of the skinning index/weights DMA tags.
<i>m_ioBufferSize</i>	Size (in bytes) of this segment's maximum IO buffer usage on the SPU.
<i>m_scratchSize</i>	Size (in 16-byte quadwords) of this segment's maximum scratch buffer usage on the SPU.
<i>m_numBlendShapes</i>	The number of blend shapes included in this segment. This is the number of blend shapes that could potentially affect this segment, which is probably smaller than the total number of blend shapes in the scene.
<i>m_blendShapeSizes</i>	Array of sizes (in bytes) of this segment's blend shape buffers. The array has <i>m_numBlendShapes</i> entries.
<i>m_blendShapes</i>	Array of pointers to this segment's blend shape buffers. The array has <i>m_numBlendShapes</i> entries.

Description

This structure contains all the data for a single geometry segment. Its member buffers are all properly byte-swapped to big-endian where necessary, and are ready to write to disk (or process immediately, if libedgegeomtool is run on the PPU).

The easiest way to create geometry segments is with the [edgeGeomPartitionSceneIntoSegments\(\)](#) function. Alternatively, the structure's members can be generated by libedgegeomtool functions, as follows:

edgeGeomMakeSpuConfigInfo()	<i>m_spuConfigInfo</i>
edgeGeomMakeIndexBuffer()	<i>m_indexes</i>
	<i>m_indexesSizes</i>
edgeGeomMakeSpuVertexBuffer()	<i>m_spuVertexes</i>
	<i>m_spuVertexesSizes</i>
	<i>m_fixedOffsetsSizes</i>
	<i>m_fixedOffsetPtrs</i>
edgeGeomMakeRsxVertexBuffer()	<i>m_rsxOnlyVertexes</i>
	<i>m_rsxOnlyVertexesSize</i>
edgeGeomMakeSpuStreamDescription()	<i>m_spuInputStreamDescriptions</i>
	<i>m_spuInputStreamDescriptionSizes</i>
edgeGeomMakeRsxStreamDescription()	<i>m_spuOutputStreamDescription</i>
	<i>m_spuOutputStreamDescriptionSize</i>
	<i>m_rsxOnlyStreamDescription</i>
	<i>m_rsxOnlyStreamDescriptionSize</i>
edgeGeomMakeSkinningBuffer()	<i>m_skinMatricesByteOffsets</i>
	<i>m_skinMatricesSizes</i>
	<i>m_skinIndexesAndWeightsSizes</i>
	<i>m_skinIndexesAndWeights</i>
edgeGeomPartitioner()	<i>m_ioBufferSize</i>
edgeGeomGetScratchBufferSizeInQwords()	<i>m_scratchSize</i>
edgeGeomMakeBlendShapeBuffer()	<i>m_numBlendShapes</i>
	<i>m_blendShapeSizes</i>
	<i>m_blendShapes</i>

The caller is responsible for all memory management associated with this structure.

See Also

[edgeGeomPartitionSceneIntoSegments](#)

EdgeGeomKCacheOptimizerUserData

Algorithm-specific data container for the [edgeGeomKCacheOptimizer\(\)](#) function.

Definition

```
#include <libedgegeomtool.h>
struct EdgeGeomKCacheOptimizerUserData
{
    float m_fifoMissCost,
    float m_lruMissCost,
    float m_k1,
    float m_k2,
    float m_k3
};
```

Members

<i>m_fifoMissCost</i>	Cost factor for a single post-transform cache miss. Recommended to be the number of input attributes to the vertex program.
<i>m_lruMissCost</i>	Cost factor for a single mini-cache miss. Recommended to be four times the number of output attributes from the vertex program.
<i>m_k1</i>	Factor controlling the importance of total cost of cache misses in a vertex neighborhood. Recommended 1.0
<i>m_k2</i>	Factor controlling the importance of number of faces rendered in a vertex neighborhood. Recommended 0.25
<i>m_k3</i>	Factor controlling the importance of the final age of the focus vertex in the post-transform cache. Recommended 0.8

Description

libedgegeomtool supports a standard interface for vertex cache optimization functions, so that users may choose an implantation whose performance characteristics (for example, the trade-off between accuracy and running time) best meet their needs. To keep the interface consistent while supporting a wide variety of implementations, each vertex cache optimizer function takes a pointer to arbitrary, function-specific user data.

This structure contains the user data for the [edgeGeomKCacheOptimizer\(\)](#) function.

See Also

[edgeGeomKCacheOptimizer](#), [EdgeGeomPartitionerInput](#)

EdgeGeomKCacheOptimizerHillclimberUserData

Algorithm-specific data container for the [edgeGeomKCacheOptimizer\(\)](#) function.

Definition

```
#include <libedgegeomtool.h>
struct EdgeGeomKCacheOptimizerHillclimberUserData
{
    uint32_t m_numIterations,
    EdgeGeomIndexesFlavor m_indexesType,
    uint32_t m_numRsxInputAttributes,
    uint32_t m_numRsxOutputAttributes
};
```

Members

<i>m_numIterations</i>	The number of iterations of the hill-climber to run. If 0, the kcache optimizer will be run once with default parameters. Setting this value higher than 100 iterations will be very slow, and will not generally result in noticeable improvements.
<i>m_indexesType</i>	Format of the output triangle list. Used to determine if the triangle list is being compressed, so that the optimized triangles should be rotated for maximum compression before being scored.
<i>m_numRsxInputAttributes</i>	The number of vertex attributes being passed as input to the vertex program.
<i>m_numRsxOutputAttributes</i>	The number of attributes passed output by the vertex program.

Description

libedgegeomtool supports a standard interface for vertex cache optimization functions, so that users may choose an implantation whose performance characteristics (for example, the trade-off between accuracy and running time) best meet their needs. To keep the interface consistent while supporting a wide variety of implementations, each vertex cache optimizer function takes a pointer to arbitrary, functions-specific user data.

This structure contains the user data for the [edgeGeomKCacheOptimizerHillclimber\(\)](#) function.

See Also

[edgeGeomKCacheOptimizerHillclimber](#), [EdgeGeomPartitionerInput](#)

EdgeGeomPartitionerInput

Structure that specifies the input parameters for [edgeGeomPartitioner\(\)](#).

Definition

```
#include <libedgegeomtool_partitioner.h>
struct EdgeGeomPartitionerInput
{
    uint32_t m_numTriangles;
    uint32_t *m_triangleList;
    uint32_t m_numInputAttributes;
    uint32_t m_numOutputAttributes;
    uint32_t m_inputVertexStride[2];
    uint32_t m_outputVertexStride;
    EdgeGeomSkinningFlavor m_skinningFlavor;
    EdgeGeomIndexesFlavor m_indexListFlavor;
    EdgeGeomMatrixFormat m_skinningMatrixFormat;

    EdgeGeomVertexCacheOptimizerFunc m_cacheOptimizerFunc;
    void *m_cacheOptimizerUserData;
    EdgeGeomCustomPartitionDataSizeFunc m_customDataSizeFunc;
    float *m_triangleCentroids;
    int32_t *m_skinningMatrixIndexesPerVertex;
    uint32_t m_deltaStreamVertexStride;
    uint32_t *m_blendedVertexIndexes;
    uint32_t m_numBlendedVertexes;
};
```

Members

<i>m_numTriangles</i>	Count of triangles.
<i>m_triangleList</i>	Array of vertex indexes where three consecutive indexes signify a triangle.
<i>m_numInputAttributes</i>	Total number of input vertex attributes (in both primary and secondary SPU input streams).
<i>m_numOutputAttributes</i>	Number of output attributes (those sent from the SPU to the GPU).
<i>m_inputVertexStride</i>	Size (in bytes) of a vertex in each of the two SPU input vertex streams.
<i>m_outputVertexStride</i>	Size (in bytes) of a vertex in the SPU output vertex stream.
<i>m_skinningFlavor</i>	The type of skinning that will be performed.
<i>m_indexListFlavor</i>	The format of the input index buffer.
<i>m_skinningMatrixFormat</i>	The format (and size) of skinning matrices, if skinning is enabled.

<i>m_cacheOptimizerFunc</i>	Pointer to a callback function used to optimize the final partitioned triangle lists for the RSX™ vertex cache. Edge provides two optimizers (edgeGeomKCacheOptimizer() and edgeGeomKCacheOptimizerHillclimber()), but other implementations (such as <i>NvTriStrip</i>) could easily be supported. If NULL, no vertex cache optimization will be performed.
<i>m_cacheOptimizerUserData</i>	This pointer will be passed to the <i>m_cacheOptimizerFunc</i> as its <i>userData</i> argument. Its interpretation depends entirely on the cache optimizer in question.
<i>m_customDataSizeFunc</i>	Pointer to an optional callback function used to provide the size of any custom non-Edge job data that will be taking up space in the partition. The callback's argument contains a collection of element counts of the partition currently being built, so that accurate custom data buffer sizes can be estimated. Note that this function will be called extremely frequently, and should therefore avoid any length O(N)-type calculations. If NULL, no space for custom data will be reserved in the partition.
<i>m_triangleCentroids</i>	Array of floats parallel to the <i>m_triangleList</i> where three components (x,y,z) indicate the position of the triangle's centroid (the average of its three corners). This member is optional (NULL if unused), but provides for better spatially compact partitions if available.
<i>m_skinningMatrixIndexesPerVertex</i>	Array of four skinning bone indexes per vertex. Unused bones should be set to -1. This member can be NULL for unskinned geometry.
<i>m_deltaStreamVertexStride</i>	Size (in bytes) of a vertex delta in the blend shape stream. Can be 0 if no blend shapes are needed.
<i>m_blendedVertexIndexes</i>	Array of all vertex indexes in the scene that have non-zero blendshape deltas. If non-NULL, the partitioner will use this data to avoid mixing blended and unblended vertexes in the same partition.
<i>m_numBlendedVertexes</i>	Number of elements in the <i>m_blendedVertexIndexes</i> array.

Description

This structure includes all the parameters for the [edgeGeomPartitioner\(\)](#) function.

See Also

[edgeGeomPartitioner](#), [EdgeGeomPartitionerOutput](#), [edgeGeomComputeTriangleCentroids](#), [edgeGeomGetBlendedVertexes](#), [edgeGeomKCacheOptimizer](#), [edgeGeomKCacheOptimizerHillclimber](#), [EdgeGeomKCacheOptimizerUserData](#), [EdgeGeomKCacheOptimizerHillclimberUserData](#)

EdgeGeomPartitionElementCounts

Structure describing the current state of a partition-in-progress; used in user callbacks to compute the size of custom, non-Edge per-partition data buffers.

Definition

```
#include <libedgegeomtool.h>
struct EdgeGeomPartitionElementCounts
{
    uint32_t m_numTriangles;
    uint32_t m_numVertexes;
    uint32_t m_numMatrices;
    uint32_t m_numBlendedVertexes;
};
```

Members

<i>m_numTriangles</i>	Number of triangles currently assigned to this partition.
<i>m_numVertexes</i>	Number of vertexes currently assigned to this partition.
<i>m_numMatrices</i>	Number of skinning matrices currently assigned to this partition. This will be zero if skinning is not enabled for the partitioner's input data.
<i>m_numBlendedVertexes</i>	Number of triangles currently assigned to this partition. This will be zero if no blend shapes were passed into the partitioner.

Description

This is primarily an internal structure used by the partitioner to keep track of the contents of the partition that is currently being built. The only place a user will encounter it is in an [EdgeGeomCustomPartitionDataSizeFunc\(\)](#) callback function, where a pointer to the structure is passed as a parameter.

See Also

[EdgeGeomPartitionerInput](#)

EdgeGeomPartitionerOutput

Structure containing the resultant partitions output from the [edgeGeomPartitioner\(\)](#) function.

Definition

```
#include <libedgegeomtool_partitioner.h>
struct EdgeGeomPartitionerOutput
{
    uint32_t m_numPartitions;
    uint32_t *m_numTrianglesPerPartition;
    uint32_t *m_triangleListOut;
    uint32_t *m_originalVertexIndexesPerPartition;
    uint32_t *m_numUniqueVertexesPerPartition;
    uint32_t *m_ioBufferSizePerPartition;
};
```

Members

<i>m_numPartitions</i>	Number of partitions in output structure.
<i>m_numTrianglesPerPartition</i>	Array of triangle counts per partition. The array has <i>m_numPartitions</i> elements.
<i>m_triangleListOut</i>	Array of triangles, three vertex indexes per triangle, in a single packed structure. Partition 0's triangles come first, followed by partition 1, 2, and so forth. The total number of elements in the array is the sum of the values in the <i>m_numTrianglesPerPartition</i> array.
<i>m_originalVertexIndexesPerPartition</i>	Array containing a mapping from each partition-local vertex to its index in the original vertex array. The tables from all partitions are laid out end-to-end. The total number of elements is the sum of the values in the <i>m_numUniqueVertexesPerPartition</i> array.
<i>m_numUniqueVertexesPerPartition</i>	Array containing the number of unique vertexes in each partition. The array has <i>m_numPartitions</i> elements.
<i>m_ioBufferSizePerPartition</i>	Array of <i>ioBufferSize</i> per partition, indicating the amount of memory to reserve when sending the partition to the SPU via DMA. The array has <i>m_numPartitions</i> elements.

Description

This is the output structure from the [edgeGeomPartitioner\(\)](#) function. To reduce memory fragmentation, the arrays it contains are allocated as single flat blocks of memory that contain the data for all partitions, one after the other.

The caller is responsible for freeing the array members using [edgeGeomFree\(\)](#).

See Also

[EdgeGeomPartitionerInput](#)

EdgeGeomSpuVertexAttributeDefinition

Structure used to define the format of a single vertex attribute sent as input into the SPU.

Definition

```
#include <libedgegeomtool.h>
struct EdgeGeomSpuVertexAttributeDefinition
{
    EdgeGeomInputAttributeType m_type;
    uint32_t m_count;
    EdgeGeomAttributeId m_attributeId;
    uint32_t m_byteOffset;
    uint32_t m_fixedPointBitDepthInteger[4];
    uint32_t m_fixedPointBitDepthFractional[4];
};
```

Members

<i>m_type</i>	Data type to convert these attributes to.
<i>m_count</i>	Count of values of <i>m_type</i> in this attribute.
<i>m_attributeId</i>	A numerical value that uniquely identifies this attribute, from the <code>EDGE_GEOM_ATTRIBUTE_ID_*</code> enumeration.
<i>m_byteOffset</i>	Specifies the byte in the input buffer where the decompression routine should begin unpacking this attribute.
<i>m_fixedPointBitDepthInteger</i>	Indicates the number of bits used to represent the integer portion of fixed point conversion. Only used for <i>m_type</i> == <code>kSpuAttr_FixedPoint</code> .
<i>m_fixedPointBitDepthFractional</i>	Indicates the number of bits used to represent the fractional portion of fixed point conversion. Only used for <i>m_type</i> == <code>kSpuAttr_FixedPoint</code> .

The following values can be set for *m_type*:

Enum	Value	Description
<code>kSpuAttr_I16N</code>	1	Normalized value $[-1..1]$ in $1/65535^{\text{th}}$ intervals
<code>kSpuAttr_F32</code>	2	Floating point 32-bit
<code>kSpuAttr_F16</code>	3	Floating point 16-bit: 1 sign/5 exponent/11 mantissa
<code>kSpuAttr_U8N</code>	4	Normalized value $[0..1]$ in $1/255^{\text{th}}$ intervals
<code>kSpuAttr_I16</code>	5	Signed 16-bit
<code>kSpuAttr_X11Y11Z10N</code>	6	32 bit, 3-component normalized
<code>kSpuAttr_U8</code>	7	Unsigned 8-bit
<code>kSpuAttr_FixedPoint</code>	8	Fixed point format, user-specified (SPU only; see restriction note in the Description section below)
<code>kSpuAttr_UnitVector</code>	9	Packed 24-bit, 3-component normalized (SPU only)

Description

This structure defines precisely how the tools will compress a specific attribute worth of data for the SPU. A single attribute that could be described by this structure, for example, would be vertex position. A single flavor contains one or more of these structures to indicate multiple attributes packed into an interleaved attribute buffer.

Note important restriction:

$(m_fixedPointBitDepthInteger + m_fixedPointBitDepthFractional) * m_count$
must be evenly divisible by 8, so the attribute is exactly byte aligned. In addition, the total number of bits (integer and fractional) for each component must not be greater than 32.

See Also

[EdgeGeomRsxVertexFormat](#)

EdgeGeomSpuVertexFormat

Structure that defines a complete input stream format sent to the SPU.

Definition

```
#include <libedgegeomtool.h>
struct EdgeGeomSpuVertexFormat
{
    uint32_t m_numAttributes;
    uint32_t m_vertexStride;
    EdgeGeomSpuVertexAttributeDefinition m_attributeDefinition[16];
};
```

Members

<i>m_numAttributes</i>	Count of EdgeGeomSpuVertexAttributeDefinition structures in <i>m_attributeDefinition</i> array. Must be <= 16
<i>m_vertexStride</i>	Number of bytes per vertex when compressed to this format
<i>m_attributeDefinition</i>	Array of EdgeGeomSpuVertexAttributeDefinition structures that describe the order and compression method per vertex attribute

Description

The [EdgeGeomSpuVertexFormat](#) structure is used in the tools to guide the compilation of geometry in two ways. First, it is used (indirectly) by the [edgeGeomPartitioner\(\)](#) function to measure the SPU memory requirements of each partition as it is built. Later, the structure is used during the vertex and index buffer generation process to control the attribute type conversion to compressed data formats.

The [EdgeGeomSpuVertexFormat](#) structure is used to describe both SPU input vertex streams and vertex delta streams (when using blend shapes). When describing vertex deltas, additional restrictions apply. Edge vertex deltas are stored in a subtracted form, so fixed point can be used to compress effectively with fewer bits. However, data formats with a maximum range of $[-1. . 1]$ (such as unit vectors and X11Y11Z10N) are not typically meaningful: two opposing unit normals, when subtracted, result in a delta that has a magnitude greater than 1.0. It is also almost always an error to use the `kSpuAttr_UnitVector` attribute type in a vertex delta format, as its magnitude is always 1.0.

The helper function [edgeGeomSpuVertexFormatIsValid\(\)](#) is provided to test a format for validity.

See Also

[EdgeGeomSpuVertexAttributeDefinition](#), [edgeGeomSpuVertexFormatIsValid](#)

EdgeGeomRsxVertexAttributeDefinition

Structure used to define the format of a single vertex attribute, sent as input to the RSX™ (either from the PPU or the SPU).

Definition

```
#include <libedgegeomtool.h>
struct EdgeGeomRsxVertexAttributeDefinition
{
    EdgeGeomOutputAttributeType m_type;
    uint32_t m_count;
    uint32_t m_byteOffset;
    EdgeGeomAttributeId m_attributeId;
};
```

Members

<i>m_type</i>	The data type of this attribute.
<i>m_count</i>	Count of values of <i>m_type</i> in this attribute.
<i>m_byteOffset</i>	Specifies the byte of this attribute within each vertex.
<i>m_attributeId</i>	A numerical value that uniquely identifies this attribute, from the EDGE_GEOM_ATTRIBUTE_ID_* enumeration.

The following values can be set for *m_type*:

Enum	Value	Description
kRsxAttr_I16N	1	Normalized value [-1 . . 1] in 1/65535 th intervals
kRsxAttr_F32	2	Floating point 32-bit
kRsxAttr_F16	3	Floating point 16-bit: 1 sign/5 exponent/11 mantissa
kRsxAttr_U8N	4	Normalized value [0 . . 1] in 1/255 th intervals
kRsxAttr_I16	5	Signed 16-bit
kRsxAttr_X11Y11Z10N	6	Packed 32 bit, 3-component normalized
kRsxAttr_U8	7	Unsigned 8-bit

Description

This structure defines precisely a specific vertex attribute's data in a renderable format that is ready for consumption by the RSX™. A single attribute that could be described by this structure, for example, would be vertex position.

The SPU-only unit vectors and fixed point compression types are not available for output conversion, because the RSX™ cannot decode them. Only RSX™-native types are supported for output.

See Also

[EdgeGeomRsxVertexFormat](#)

EdgeGeomRsxVertexFormat

Structure that defines a vertex stream format suitable for consumption by the RSX™.

Definition

```
#include <libedgegeomtool.h>
struct EdgeGeomRsxVertexFormat
{
    unsigned m_numAttributes;
    unsigned m_vertexStride;
    EdgeGeomRsxVertexAttributeDefinition m_attributeDefinition[16];
};
```

Members

<i>m_numAttributes</i>	Number of valid entries in the <i>m_attributeDefinition</i> array. Must be <= 16;
<i>m_vertexStride</i>	Number of bytes per vertex.
<i>m_attributeDefinition</i>	Array of EdgeGeomRsxVertexAttributeDefinition structures that describe the order and compression method per vertex attribute.

Description

This structure describes the format of the vertex stream that is intended to be processed by the RSX™. It is a subset of the [EdgeGeomSpuVertexFormat](#) (the latter supports a wider variety of vertex attribute types).

The helper function [edgeGeomRsxVertexFormatIsValid\(\)](#) is provided to test a format for validity.

See Also

[EdgeGeomRsxVertexAttributeDefinition](#)

libedgegeomtool Functions

edgeGeomPartitioner

Splits geometry units into partitions suitable for SPU consumption.

Definition

```
#include <libedgegeomtool_partitioner.h>
void edgeGeomPartitioner (
    const EdgeGeomPartitionerInput &dataIn,
    EdgeGeomPartitionerOutput *dataOut,
)
```

Calling Conditions

Multithread safe, but depends on the `ioBufferCallback`'s calling conditions.

Can be called from a thread.

Arguments

<i>dataIn</i>	Contains all the parameters required by the partitioner.
<i>dataOut</i>	Contains all the data emitted by the partitioner. The caller is responsible for freeing all the fields of the output structure using edgeGeomFree() .

Return Values

There is no error return code for this function. All failures are triggered through `EDGEASSERT` macro, which by default throws an exception of type `(char*)`.

Description

This function splits an input geometry data into one or more output partitions based on the amount of memory available in the SPU during runtime processing, taking into account the kind of processing the geometry will undergo as well as the number of bytes its compressed form will require.

The essence of the algorithm is outlined below:

- (1) Find the best unallocated triangle and allocate it to the current partition.
- (2) Find the best triangle sharing an edge with some triangle in the current partition.
- (3) If the partition is over its memory limit, remove the last triangle, start a new partition, and go to step 1. Otherwise, go to step 2.
- (4) When no triangles remain unallocated, any partially filled partition under construction is kept.

The algorithm it uses to select groups of triangles to collect is a greedy algorithm. The definition of "best" in step 1 is the face with the minimum distance to the existing partition's centroid.

See Also

[EdgeGeomPartitionerInput](#), [EdgeGeomPartitionerOutput](#), [edgeGeomFree](#)

edgeGeomAlloc

Allocates memory using a user-defined callback.

Definition

```
#include <libedgegeomtool.h>
void *edgeGeomAllocEx(
    size_t allocSize,
    const char *filename,
    const uint32_t lineNumber
)
#define edgeGeomAlloc(allocSize) edgeGeomAllocEx(allocSize, __FILE__, __LINE__)
```

Calling Conditions

Multithread safe, but depends on the user callback's calling conditions.

Can be called from a thread.

Arguments

allocSize Size (in bytes) of the requested allocation

Return Values

A pointer to the newly allocated block of memory. This pointer should be passed to [edgeGeomFree\(\)](#) when the memory is no longer needed.

Description

A wrapper around a user-defined memory allocation callback (specified with [edgeGeomSetAllocFunc\(\)](#)). It is used internally throughout libedgegeomtool whenever dynamic memory allocation is needed.

The default memory allocation callback is `malloc()`.

See Also

[edgeGeomFree](#), [edgeGeomSetAllocFunc](#)

edgeGeomFree

Frees memory using a user-defined callback.

Definition

```
#include <libedgegeomtool.h>
void edgeGeomFreeEx(
    void *ptr,
    const char *filename,
    const uint32_t lineNumber
)
#define edgeGeomFree(ptr) edgeGeomFreeEx(ptr, __FILE__, __LINE__)
```

Calling Conditions

Multithread safe, but depends on the user callback's calling conditions.

Can be called from a thread.

Arguments

<i>ptr</i>	Pointer to the block of memory to free. This memory must have been allocated with edgeGeomAlloc() .
------------	---

Return Values

None.

Description

A wrapper around a user-defined memory freeing callback (specified with [edgeGeomSetFreeFunc\(\)](#)). It is used internally throughout libedgegeomtool whenever dynamically allocated memory is freed. In addition, all dynamically-allocated buffers returned by libedgegeomtool functions must be deleted with [edgeGeomFree\(\)](#).

The default memory freeing callback is `free()`.

See Also

[edgeGeomAlloc](#), [edgeGeomSetFreeFunc](#)

edgeGeomSetAllocFunc

Specifies the user-defined memory allocation callback, invoked by the [edgeGeomAlloc\(\)](#) wrapper.

Definition

```
#include <libedgegeomtool.h>
typedef void* (*EdgeGeomAllocFunc)(size_t allocSize,
    const char *filename, const uint32_t lineNumber);
void edgeGeomSetAllocFunc(
    EdgeGeomAllocFunc func
)
```

Calling Conditions

Multithread safe.

Can be called from a thread.

Arguments

<i>func</i>	Function pointer to the desired memory allocation callback. This callback must be compatible with the memory freeing callback wrapped by edgeGeomFree() .
-------------	---

Return Values

None.

Description

Use this function to override the memory allocation function used internally within libedgegeomtool. If a custom allocator is used, make sure to update the memory freeing callback using the [edgeGeomSetFreeFunc\(\)](#).

By default, the memory allocation callback is set to `malloc()`.

See Also

[edgeGeomAlloc](#)

edgeGeomSetFreeFunc

Specifies the user-defined memory freeing callback, invoked by the [edgeGeomFree\(\)](#) wrapper.

Definition

```
#include <libedgegeomtool.h>
typedef void (*EdgeGeomFreeFunc)(void *ptr, const char *filename,
    const uint32_t lineNumber);
void edgeGeomSetFreeFunc(
    EdgeGeomFreeFunc func
)
```

Calling Conditions

Multithread safe.

Can be called from a thread.

Arguments

<i>func</i>	Function pointer to the desired memory freeing callback. This callback must be compatible with the memory allocation callback wrapped by edgeGeomAlloc() .
-------------	--

Return Values

None.

Description

Use this function to override the memory freeing function used internally within libedgegeomtool. If a custom allocator is used, make sure to update the memory allocation callback using [edgeGeomSetAllocFunc\(\)](#).

By default, the memory freeing callback is set to `free()`.

See Also

[edgeGeomFree](#)

edgeGeomGetCommandBufferHoleSize

Determines the maximum space required in the RSX™ command buffer by a given Edge geometry segment's draw commands.

Definition

```
#include <libedgegeomtool.h>
void edgeGeomGetCommandBufferHoleSize(
    uint32_t numOutputAttributes,
    uint32_t numIndexes
)
```

Calling Conditions

Multithread safe.

Can be called from a thread.

Arguments

<i>numOutputAttributes</i>	The number of vertex attributes sent to the RSX™ by this segment
<i>numIndexes</i>	The number of entries in the segment's index buffer

Return Values

The maximum number of bytes of RSX™ command data that will be generated by the Edge runtime in order to render this segment. This value will be rounded up to the nearest multiple of 16 bytes.

Description

The Edge runtime needs to know how large a hole to create in the RSX™ command buffer for the draw commands associated with each segment. The exact size of the commands depends on the number of vertex attributes and the number of unculled triangles that end up being rendered. This function calculates the maximum space that could possibly be required by a given segment (assuming none of its triangles were culled).

The results of this function should be passed as an argument to [edgeGeomMakeSpuConfigInfo\(\)](#).

See Also

[edgeGeomMakeSpuConfigInfo](#)

edgeGeomGetScratchBufferSizeInQwords

Determines the scratch space required on the SPU by a given geometry segment.

Definition

```
#include <libedgegeomtool.h>
uint32_t edgeGeomGetScratchBufferSizeInQwords (
    uint32_t numInputAttributes,
    uint32_t numUniqueVertexes
)
```

Calling Conditions

Multithread safe.

Can be called from a thread.

Arguments

<i>numInputAttributes</i>	The total number of SPU input vertex attributes, in all SPU input streams.
<i>numUniqueVertexes</i>	The number of unique vertexes in this partition/segment. This can be retrieved from the EdgeGeomPartitionerOutput structure.

Return Values

Returns the amount of scratch space required on the SPU (in 16-byte qwords).

Description

The SPU job manager (for example SPURS) needs to know how much extra temporary storage space will be required by each job, in addition to the space needs for the job's input and output buffer. This function calculates the scratch space for a given partition.

The space required is one uniform table per input vertex attribute, plus one extra table if any culling is enabled or if a custom blendshape flavor is used. Each uniform table contains one 16-byte quadword per vertex (rounded up to a multiple of eight vertexes).

See Also

[EdgeGeomPartitionerOutput](#)

edgeGeomGetSpuVertexFormat

Retrieves a pointer to the built-in SPU vertex format associated with a given numerical ID.

Definition

```
#include <libedgegeomtool.h>
EdgeGeomSpuVertexFormat* edgeGeomGetSpuVertexFormat (
    uint32_t formatId,
    EdgeGeomSpuVertexFormat *outFormat = NULL
)
```

Calling Conditions

Multithread safe (note the exception below).

Can be called from a thread.

This function calls `edgeGeomAlloc()`, which can be overridden by the user with an arbitrary function. However, if the user's allocation function is not multithread safe, then this function is not multithread safe.

Arguments

<i>formatId</i>	The numerical ID of the desired built-in SPU vertex format (from the <code>EDGE_GEOM_SPU_VERTEX_FORMAT_*</code> enum)
<i>outFormat</i>	If not <code>NULL</code> , the requested vertex format will be loaded into this pointer's target. If <code>NULL</code> , a new EdgeGeomSpuVertexFormat object will be allocated.

Return Values

If the supplied ID is valid, a pointer to the specified format object will be returned. If `outFormat` was `NULL`, the caller is responsible for deleting this object with [edgeGeomFree\(\)](#).

If the ID is invalid, the return value will be `NULL`, and the object pointed to by `outFormat` (if any) will be unchanged.

Description

Retrieves a pointer to the built-in SPU vertex format associated with a given numerical ID. This function should only be used to retrieve formats for streams that will be processed by the SPU (input vertex and vertex delta streams). SPU output streams and RSX™-only streams have a separate range of format IDs, and should use [edgeGeomGetRsxVertexFormat\(\)](#).

See Also

[edgeGeomGetRsxVertexFormat](#)

edgeGeomGetRsxVertexFormat

Retrieves a pointer to the built-in RSX™ vertex format associated with a given numerical ID.

Definition

```
#include <libedgegeomtool.h>
EdgeGeomRsxVertexFormat* edgeGeomGetRsxVertexFormat (
    uint32_t formatId,
    EdgeGeomRsxVertexFormat *outFormat = NULL
)
```

Calling Conditions

Multithread safe (note the exception below).

Can be called from a thread.

This function calls [edgeGeomAlloc\(\)](#), which can be overridden by the user with an arbitrary function. However, if the user's allocation function is not multithread safe, then this function is not multithread safe.

Arguments

<i>formatId</i>	The numerical ID of the desired built-in RSX™ vertex format (from the <code>EDGE_GEOM_RSX_VERTEX_FORMAT_*</code> enum)
<i>outFormat</i>	If not NULL, the requested vertex format will be loaded into this pointer's target. If NULL, a new EdgeGeomRsxVertexFormat object will be allocated

Return Values

If the supplied ID is valid, a pointer to the specified format object will be returned. If *outFormat* was NULL, the caller is responsible for deleting this object with [edgeGeomFree\(\)](#).

If the ID is invalid, the return value will be NULL, and the object pointed to by *outFormat* (if any) will be unchanged.

Description

Retrieves a pointer to the built-in RSX™ vertex format associated with a given numerical ID. This function should only be used to retrieve formats for streams that will be processed by the RSX™. Streams that will be processed by the SPU (input vertex and vertex delta streams) have a separate range of format IDs, and should use [edgeGeomGetSpuVertexFormat\(\)](#).

See Also

[edgeGeomGetSpuVertexFormatId](#)

edgeGeomGetSpuVertexFormatId

Retrieves the numerical ID of a given SPU vertex format.

Definition

```
#include <libedgegeomtool.h>
uint8_t edgeGeomGetSpuVertexFormatId(
    const EdgeGeomSpuVertexFormat &format
)
```

Calling Conditions

Multithread safe.

Can be called from a thread.

Arguments

<i>format</i>	The SPU vertex format whose ID should be retrieved
---------------	--

Return Values

If the supplied format matches one of the built-in SPU vertex formats, its numerical ID (from the `EDGE_GEOM_SPU_VERTEX_FORMAT_*` enum) will be returned.

If no match is found, the special ID `0xFF` will be returned (indicating a custom format).

Description

Retrieves the numerical ID of a given SPU vertex format. This function should only be used to retrieve formats for streams that will be processed by the SPU (input vertex and vertex delta streams). SPU output streams and RSX™-only streams have a separate range of format IDs, and should use [edgeGeomGetRsxVertexFormatId\(\)](#).

See Also

[edgeGeomGetSpuVertexFormat](#)

edgeGeomGetRsxVertexFormatId

Retrieves the numerical ID of a given RSX™ vertex format.

Definition

```
#include <libedgegeomtool.h>
uint8_t edgeGeomGetRsxVertexFormatId(
    const EdgeGeomRsxVertexFormat &format
)
```

Calling Conditions

Multithread safe.

Can be called from a thread.

Arguments

<i>format</i>	The RSX™ vertex format whose ID should be retrieved
---------------	---

Return Values

If the supplied format matches one of the built-in RSX™ vertex formats, its numerical ID (from the `EDGE_GEOM_RSX_VERTEX_FORMAT_*` enum) will be returned.

If no match is found, the special ID `0xFF` will be returned (indicating a custom format).

Description

Retrieves the numerical ID of a given RSX™ vertex format. This function should only be used to retrieve formats for streams that will be processed by the RSX™. SPU output streams and RSX™-only streams have a separate range of format IDs, and should use [edgeGeomGetSpuVertexFormatId\(\)](#).

See Also

[EdgeGeomRsxVertexFormat](#)

edgeGeomSpuVertexFormatIsValid

Verifies the consistency of an SPU vertex format.

Definition

```
#include <libedgegeomtool.h>
bool edgeGeomSpuVertexFormatIsValid(
    const EdgeGeomSpuVertexFormat &format
)
```

Calling Conditions

Can be called from a thread.

Multithread safe.

Arguments

<i>format</i>	The SPU vertex format to validate
---------------	-----------------------------------

Return Values

Returns true if the specified format passes a series of internal consistency checks, which detect errors in the types, component counts, and byte offsets of the format's attributes.

Returns false if any tests are failed.

Description

This function performs a series of internal consistency checks on an SPU vertex format.

RSX™ vertex formats should use [edgeGeomRsxVertexFormatIsValid\(\)](#).

See Also

[EdgeGeomSpuVertexFormat](#)

edgeGeomRsxVertexFormatIsValid

Verifies the consistency of an RSX™ vertex format.

Definition

```
#include <libedgegeomtool.h>
bool edgeGeomRsxVertexFormatIsValid(
    const EdgeGeomRsxVertexFormat &format
)
```

Calling Conditions

Can be called from a thread.

Multithread safe.

Arguments

<i>format</i>	The RSX™ vertex format to validate
---------------	------------------------------------

Return Values

Returns true if the specified format passes a series of internal consistency checks, which detect errors in the types, component counts and byte offsets of the format's attributes.

Returns false if any tests are failed.

Description

This function performs a series of internal consistency checks on an RSX™ vertex format.

See Also

[EdgeGeomSpuVertexFormat](#)

edgeGeomGetSpuVertexAttributeSize

Returns the size (in bytes) of each instance of a given vertex attribute in a vertex stream.

Definition

```
#include <libedgegeomtool.h>
uint32_t edgeGeomGetSpuVertexAttributeSize (
    const EdgeGeomSpuVertexAttributeDefinition &attr
)
```

Calling Conditions

Can be called from a thread.

Multithread safe.

Arguments

<i>attr</i>	The SPU vertex attribute whose size should be returned
-------------	--

Return Values

Returns the size (in bytes) of the specified attribute.

Description

This function is useful for building custom vertex formats.

See Also

[edgeGeomGetRsxVertexAttributeSize](#)

edgeGeomGetRsxVertexAttributeSize

Returns the size (in bytes) of each instance of a given vertex attribute in a vertex stream.

Definition

```
#include <libedgegeomtool.h>
uint32_t edgeGeomGetRsxVertexAttributeSize (
    const EdgeGeomRsxVertexAttributeDefinition &attr
)
```

Calling Conditions

Can be called from a thread.

Multithread safe.

Arguments

<i>attr</i>	The RSX™ vertex attribute whose size should be returned
-------------	---

Return Values

Returns the size (in bytes) of the specified attribute.

Description

This function is useful for building custom vertex formats.

See Also

[edgeGeomGetSpuVertexAttributeSize](#)

edgeGeomGetAttributeSlotIndex

Retrieves the vertex program slot index associated with a vertex attribute.

Definition

```
#include <libedgegeomtool.h>
uint8_t edgeGeomGetAttributeSlotIndex (
    EdgeGeomAttributeId attrId
)
```

Calling Conditions

Can be called from a thread.

Multithread safe.

Arguments

attrId The ID of the vertex attribute whose vertex program slot index should be retrieved

Return Values

Returns the vertex program slot index for the specified attribute. The attribute indexes returned by this function must match the attribute mapping used by the associated vertex program.

Description

This function is used internally by [edgeGeomMakeSpuConfigInfo\(\)](#). Final correct vertex program slot indexes must be set (using [edgeGeomSetAttributeSlotIndex\(\)](#)) before that function is called.

See Also

[edgeGeomSetAttributeSlotIndex](#), [edgeGeomMakeSpuConfigInfo](#)

edgeGeomSetAttributeSlotIndex

Sets the vertex program slot index associated with a vertex attribute.

Definition

```
#include <libedgegeomtool.h>
void edgeGeomSetAttributeSlotIndex(
    EdgeGeomAttributeId attrId,
    uint8_t slotIndex
)
```

Calling Conditions

Can be called from a thread.

Multithread safe.

Arguments

<i>attrId</i>	The ID of the vertex attribute whose vertex program slot index should be set.
<i>slotIndex</i>	The vertex program slot index for the specified attribute. Valid values are 0 – 15.

Return Values

None.

Description

Sets the vertex program slot index for the specified attribute. The attribute indexes set by this function must match the attribute mapping used by the associated vertex program.

The vertex program slot indexes for all attributes must be set to their final values before [edgeGeomMakeSpuConfigInfo\(\)](#) is called. Note that reasonable defaults are provided for all common vertex attributes (corresponding to the mapping used by sce-cgc).

Note that the vertex program slot indices for Edge's built-in vertex formats are hard-coded in the Edge SPU runtime code. If this function is used to change the slot indices for any of Edge's built-in vertex attributes (such as position, normal, tangent, binormal), the corresponding values must also be updated in the runtime code (see `edgeGeomSetVertexDataArrays()` in `target/src/edge/geom./edgegeom.cpp`).

See Also

[edgeGeomGetAttributeSlotIndex](#)

edgeGeomComputeTriangleCentroids

Generates an array containing the centroid (spatial average) of each triangle in a triangle list.

Definition

```
#include <libedgegeomtool.h>
void edgeGeomComputeTriangleCentroids (
    const float *vertexes,
    uint32_t numFloatsPerVertex,
    uint16_t positionAttributeIndex,
    const uint32_t *triangles,
    uint32_t numTriangles,
    float **outTriangleCentroids
)
```

Calling Conditions

Can be called from a thread.
Multithread safe.

Arguments

<i>vertexes</i>	Pointer to the scene's main vertex stream.
<i>numFloatsPerVertex</i>	Stride (in 32-bit floats) of each vertex in the main vertex stream.
<i>positionAttributeIndex</i>	Index of the position attribute within each vertex's block of floats.
<i>triangles</i>	Pointer to the scene's triangle list. Array must have at least <i>numTriangles*3</i> elements.
<i>numTriangles</i>	Number of triangles in the scene's triangle list.
<i>outTriangleCentroids</i>	A pointer to the output triangle centroids will be written here. The array will contain <i>numTriangles*3</i> elements. The caller is responsible for freeing this memory using edgeGeomFree() .

Return Values

None.

Description

Although they are not strictly necessary, triangle centroids help the Edge partitioner create spatially-coherent partitions. The output array contains the centroid of each triangle in the input list (the centroid is the geometric average of the triangle's three corners).

See Also

[EdgeGeomPartitionerInput](#)

edgeGeomGetBlendedVertexes

Generates an sorted array containing the indexes of all vertexes with non-zero blend shape deltas in at least one blend shape.

Definition

```
#include <libedgegeomtool.h>
void edgeGeomGetBlendedVertexes (
    const float *vertexDeltas,
    uint32_t numVertexes,
    uint32_t numFloatsPerDelta,
    const EdgeGeomSpuVertexFormat & deltaFormat,
    const uint16_t blendedAttributeIndexes,
    const EdgeGeomAttributeId *blendedAttributeIds,
    uint8_t numBlendedAttributes,
    uint32_t numBlendShapes,
    const uint32_t *triangles,
    uint32_t numTriangles,
    uint32_t **outBlendedVertexIndexes,
    uint32_t *outNumBlendedVertexes
)
```

Calling Conditions

Can be called from a thread.

Multithread safe.

Arguments

<i>vertexDeltas</i>	Pointer to the scene's array of vertex deltas. See EdgeGeomScene for a description of the format of this array. The array must have at least $\text{numVertexes} * \text{numFloatsPerDelta} * \text{numBlendShapes}$ elements.
<i>numVertexes</i>	The number of vertexes in the scene.
<i>numFloatsPerDelta</i>	The stride (in 32-bit floats) of each delta in the <i>vertexDeltas</i> array.
<i>deltaFormat</i>	The vertex delta format that will be used to generate the final data. Only attributes present in this format will be tested for non-zero delta data.
<i>blendedAttributeIndexes</i>	Array of indexes of all blended vertex attribute within each vertex delta's block of float data within <i>vertexDeltas</i> . The array must contain at least <i>numBlendedAttributes</i> entries, which must be listed in the same order as the entries of the <i>blendedAttributeIds</i> array.
<i>blendedAttributeIds</i>	Array of attribute IDs for all blended vertex attributes. The array must contain at least <i>numBlendedAttributes</i> entries, which must be listed in the same order as the entries of the <i>blendedAttributeIndexes</i> array.
<i>numBlendedAttributes</i>	The number of attributes in each vertex delta in the <i>vertexDeltas</i> array.
<i>numBlendShapes</i>	The number of blend shape targets in the scene.
<i>triangles</i>	Pointer to the scene's triangle list. The array must have at least $\text{numTriangles} * 3$ elements.
<i>numTriangles</i>	The number of triangles in the scene.

<i>outBlendedVertexIndexes</i>	The array of vertex indexes with non-zero deltas in at least one blend shape will be written here. The array will be sorted in ascending order. The array will have at least *outNumBlendedVertexes elements. The caller is responsible for freeing this memory using edgeGeomFree() .
<i>outNumBlendedVertexes</i>	The number of elements in the * <i>outBlendedVertexIndexes</i> array will be written here.

Return Values

None.

Description

Although not strictly necessary, the partitioner can produce more efficient results if it knows which vertexes can possibly be affected by blend shapes. With this data, the partitioner can avoid placing a few blended vertexes in a partition with many non-blended vertexes; because the entire partition must be blended if any of its vertexes are, this can lead to extremely inefficient segments.

This function creates a sorted list of all vertex indexes in the scene which have non-zero delta data in at least one blend shape. This array is suitable for including in the [EdgeGeomPartitionerInput](#) structure. Only vertex attributes that are present in the provided vertex delta format will be checked for non-zero values.

See Also

[EdgeGeomPartitionerInput](#)

edgeGeomBlendShapeAffectsSegment

Determines whether the specified blend shape has non-zero deltas for any vertexes in a given geometry segment.

Definition

```
#include <libedgegeomtool.h>
bool edgeGeomBlendShapeAffectsSegment(
    const float *shapeDeltas,
    uint32_t numFloatsPerDelta,
    const EdgeGeomSpuVertexFormat& deltaFormat,
    const uint16_t *blendedAttributeIndexes,
    const EdgeGeomAttributeId *blendedAttributeIds,
    uint8_t numBlendedAttributes,
    const uint32_t *originalVertexIndexes,
    uint32_t numUniqueVertexes
)
```

Calling Conditions

Can be called from a thread.
Multithread safe.

Arguments

<i>shapeDeltas</i>	Pointer to the blend shape deltas for the shape to test. The array must have at least <i>numUniqueVertexes * numFloatsPerDelta</i> elements.
<i>numFloatsPerDelta</i>	The stride (in 32-bit floats) of each delta in the <i>shapeDeltas</i> array.
<i>deltaFormat</i>	The vertex delta format that will be used to generate the final data. Only attributes present in this format will be tested for non-zero delta data.
<i>blendedAttributeIndexes</i>	Array of indexes of all blended vertex attribute within each vertex delta's block of float data within <i>shapeDeltas</i> . The array must contain at least <i>numBlendedAttributes</i> entries, which must be listed in the same order as the entries of the <i>blendedAttributeIds</i> array.
<i>blendedAttributeIds</i>	Array of attribute IDs for all blended vertex attributes. The array must contain at least <i>numBlendedAttributes</i> entries, which must be listed in the same order as the entries of the <i>blendedAttributeIndexes</i> array.
<i>numBlendedAttributes</i>	The number of attributes in each vertex delta in the <i>shapeDeltas</i> array.
<i>originalVertexIndexes</i>	A mapping table from the partition-local vertex order to the vertex indexes in the scene's original vertex array. This data can be found in the EdgeGeomPartitionerOutput structure. The array must have at least <i>numUniqueVertexes</i> valid entries.
<i>numUniqueVertexes</i>	The number of unique vertexes in the partition.

Return Values

Returns true if at least one attribute of at least one vertex in the *originalVertexIndexes* array contains non-zero delta data in the *shapeDeltas* array. Only attributes present in the provided *deltaFormat* will be tested.

Returns false if no non-zero delta data is found in any of the vertexes tested.

Description

This function should be called after the scene has been partitioned, to determine which of the scene's blend shapes are relevant to each geometry segment.

edgeGeomMakeSpuConfigInfo

Builds a geometry segment's EdgeGeomSpuConfigInfo object in a memory buffer, suitable for serialization or immediate processing by the Edge Geometry runtime.

Definition

```
#include <libedgegeomtool.h>
void edgeGeomMakeSpuConfigInfo(
    uint32_t numUniqueVertexes,
    uint32_t numTriangles,
    uint32_t numInputAttributes,
    uint8_t segmentFlags,
    EdgeGeomIndexesFlavor indexListType,
    EdgeGeomSkinningFlavor skinType,
    EdgeGeomMatrixFormat skinMatrixFormat,
    uint8_t inputVertexFormatId,
    uint8_t secondaryInputVertexFormatId,
    uint8_t outputVertexFormatId,
    uint8_t vertexDeltaFormatId,
    uint32_t commandBufferHoleSize,
    uint8_t **outSpuConfigInfo
)
```

Calling Conditions

Can be called from a thread.

Multithread safe.

Arguments

<i>numUniqueVertexes</i>	The number of unique vertexes in this geometry segment.
<i>numTriangles</i>	The number of triangles in this segment's triangle list.
<i>numInputAttributes</i>	The number of vertex attributes being sent to the SPU. This includes attributes in secondary vertex stream, if one exists.
<i>segmentFlags</i>	Reserved (must be zero).
<i>indexListType</i>	The index list format used by this geometry segment. <i>Must</i> match the value used to build this segment.
<i>skinType</i>	The skinning algorithm to be used by this segment. <i>Must</i> match the value used to build this segment.
<i>skinMatrixFormat</i>	The format (and size) of skinning matrices, if skinning is enabled.
<i>inputVertexFormatId</i>	The numeric ID of this segment's primary input vertex stream format (from the list of built-in SPU vertex formats). An ID of 255 (0xFF) indicates a custom input format.
<i>secondaryInputVertexFormatId</i>	The numeric ID of this segment's secondary input vertex stream format (from the list of built-in RSX™ vertex formats). An ID of 255 (0xFF) indicates a custom input format. If only one input stream is present, this field will be ignored (and can take any value).
<i>outputVertexFormatId</i>	The numerical ID of this segment's output vertex stream format (from the list of built-in RSX™ vertex formats). An ID of 255 (0xFF) indicates a custom output format.

<i>vertexDeltaFormatId</i>	The numerical ID of this segment's blend shape vertex delta stream format (from the list of built-in SPU vertex formats). An ID of 255 (0xFF) indicates a custom format. If no blend shapes are present, this field will be ignored (and can take any value).
<i>commandBufferHoleSize</i>	Size (in bytes) of the command buffer.
<i>outSpuConfigInfo</i>	A pointer to the final <code>EdgeGeomSpuConfigInfo</code> object will be written here. The buffer is already byte-swapped to big-endian format. The caller is responsible for releasing this memory using edgeGeomFree() .

Return Values

None.

Description

To determine the numerical IDs of the segment's input/output/delta formats, use the [edgeGeomGetSpuVertexFormatId\(\)](#) and [edgeGeomGetRsxVertexFormatId\(\)](#) functions.

To determine the command buffer hole size, use the [edgeGeomGetCommandBufferHoleSize\(\)](#) function.

See Also

[edgeGeomGetSpuVertexFormatId](#), [edgeGeomGetCommandBufferHoleSize](#)

edgeGeomMakeIndexBuffer

Builds a geometry segment's final index buffer, suitable for serialization or immediate processing by the Edge Geometry runtime.

Definition

```
#include <libedgegeomtool.h>
void edgeGeomMakeIndexBuffer(
    const uint32_t *triangles,
    uint32_t numTriangles,
    EdgeGeomIndexesFlavor indexListType,
    uint8_t **outIndexBuffer,
    uint16_t outIndexDmaSizes[2]
)
```

Calling Conditions

Can be called from a thread.
Multithread safe.

Arguments

<i>triangles</i>	The triangle list for this geometry segment. Must contain at least <i>numTriangles*3</i> entries.
<i>numTriangles</i>	The number of triangles in this segment's triangle list.
<i>indexListType</i>	The index list format used by this geometry segment. Must match the value used to build this segment.
<i>outIndexBuffer</i>	A pointer to the final index buffer will be written here. The buffer will already be byte-swapped to a big-endian format. The caller is responsible for releasing this memory with edgeGeomFree() .
<i>outIndexDmaSizes</i>	The size of the index buffer DMA tags will be stored here.

Return Values

None.

Description

Use this function to produced the final, compressed index buffer from the per-segment triangle lists generated by Edge partitioner.

See Also

[EdgeGeomPartitionerOutput](#)

edgeGeomMakeSpuVertexBuffer

Builds a vertex buffer suitable for processing by the Edge Geometry SPU runtime.

Definition

```
#include <libedgegeomtool.h>
void edgeGeomMakeSpuVertexBuffer(
    const float *sourceVertexes,
    uint32_t numFloatsPerSourceVertex,
    const uint16_t *sourceAttributeIndexes,
    const EdgeGeomAttributeId *sourceAttributeIds,
    uint8_t numSourceAttributes,
    const uint32_t *originalVertexIndexes,
    uint32_t numUniqueVertexes,
    const EdgeGeomSpuVertexFormat &vertexFormat,
    uint8_t **outVertexBuffer,
    uint16_t outVertexDmaSizes[3],
    uint32_t **outFixedPointOffsets,
    uint32_t *outFixedPointOffsetsSize
)
```

Calling Conditions

Can be called from a thread.

Multithread safe.

Arguments

<i>sourceVertexes</i>	Pointer to the scene's original vertex stream.
<i>numFloatsPerSourceVertex</i>	The stride (in 32-bit floats) of each delta in the <i>sourceVertexes</i> array.
<i>sourceAttributeIndexes</i>	Array of indexes of all vertex attribute within each vertex's block of float data within <i>sourceVertexes</i> . The array must contain at least <i>numSourceAttributes</i> entries, which must be listed in the same order as the entries of the <i>sourceAttributeIds</i> array.
<i>sourceAttributeIds</i>	Array of attribute IDs for all vertex attributes. The array must contain at least <i>numSourceAttributes</i> entries, which must be listed in the same order as the entries of the <i>sourceAttributeIndexes</i> array.
<i>numSourceAttributes</i>	The number of attributes in each vertex in the <i>sourceVertexes</i> array.
<i>originalVertexIndexes</i>	A mapping table from the partition-local vertex order to the vertex indexes in the scene's original vertex array. This data can be found in the EdgeGeomPartitionerOutput structure. The array must have at least <i>numUniqueVertexes</i> valid entries.
<i>numUniqueVertexes</i>	The number of unique vertexes in the partition.
<i>vertexFormat</i>	The vertex format that should be used to generate the vertex buffer for this segment.
<i>outVertexBuffer</i>	A pointer to the final vertex buffer will be written here. The buffer will already be byte-swapped to a big-endian format. The caller is responsible for releasing this memory using edgeGeomFree() .
<i>outVertexDmaSizes</i>	The size of the vertex buffer DMA tags will be stored here.

<i>outFixedPointOffsets</i>	The table of fixed-point offsets will be written here. The table will have four entries per fixed-point attribute in <i>inputFlavor</i> . The caller is responsible for freeing this memory using edgeGeomFree() .
<i>outFixedPointOffsetsSize</i>	The number of elements in the <i>outFixedPointOffsets</i> array will be written here. There will be four entries per fixed-point vertex attribute in <i>inputFlavor</i> .

Return Values

None.

Description

Use this function to produce a final, compressed SPU input vertex buffer for a geometry segment, after running the Edge partitioner to determine which unique vertexes are present in each segment.

This function can be used for both primary and secondary SPU input vertex streams; the only difference would be the value of *vertexFormat*.

See Also

[EdgeGeomPartitionerOutput](#)

edgeGeomMakeRsxVertexBuffer

Builds a vertex buffer suitable for rendering by the RSX™.

Definition

```
#include <libedgegeomtool.h>
void edgeGeomMakeRsxVertexBuffer (
    const float *sourceVertexes,
    uint32_t numFloatsPerSourceVertex,
    const uint16_t *sourceAttributeIndexes,
    const EdgeGeomAttributeId *sourceAttributeIds,
    uint8_t numSourceAttributes,
    const uint32_t *originalVertexIndexes,
    uint32_t numUniqueVertexes,
    const EdgeGeomRsxVertexFormat &vertexFormat,
    bool writeBigEndian,
    uint8_t **outVertexBuffer,
    uint16_t outVertexDmaSizes[3]
)
```

Calling Conditions

Can be called from a thread.
Multithread safe.

Arguments

<i>sourceVertexes</i>	Pointer to the scene's original vertex stream.
<i>numFloatsPerSourceVertex</i>	The stride (in 32-bit floats) of each delta in the <i>sourceVertexes</i> array.
<i>sourceAttributeIndexes</i>	Array of indexes of all vertex attribute within each vertex's block of float data within <i>sourceVertexes</i> . The array must contain at least <i>numSourceAttributes</i> entries, which must be listed in the same order as the entries of the <i>sourceAttributeIds</i> array.
<i>sourceAttributeIds</i>	Array of attribute IDs for all vertex attributes. The array must contain at least <i>numSourceAttributes</i> entries, which must be listed in the same order as the entries of the <i>sourceAttributeIndexes</i> array.
<i>numSourceAttributes</i>	The number of attributes in each vertex in the <i>sourceVertexes</i> array.
<i>originalVertexIndexes</i>	A mapping table from the partition-local vertex order to the vertex indexes in the scene's original vertex array. This data can be found in the EdgeGeomPartitionerOutput structure. The array must have at least <i>numUniqueVertexes</i> valid entries.
<i>numUniqueVertexes</i>	The number of unique vertexes in the partition.
<i>vertexFormat</i>	The vertex format that should be used to generate the vertex buffer for this segment.
<i>writeBigEndian</i>	If true, the output buffer will be written in big-endian (PlayStation®3-native) format. If false, the stream will be written in little-endian (x86-native) format.
<i>outVertexBuffer</i>	A pointer to the final vertex buffer will be written here. The buffer will already be byte-swapped to a big-endian format. The caller is responsible for releasing this memory using edgeGeomFree() .
<i>outVertexDmaSizes</i>	The size of the vertex buffer DMA tags will be stored here.

Return Values

None.

Description

Use this function to produce a final, compressed RSX™ input vertex buffer for a geometry segment, after running the Edge partitioner to determine which unique vertexes are present in each segment. These attributes would not be passed through Edge at run time, but are sorted in the same order as the segment's SPU input streams.

Alternately, this function could be used to generate regular RSX™-only streams completely independent of Edge. In this case, `numUniqueVertexes` should equal the total number of input vertexes, and `originalVertexIndexes` should be an identity mapping `[0, 1, 2, ..., numUniqueVertexes-1]`.

See Also

[EdgeGeomPartitionerOutput](#)

edgeGeomMakeSkinningBuffer

Builds a geometry segment's final skinning buffers, suitable for serialization or immediate processing by the Edge Geometry runtime.

Definition

```
#include <libedgegeomtool.h>
void edgeGeomMakeSkinningBuffer(
    const int32_t *matrixIndexes,
    const float *skinningWeights,
    EdgeGeomSkinningFlavor skinType,
    EdgeGeomMatrixFormat skinMatrixFormat,
    const uint32_t *originalVertexIndexes,
    uint32_t numUniqueVertexes,
    uint8_t **outSkinIndexesAndWeights,
    uint16_t outSkinIndexesAndWeightsDmaSizes[2],
    uint16_t outSkinMatricesByteOffsets[2],
    uint16_t outSkinMatricesSizes[2]
)
```

Calling Conditions

Can be called from a thread.
Multithread safe.

Arguments

<i>matrixIndexes</i>	Pointer to the scene's array of bone matrix indexes. The array contains four elements per vertex, with unused influences set to -1.
<i>skinningWeights</i>	Pointer to the scene's array of skinning weights. The array contains four elements per vertex in the range from 0.0 to 1.0, with each vertex's four weights summing to 1.0.
<i>skinType</i>	The skinning algorithm that will be used on this segment. It must match the one used passed to the partitioner earlier!
<i>skinMatrixFormat</i>	The format (and size) of skinning matrices.
<i>originalVertexIndexes</i>	A mapping table from the partition-local vertex order to the vertex indexes in the scene's original vertex array. This data can be found in the EdgeGeomPartitionerOutput structure. The array must have at least <i>numUniqueVertexes</i> valid entries.
<i>numUniqueVertexes</i>	The number of unique vertexes in the partition.
<i>outSkinIndexesAndWeights</i>	A pointer to the final skinning indexes/weights buffer will be written here. The buffer will already be byte-swapped to a big-endian format. The caller is responsible for releasing this memory using edgeGeomFree() .
<i>outSkinIndexesAndWeightsDmaSizes</i>	The size of the skinning weights/indexes buffer DMA tags will be stored here.
<i>outSkinMatricesByteOffsets</i>	The byte offsets to the beginning of the two skinning matrix ranges used by this segment will be stored here.
<i>outSkinMatricesSizes</i>	The sizes (in bytes) of the two matrix ranges used by this segment will be stored here.

Return Values

None.

Description

Use this function to produced the final, compressed skinning buffer for a geometry segment, after running the Edge partitioner to determine which unique vertexes are present in each segment.

Rather than sending all the scene's skinning matrices to the SPUs along with every geometry segment, the Edge tools find two subranges of the full bone array that include all the matrices used by each segment. The offsets and sizes of these ranges are returned by these function. At run time, the PPU transfers these two ranges to the SPU one after the other (in the order provided here), forming one contiguous matrix buffer. The matrix indexes generated by this function (in the skinning weights/indexes array) are relative to this final contiguous table.

See Also

[EdgeGeomPartitionerOutput](#)

edgeGeomMakeBlendShapeBuffer

Builds a geometry segment's final blend shape buffer for a given blend shape, suitable for serialization or immediate processing by the Edge Geometry runtime.

Definition

```
#include <libedgegeomtool.h>
void edgeGeomMakeBlendShapeBuffer(
    const float *shapeDeltas,
    uint32_t numFloatsPerDelta,
    const EdgeGeomSpuVertexFormat& deltaFormat,
    const uint16_t *blendedAttributeIndexes,
    const EdgeGeomAttributeId *blendedAttributeIds,
    uint8_t numBlendedAttributes,
    const uint32_t *originalVertexIndexes,
    uint32_t numUniqueVertexes,
    uint8_t **outShapeBuffer,
    uint16_t *outShapeBufferSize
)
```

Calling Conditions

Can be called from a thread.
Multithread safe.

Arguments

<i>shapeDeltas</i>	Pointer to the blend shape deltas for the shape to test. The array must have at least <i>numUniqueVertexes</i> * <i>numFloatsPerDelta</i> elements.
<i>numFloatsPerDelta</i>	The stride (in 32-bit floats) of each delta in the <i>shapeDeltas</i> array.
<i>deltaFormat</i>	The blend shape vertex delta format that will be used to generate the final data. Only attributes present in this format will be tested for non-zero delta data.
<i>blendedAttributeIndexes</i>	Array of indexes of all blended vertex attribute within each vertex delta's block of float data within <i>shapeDeltas</i> . The array must contain at least <i>numBlendedAttributes</i> entries, which must be listed in the same order as the entries of the <i>blendedAttributeIds</i> array.
<i>blendedAttributeIds</i>	Array of attribute IDs for all blended vertex attributes. The array must contain at least <i>numBlendedAttributes</i> entries, which must be listed in the same order as the entries of the <i>blendedAttributeIndexes</i> array.
<i>numBlendedAttributes</i>	The number of attributes in each vertex delta in the <i>shapeDeltas</i> array.
<i>originalVertexIndexes</i>	A mapping table from the partition-local vertex order to the vertex indexes in the scene's original vertex array. This data can be found in the EdgeGeomPartitionerOutput structure. The array must have at least <i>numUniqueVertexes</i> valid entries.
<i>numUniqueVertexes</i>	The number of unique vertexes in the partition.
<i>outShapeBuffer</i>	A pointer to the final blend shape buffer buffer will be written here. The buffer will already be byte-swapped to a big-endian format. The caller is responsible for releasing this memory using edgeGeomFree() .
<i>outShapeBufferSize</i>	The size of the blend shape buffer will be stored here.

Return Values

None.

Description

Use this function to produce the final, compressed blend shape buffer for a given blend shape applied to a given geometry segment, after running the Edge partitioner to determine which unique vertexes are present in each segment.

Rather than creating a blend shape buffer for every blend shape in the scene applied to every segment, it is recommended that the [edgeGeomBlendShapeAffectsSegment\(\)](#) function be used to include only the blend shapes that will actually affect each segment.

See Also

[EdgeGeomPartitionerOutput](#), [edgeGeomBlendShapeAffectsSegment](#)

edgeGeomMergeIdenticalVertexes

Identifies and removes all duplicate vertexes from an [EdgeGeomScene](#) object. Only the triangle list is modified; the actual vertex data is kept in its original state.

Definition

```
#include <libedgegeomtool_wrap.h>
uint32_t edgeGeomMergeIdenticalVertexes (
    EdgeGeomScene &edgeScene
)
```

Calling Conditions

Can be called from a thread.

Multithread safe.

Arguments

edgeScene

The input scene to process. The scene's triangle list will be modified such that all references to equivalent vertexes will refer to the same vertex.

Return Values

Returns the number of vertexes that were removed due to duplication.

Description

This function uses a closed hash table to determine duplicated vertexes. The hash key is simply the sum of all the floating-point attributes associated with the vertex. This is not ideal, because it does not consider any specific input flavors, or detect that – after compression/conversion – two slightly different vertexes are actually identical. Instead, this function simply defends against poorly written exporters or asset pipeline conversion routines that duplicate data unnecessarily.

This function should be called before partitioning.

edgeGeomPartitionSceneIntoSegments

Performs all end-to-end processing necessary to convert an input scene into a list of geometry segments. This includes partitioning into batches, partitioning batches into segments, and generating all necessary data buffers for each segment.

Definition

```
#include <libedgegeomtool_wrap.h>
void edgeGeomPartitionSceneIntoSegments (
    const EdgeGeomScene &edgeScene,
    const EdgeGeomSegmentFormat &edgeFormat,
    EdgeGeomSegment **outSegments,
    uint32_t *outNumSegments,
    EdgeGeomCustomPartitionDataSizeFunc partitionDataSizeFunc = 0,
    EdgeGeomCustomCommandBufferHoleSizeFunc commandBufferHoleSizeFunc = 0
)
```

Calling Conditions

Can be called from a thread.
Multithread safe.

Arguments

<i>edgeScene</i>	The input scene to process.
<i>edgeFormat</i>	Contains formatting information for the output segments.
<i>outSegments</i>	The array of output geometry segments will be written here. The array will have <i>*outNumSegments</i> elements. The caller is responsible for freeing this array (and all array members within each segment) with edgeGeomFree() .
<i>outNumSegments</i>	The number of elements in the <i>outSegments</i> array will be written here.
<i>partitionDataSizeFunc</i>	Optional callback to register any additional data to be included in the partitions (apart from the default Edge data). Can be NULL if unnecessary.
<i>commandBufferHoleSizeFunc</i>	Optional callback to register any additional GCM commands that will be called from the Edge job (apart from the ones expected by Edge). Can be NULL if unnecessary.

Return Values

None.

Description

This function is a wrapper around various Edge API functions which provides a simple, all-in-one interface to Edge Geometry processing. For many applications, this could be the only Edge Geometry function your tools need to call.

The function first partitions the scene into batches (portions of the scene which use a single material ID and skinning settings). Each batch is then passed into [edgeGeomPartitioner\(\)](#) to be partitioned into geometry segments. Finally, the data buffers for each geometry segment are generated using the `edgeGeomMake*` functions.

The caller must to release the memory of each [EdgeGeomSegment](#)'s data buffers before releasing the *outSegments* array itself, using [edgeGeomFreeSegmentData\(\)](#).

If your Edge jobs will perform any runtime custom processing that requires additional data or writing additional GCM commands to the command buffer hole, use the two optional callback functions (*partitionDataSizeFunc* and *commandBufferHoleSizeFunc*) to inform the partitioner of these additional requirements. It is unnecessary to use these functions for standard Edge data tables or for GCM commands to the command buffer hole that are required to draw your final triangles or that are called internally by runtime Edge functions.

See Also

[edgeGeomMergeIdenticalVertexes](#), [edgeGeomFreeSegmentData](#)

edgeGeomFreeSegmentData

Releases the memory for all data associated with an [EdgeGeomSegment](#) object (but not the object itself).

Definition

```
#include <libedgegeomtool_wrap.h>
void edgeGeomFreeSegmentData (
    EdgeGeomSegment &segment
)
```

Calling Conditions

Should not be called from a thread.

Not multithread safe.

Arguments

<i>segment</i>	The segment whose data should be deleted
----------------	--

Return Values

None.

Description

This function simply calls [edgeGeomFree\(\)](#) on each array within an [EdgeGeomSegment](#) structure. It is highly recommended that users call this function instead of freeing a segment's buffers themselves, to protect their code against future changes to the [EdgeGeomSegment](#) structure.

See Also

[edgeGeomFree](#)

edgeGeomTriangulatePolygons

Converts an arbitrary polygon list into triangles.

Definition

```
#include <libedgegeomtool_wrap.h>
void edgeGeomTriangulatePolygons(
    const int32_t *polygons,
    const int32_t *materialIdPerPolygon,
    uint32_t **outTriangles,
    int32_t **outMaterialIdPerTriangle,
    uint32_t *outNumTriangles
)
```

Calling Conditions

Can be called from a thread.
Multithread safe.

Arguments

<i>polygons</i>	Array of vertex indexes, where each polygon is terminated by a -1, and the list is terminated by another -1.
<i>materialIdPerPolygon</i>	Array of material IDs, where one ID is supplied per polygon. The ID can be any unique identifier for the material (a hash of its filename, a pointer, and so forth).
<i>outTriangles</i>	A pointer to the output triangle list will be written here. The array will have <i>*outNumTriangles</i> * 3 elements. It is the caller's responsibility to free this memory using edgeGeomFree() .
<i>outMaterialIdPerTriangle</i>	A pointer to the output per-triangle material ID. The array will have <i>*outNumTriangles</i> elements. It is the caller's responsibility to free this memory using edgeGeomFree() .
<i>outNumTriangles</i>	The number of triangles in the output array will be written here.

Return Values

None.

Description

This function triangulates polygons by fanning them from the first vertex in the polygon. It is widely recognized that fanning is not the ideal method for handling complex polygons, but this function is supplied as a convenience. It will suffice for geometry in which polygons tend to be convex and mostly regular.

A properly formatted input data stream should look like this:

polygons:

0	1	2	3	-1	1	0	4	5	-1	-1
---	---	---	---	----	---	---	---	---	----	----

materialIdPerPolygon:

0	0
---	---

edgeGeomMakeSpuStreamDescription

Generates a stream description structure for an [EdgeGeomSpuVertexFormat](#).

Definition

```
#include <libedgegeomtool.h>
void edgeGeomMakeSpuStreamDescription (
    const EdgeGeomSpuVertexFormat &vertexFormat,
    uint8_t **outStreamDescription,
    uint16_t *outStreamDescriptionSize
)
```

Calling Conditions

Can be called from a thread.
Multithread safe.

Arguments

<i>vertexFormat</i>	SPU vertex format to build the stream description from.
<i>outStreamDescription</i>	A pointer to the new stream description will be written here. It is the caller's responsibility to free this memory with edgeGeomFree() .
<i>outStreamDescriptionSize</i>	The size (in bytes) of the new stream description structure will be written here.

Return Values

None.

Description

This function generates a stream description buffer for the specified SPU vertex format.

Note

The buffers generated by this function are used by the Edge Geometry runtime to decompress an input vertex stream using a generic, data-driven algorithm. This approach provides increased flexibility for users with a large number of vertex formats. For users with only a few vertex stream formats, using pre-compiled flavors is still the recommended approach; the code generated by the Edge vertex compression macros is much more efficient than the data-driven compression functions that work with stream descriptions.

edgeGeomMakeRsxStreamDescription

Generates a stream description structure for an [EdgeGeomRsxVertexFormat](#).

Definition

```
#include <libedgegeomtool.h>
void edgeGeomMakeRsxStreamDescription (
    const EdgeGeomRsxVertexFormat &vertexFormat,
    uint8_t **outStreamDescription,
    uint16_t *outStreamDescriptionSize
)
```

Calling Conditions

Can be called from a thread.
Multithread safe.

Arguments

<i>vertexFormat</i>	RSX™ vertex format to build the stream description from.
<i>outStreamDescription</i>	A pointer to the new stream description will be written here. It is the caller's responsibility to free this memory with edgeGeomFree() .
<i>outStreamDescriptionSize</i>	The size (in bytes) of the new stream description structure will be written here.

Return Values

None.

Description

This function generates a stream description buffer for the specified RSX™ vertex format.

Note

The buffers generated by this function are used by the Edge Geometry runtime to compress an output vertex stream using a generic, data-driven algorithm. This approach provides increased flexibility for users with a large number of vertex formats. For users with only a few vertex stream formats, using pre-compiled flavors is still the recommended approach; the code generated by the Edge vertex compression macros is much more efficient than the data-driven compression functions that work with stream descriptions.

edgeGeomKCacheOptimizer

Reorders triangles and vertex order within triangles to minimize cache misses on GPU.

Definition

```
#include <libedgegeomtool.h>
void edgeGeomKCacheOptimizer(
    const uint32_t *inTriangles,
    uint32_t numTriangles,
    void *userData,
    uint32_t *outTriangles
)
```

Calling Conditions

Can be called from a thread.

Multithread safe.

Arguments

<i>inTriangles</i>	Array of vertex index triples representing triangles. The array must have <i>numTriangles</i> *3 entries.
<i>numTriangles</i>	Count of triangles in <i>inTriangles</i> and <i>outTriangles</i> .
<i>userData</i>	Pointer to EdgeGeomKCacheOptimizerUserData structure, containing tuning factors for the kcache algorithm.
<i>outTriangles</i>	Pre-allocated array where the optimized triangle list will be written. Must have the same number of elements as <i>inTriangles</i> .

Return Values

None.

Description

This function is a modified version of the algorithm presented in *An Improved Vertex Caching Scheme for 3D Mesh Rendering* by Gang Lin, Thomas P.-Y. Yu of the ECSE & Math Department at Rensselaer Polytechnic Institute (for a URL, see the “Related Documentation and Other Resources” section of the *PlayStation®Edge Library Overview*). The modifications are specifically reordering vertexes in each triangle to match the mini-cache better, and by not bothering with the counter-clockwise iteration around a vertex neighborhood – rather, the vertex neighborhood is searched for the least expensive triangle each time. Both modifications yield significant improvement on test models.

The ratio between *m_fifoMiss* and *m_lruMissCost* has a significant effect on how the algorithm attempts to optimize for triangle order. When *m_lruMissCost* is significantly greater than *m_fifoMissCost*, the triangle list will be highly optimized for the short-term gain of selecting triangles that have high coherency in the mini-cache, but may exhibit poor post-transform cache behavior. Similarly, if the *m_fifoMissCost* is relatively large, the vertexes may incur a stiff penalty by constantly needing to be re-copied to the mini-cache, even though the vertex program need not be run again. The *m_fifoMissCost:m_lruMissCost* ratio that has given the most consistent results is 4:1.

Note that the constants *m_k1*, *m_k2*, and *m_k3* are not exactly as described in the paper. This is because the overall score for a potential focus vertex is not strictly the number of post-transform cache misses. Thus, the resultant 3-D graph of constant values presented in the paper is not the same as that which developers using Edge are likely to encounter. With this algorithm, the surface is less smooth,

and often there is no clear gradient to follow to find best results for all models. Varying these constants is the only way to achieve the optimal triangle list.

See Also

[EdgeGeomKCacheOptimizerUserData](#)

edgeGeomKCacheOptimizerHillclimber

Front end for [edgeGeomKCacheOptimizer](#) that automatically varies constants k1,k2,k3.

Definition

```
#include <libedgegeomtool.h>
void edgeGeomKCacheOptimizerHillclimber(
    const uint32_t *inTriangles,
    uint32_t numTriangles,
    void *userData,
    uint32_t *outTriangles
)
```

Calling Conditions

Can be called from a thread.

Multithread safe.

Arguments

<i>inTriangles</i>	Array of vertex index triples representing triangles. The array must have <i>numTriangles</i> *3 entries.
<i>numTriangles</i>	Count of triangles in <i>inTriangles</i> and <i>outTriangles</i> .
<i>userData</i>	Pointer to EdgeGeomKCacheOptimizerHillclimberUserData structure, containing tuning factors for the kcache algorithm.
<i>outTriangles</i>	Pre-allocated array where the optimized triangle list will be written. Must have the same number of elements as <i>inTriangles</i> .

Return Values

None.

Description

This function makes a specified number of probes into the constant space that guides the quality of the vertex cache optimizer. Because there is no clear gradient to the constant surface that leads to better results, the hill climber function takes progressively smaller steps away from the current constant set, in an effort to refine the search for better results.

See Also

[EdgeGeomKCacheOptimizerHillclimberUserData](#)

Callback Functions

EdgeGeomVertexCacheOptimizerFunc

Callback function that is called by the library to optimize partitioned triangle lists for RSX™ cache.

Definition

```
#include <libedgegeomtool.h>
typedef void (*EdgeGeomVertexCacheOptimizerFunc) (
    const uint32_t *triangles,
    uint32_t numTriangles,
    void *userData,
    uint32_t *outTriangles
);
```

Arguments

<i>triangles</i>	Input array of vertex indexes where three consecutive indexes signify a triangle.
<i>numTriangles</i>	Count of triangles.
<i>userData</i>	Pointer to user specified data.
<i>outTriangles</i>	Pointer to a buffer for outputting optimized triangle list. The size of this buffer is $3 * \text{numTriangles} * \text{sizeof}(\text{uint32_t})$.

Return Values

None.

Description

This callback function is called when the library needs to optimize a triangle list for a certain partition. Users can register their own function and specify a data pointer through *m_cacheOptimizerFunc* and *m_cacheOptimizerUserData* in the [EdgeGeomPartitionerInput](#) structure. Edge provided two optimization functions, [edgeGeomKCacheOptimizer\(\)](#) and [edgeGeomKCacheOptimizerHillclimber\(\)](#).

See Also

[EdgeGeomPartitionerInput](#), [edgeGeomKCacheOptimizer](#), [edgeGeomKCacheOptimizerHillclimber](#)

EdgeGeomCustomPartitionDataSizeFunc

Callback function that returns the size of the extra custom data for the input partition.

Definition

```
#include <libedgegeomtool.h>
typedef uint32_t (*EdgeGeomCustomPartitionDataSizeFunc) (
    const EdgeGeomPartitionElementCounts &partitionContents
);
```

Arguments

partitionContents Description of the current partition

Return Values

Size of the data that the user application needs for input partition.

Description

The Edge partitioner calls this function when choosing which triangle to add to a partition.

partitionContents is a description of the partition currently under construction.

This function must return the size (in bytes) of any extra data (beyond the standard Edge buffers) that would be required by the partition.

Notes

This function might be called frequently. It should therefore perform as simple a calculation as possible, or else it will quickly dominate the tool's running time.

See Also

[EdgeGeomPartitionerInput](#), [EdgeGeomPartitionElementCounts](#)

Enumerations

EdgeGeomIndexesFlavor

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

Definition

```
enum EdgeGeomIndexesFlavor
{
    kIndexesU16TriangleListCW =
        EDGE_GEOM_INDEXES_U16_TRIANGLE_LIST_CW,
    kIndexesU16TriangleListCCW =
        EDGE_GEOM_INDEXES_U16_TRIANGLE_LIST_CCW,
    kIndexesCompressedTriangleListCW =
        EDGE_GEOM_INDEXES_COMPRESSED_TRIANGLE_LIST_CW,
    kIndexesCompressedTriangleListCCW =
        EDGE_GEOM_INDEXES_COMPRESSED_TRIANGLE_LIST_CCW,
};
```

Description

enum Value	Description
kIndexesU16TriangleListCW	16-bit index data with clockwise triangle ordering
kIndexesU16TriangleListCCW	16-bit index data with counter-clockwise triangle ordering
kIndexesCompressedTriangleListCW	Software Compressed index data with clockwise triangle ordering
kIndexesCompressedTriangleListCCW	Software Compressed index data with counter-clockwise triangle ordering

See Also

[EdgeGeomSegmentFormat](#), [EdgeGeomKCacheOptimizerHillclimberUserData](#),
[EdgeGeomPartitionerInput](#), [edgeGeomMakeSpuConfigInfo](#), [edgeGeomMakeIndexBuffer](#)

EdgeGeomSkinningFlavor

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

Definition

```
enum EdgeGeomSkinningFlavor
{
    kSkinNone = EDGE_GEOM_SKIN_NONE,
    kSkinNoScaling = EDGE_GEOM_SKIN_NO_SCALING,
    kSkinUniformScaling = EDGE_GEOM_SKIN_UNIFORM_SCALING,
    kSkinNonUniformScaling = EDGE_GEOM_SKIN_NON_UNIFORM_SCALING,
    kSkinSingleBoneNoScaling = EDGE_GEOM_SKIN_SINGLE_BONE_NO_SCALING,
    kSkinSingleBoneUniformScaling =
        EDGE_GEOM_SKIN_SINGLE_BONE_UNIFORM_SCALING,
    kSkinSingleBoneNonUniformScaling =
        EDGE_GEOM_SKIN_SINGLE_BONE_NON_UNIFORM_SCALING,
};
```

Description

enum Value	Description
kSkinNone	Do not perform skinning.
kSkinNoScaling	Unit matrix skinning.
kSkinUniformScaling	Perform skinning.
kSkinNonUniformScaling	Perform skinning and compute cofactor matrices for normal transformation.
kSkinSingleBoneNoScaling	Single bone unit matrix skinning.
kSkinSingleBoneUniformScaling	Single bone skinning.
kSkinSingleBoneNonUniformScaling	Single bone skinning. Computes cofactor matrices for normal transformation.

See Also

[EdgeGeomSegmentFormat](#), [EdgeGeomPartitionerInput](#), [edgeGeomMakeSpuConfigInfo](#), [edgeGeomMakeSkinningBuffer](#)

EdgeGeomMatrixFormat

Constants expressing the supported skinning matrix formats.

Definition

```
enum EdgeGeomMatrixFormat
{
    kMatrix3x4RowMajor = EDGE_GEOM_MATRIX_3x4_ROW_MAJOR,
    kMatrix4x4RowMajor = EDGE_GEOM_MATRIX_4x4_ROW_MAJOR,
    kMatrix4x4ColumnMajor = EDGE_GEOM_MATRIX_4x4_COLUMN_MAJOR,
};
```

Description

enum Value	Description
kMatrix3x4RowMajor	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.
kMatrix4x4RowMajor	Specifies "DirectX-style" row-major 4x4 matrices. This is provided primarily for convenience; obviously, 4x4 matrices use 33% more memory than 3x4 matrices.
kMatrix4x4ColumnMajor	Specifies "OpenGL-style" column-major 4x4 matrices. This is provided primarily for convenience; obviously, 4x4 matrices use 33% more memory than 3x4 matrices.

See Also

[EdgeGeomSegmentFormat](#), [EdgeGeomPartitionerInput](#), [edgeGeomMakeSpuConfigInfo](#), [edgeGeomMakeSkinningBuffer](#)

EdgeGeomCullingFlavor

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

Definition

```
enum EdgeGeomCullingFlavor
{
    kCullNone = EDGE_GEOM_CULL_NONE,
    kCullFrustum = EDGE_GEOM_CULL_FRUSTUM,
    kCullBackFacesAndFrustum = EDGE_GEOM_CULL_BACKFACES_AND_FRUSTUM,
    kCullFrontFacesAndFrustum = EDGE_GEOM_CULL_FRONTFACES_AND_FRUSTUM,
};
```

Description

enum Value	Description
kCullNone	Do not perform triangle culling.
kCullFrustum	Perform frustum and pixel center triangle culling.
kCullBackFacesAndFrustum	Perform back face, frustum, and pixel center triangle culling.
kCullFrontFacesAndFrustum	Perform front face, frustum, and pixel center triangle culling.

See Also

[EdgeGeomSegmentFormat](#), [EdgeGeomPartitionerInput](#), [edgeGeomGetScratchBufferSizeInQwords](#)