

# **PlayStation®Edgeジオメトリ ライブラリ リファレンス**

© 2010 Sony Computer Entertainment Inc.  
All Rights Reserved.  
SCE Confidential

# 目次

はじめに .....	4
このドキュメントについて .....	5
<b>実行時データ型 .....</b>	<b>6</b>
EdgeGeomSpuConfigInfo .....	7
EdgeGeomPpuConfigInfo .....	11
EdgeGeomBlendShapeInfo .....	14
EdgeGeomViewportInfo .....	15
EdgeGeomLocalToWorldMatrix .....	16
EdgeGeomSharedBufferInfo .....	17
EdgeGeomRingBufferInfo .....	18
EdgeGeomOutputBufferInfo .....	20
EdgeGeomAllocationInfo .....	22
EdgeGeomLocation .....	23
EdgeGeomCullingResults .....	24
EdgeGeomVertexStreamDescription .....	25
EdgeGeomCustomVertexFormatInfo .....	27
EdgeGeomCustomTransformVertexesForCullCallbackInfo .....	29
EdgeGeomSpuContext .....	30
<b>SPU実行時の関数 .....</b>	<b>31</b>
edgeGeomInitialize .....	32
edgeGeomValidateBufferOrder .....	34
edgeGeomGetSpuConfigInfo .....	36
edgeGeomGetViewportInfo .....	37
edgeGeomGetLocalToWorldMatrix .....	38
edgeGeomGetUniformTable .....	39
edgeGeomGetUniformTableByAttribute .....	40
edgeGeomGetUniformTables .....	41
edgeGeomGetUniformTableCount .....	42
edgeGeomAssignUniformTable .....	43
edgeGeomUnassignUniformTable .....	44
edgeGeomGetPositionUniformTable .....	45
edgeGeomSetPositionUniformTable .....	46
edgeGeomGetTransformUniformTable .....	47
edgeGeomGetNormalUniformTable .....	48
edgeGeomSetNormalUniformTable .....	49
edgeGeomGetTangentUniformTable .....	50
edgeGeomSetTangentUniformTable .....	51
edgeGeomGetBinormalUniformTable .....	52
edgeGeomSetBinormalUniformTable .....	53
edgeGeomGetIndexTable .....	54
edgeGeomGetIndexCount .....	55
edgeGeomSetIndexCount .....	56
edgeGeomGetVertexCount .....	57
edgeGeomSetVertexCount .....	58

edgeGeomGetFreePtr .....	59
edgeGeomSetFreePtr .....	60
edgeGeomIsAllocatedFromRingBuffer .....	61
edgeGeomGetOutputVertexStride .....	62
edgeGeomDecompressVertexes .....	63
edgeGeomDecompressIndexes .....	67
edgeGeomProcessBlendShapes .....	68
edgeGeomNormalizeUniformTable .....	69
edgeGeomSkinVertexes .....	70
edgeGeomTransformVertexes .....	71
edgeGeomCullOccludedTriangles .....	72
edgeGeomCullTriangles .....	74
edgeGeomCalculateDefaultOutputSize .....	76
edgeGeomAllocateOutputSpace .....	77
edgeGeomUseOutputSpace .....	79
edgeGeomOutputIndexes .....	80
edgeGeomCompressVertexes .....	81
edgeGeomOutputVertexes .....	84
edgeGeomBeginCommandBufferHole .....	85
edgeGeomSetVertexDataArrays .....	86
edgeGeomEndCommandBufferHole .....	87
<b>コールバック関数 .....</b>	<b>88</b>
EdgeGeomGetOutputVertexStrideCallback .....	89
EdgeGeomDecompressVertexStreamCallback .....	90
EdgeGeomBlendVertexStreamCallback .....	91
EdgeGeomCompressVertexStreamCallback .....	92
EdgeGeomSetVertexDataArraysCallback .....	93
EdgeGeomTransformVertexesForCullCallback .....	94
<b>定数 .....</b>	<b>95</b>
トライアングルカリングモード .....	96
スキニングモード .....	97
インデックスデータの種類 .....	98
スキニング行列フォーマット .....	99
ジオメトリ設定フラグ .....	100
出力モード .....	101

# はじめに

# このドキュメントについて

## 目的

このドキュメントは、Edge ライブラリのジオメトリコンポーネントの API リファレンスです。このコンポーネントは、スキニング、座標変換とトライアングルカリングなどを、SPU を使って効率的に実行するために使われます。

## 対象読者と前提条件

このドキュメントは、PlayStation®3 用の高性能アプリケーションを書こうとしている PlayStation®3 デベロッパのため書かれたものです。デベロッパは、以下の点に関して熟知していることを前提にしています。

- C と C++
- PlayStation®3 のハードウェア
- SCE の標準ライブラリ関数

## 関連ドキュメント

このリファレンスと以下のドキュメントを併用することにより、Edge ライブラリの使用法やリファレンスについての完全な情報を得ることができます。

- PlayStation® Edge ライブラリ概要
- PlayStation® Edge ジオメトリライブラリ クイックスタート
- PlayStation® Edge オフラインツール用ジオメトリライブラリ リファレンス
- PlayStation® Edge アニメーションライブラリ リファレンス
- PlayStation® Edge オフラインツール用アニメーションライブラリ リファレンス
- PlayStation® Edge Zlib ライブラリ リファレンス
- PlayStation® Edge LZMA ライブラリ リファレンス
- PlayStation® Edge LZ0 ライブラリ リファレンス
- PlayStation® Edge DXT ライブラリ リファレンス
- PlayStation® Edge Post ライブラリ リファレンス

## 表記法

このドキュメントでは、以下のような印刷上の表記法を使います。

記法	意味
等幅フォント	プログラミングコード、および処理命令、レジスタ名、データ型、イベント、そしてファイル名などのリテラルを表します。また、関数、構造体、マクロ名なども表します。
等幅フォント+太字	構造体や関数の定義の中でのみ、構造体や関数の名前を示します。
等幅フォント+斜体	引数、パラメータ、変数を表します。
<a href="#">青字 + 下線</a>	ハイパーリンクを表します(青色で表示されるのは、カラープリンタもしくはオンラインの場合だけです)。

## 実行時データ型

# EdgeGeomSpuConfigInfo

## SPU の処理設定を格納するための構造体

### 定 義

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

### メ ン バ

<i>flagsAndUniformTableCount</i>	上位 4 ビットには処理フラグ、下位 4 ビットには（ユニフォームテーブルの数-1）が格納される。 「ジオメトリ設定フラグ」を参照
<i>commandBufferHoleSize</i>	出力に必要とされるコマンドバッファホールの最大サイズ（クワッドワード）
<i>inputVertexFormatId</i>	EDGE_GEOM_SPU_VERTEX_FORMAT_* 列挙体からの、一次入力頂点ストリームのフォーマット ID。値が 0xFF の場合は、カスタム（非組み込み）フォーマットであることを示します
<i>secondaryInputVertexFormatId</i>	EDGE_GEOM_SPU_VERTEX_FORMAT_* 列挙体からの、二次入力頂点ストリームのフォーマット ID。このセグメントの二次頂点ストリームのサイズが 0 の場合、このフィールドは任意の値を取ることができます。値が 0xFF の場合は、カスタム（非組み込み）フォーマットであることを示します
<i>outputVertexFormatId</i>	EDGE_GEOM_RSX_VERTEX_FORMAT_* 列挙体からの、出力頂点ストリームのフォーマット ID。値が 0xFF の場合は、カスタム（非組み込み）フォーマットであることを示します
<i>unused</i>	未使用のデータフィールド
<i>indexesFlavorAndSkinningFlavor</i>	入力インデックスストリーム、およびスキニング計算のフレーバー。「 <a href="#">スキニングモード</a> 」ならびに「 <a href="#">インデックスデータの種類</a> 」を参照
<i>skinningMatrixFormat</i>	EDGE_GEOM_MATRIX_* 列挙型から選択されたスキニング行列のフォーマット ID（もしあれば）。このフィールドは、スキニングが行われない場合には無視されます
<i>numVertexes</i>	頂点の数。
<i>numIndexes</i>	インデックスの数。ユーザは <a href="#">edgeGeomInitialize()</a> を呼び出した後で <a href="#">edgeGeomGetIndexCount()</a> を呼び出して、インデックスの数にアクセスする必要があります
<i>indexesOffset</i>	このフィールドには、ほとんどのユーザが 0xFFFFFFFF を設定するはずです。 完全な説明については、下記参照

## 解 説

EdgeGeomSpuConfigInfo データ構造体は、設定情報の静的構造です。この設定情報は、Edge ジオメトリプロセスを初期化するために必要です。あらかじめセグメント化されたジオメトリの場合、この構造体は、オフラインツールで生成する必要があります。

*flagsAndUniformTableCount* の下位 4 ビットは、libedgegeomtool が処理中に割り当てる必要のあるユニフォームテーブルの数を示しています。この数は、頂点属性の数 - 1 と同じです（1~16 の数を 4 ビットに収めるため）。けれども、Edge 操作の中には、スクラッチ領域用に追加のユニフォームテーブルが必要なものもあります（スキニング、カリング、オクルージョン、ブレンド形状など）。このような操作を行う際には、ユニフォームテーブルの数を 1 つ増やす必要があります。上位 4 ビットは、処理フラグ用に予約されています。定義されているフラグは、現在のところ以下の通りです。

マクロ名	値	使用法
EDGE_GEOM_FLAG_STATIC_GEOMETRY_FAST_PATH	0x10	設定された場合、次の Edge 関数は即座にリターンします： edgeGeomSkinVertexes()、 edgeGeomProcessBlendShapes()、 edgeGeomCompressVertexes()、 edgeGeomOutputVertexes()
EDGE_GEOM_FLAG_INCLUDES_EXTRA_UNIFORM_TABLE	0x80	このセグメントでスキニング、カリング、オクルージョン、ブレンド形状を利用する場合、このような操作には、中間データを処理するために追加の一樣テーブルが必要なので、このフラグを設定する必要があります

*commandBufferHoleSize* は、グラフィックコマンドを（qword で）出力するために必要なコマンドバッファホール（最大サイズ）を表します。これは edgeGeomGetCommandBufferHoleSize() を用いてツール内で計算しなければなりません。

各種のフォーマット ID フィールドは、以下に示す組み込みの頂点フォーマットのリストに対するインデックスです。カスタムフォーマットの ID は -1（0xFF）です。この場合は、ユーザによって提供されたコールバックによって、またはオフラインツールで生成され SPU に入力データの一部としてアップロードされたストリーム記述構造体が存在する場合はこれによって処理されます。

一次と二次の頂点フォーマット ID は、以下に示す組み込みフォーマットのリストを参照します。

### 入カレイアウト 0：EDGE\_GEOM\_SPU\_VERTEX\_FORMAT\_F32c3

バイトオフセット	フォーマット	成分の数	属性 ID
0	F32	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION

### 入カレイアウト 1：EDGE\_GEOM\_SPU\_VERTEX\_FORMAT\_F32c3\_X11Y11Z10N\_X11Y11Z10N

バイトオフセット	フォーマット	成分の数	属性 ID
0	F32	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION
12	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL
16	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT

### 入カレイアウト 2：EDGE\_GEOM\_SPU\_VERTEX\_FORMAT\_F32c3\_X11Y11Z10N\_I16Nc4

バイトオフセット	フォーマット	成分の数	属性 ID
0	F32	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION
12	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL
16	I16N	4	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT



**入力レイアウト 3: EDGE\_GEOM\_SPU\_VERTEX\_FORMAT\_F32c3\_X11Y11Z10N\_X11Y11Z10N\_X11Y11Z10N**

バイトオフセット	フォーマット	成分の数	属性 ID
0	F32	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION
12	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL
16	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT
20	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_BINORMAL

**入力レイアウト 4: EDGE\_GEOM\_SPU\_VERTEX\_FORMAT\_FIXED\_UNITVEC\_UNITVEC**

バイトオフセット	フォーマット	成分の数	属性 ID
0	SW FIXED	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION
6	SW UNIT VEC	4	EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL
9	SW UNIT VEC	4	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT

**入力レイアウト 5: EDGE\_GEOM\_SPU\_VERTEX\_FORMAT\_FIXED\_UNITVEC\_UNITVEC\_UNITVEC**

バイトオフセット	フォーマット	成分の数	属性 ID
0	SW FIXED	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION
6	SW UNIT VEC	4	EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL
9	SW UNIT VEC	4	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT
12	SW UNIT VEC	4	EDGE_GEOM_ATTRIBUTE_USAGE_BINORMAL

**入力レイアウト 6: EDGE\_GEOM\_SPU\_VERTEX\_FORMAT\_EMPTY**

バイトオフセット	フォーマット	成分の数	属性 ID
属性は含まれません			

出力フォーマット ID は、以下に示す組み込みフォーマットのリストを参照します。

**出力レイアウト 0: EDGE\_GEOM\_SPU\_VERTEX\_FORMAT\_F32c3**

バイトオフセット	フォーマット	成分の数	属性 ID
0	F32	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION

**出力レイアウト 1: EDGE\_GEOM\_RSX\_VERTEX\_FORMAT\_F32c3\_X11Y11Z10N\_X11Y11Z10N**

バイトオフセット	フォーマット	成分の数	属性 ID
0	F32	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION
12	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL
16	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT

**出力レイアウト 2: EDGE\_GEOM\_RSX\_VERTEX\_FORMAT\_F32c3\_X11Y11Z10N\_I16Nc4**

バイトオフセット	フォーマット	成分の数	属性 ID
0	F32	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION
12	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL
16	I16N	4	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT

**出力レイアウト 3: EDGE\_GEOM\_RSX\_VERTEX\_FORMAT\_F32c3\_X11Y11Z10N\_X11Y11Z10N\_X11Y11Z10N**

バイトオフセット	フォーマット	成分の数	属性 ID
0	F32	3	EDGE_GEOM_ATTRIBUTE_USAGE_POSITION
12	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_NORMAL
16	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_TANGENT
20	X11Y11Z10N	3	EDGE_GEOM_ATTRIBUTE_USAGE_BINORMAL

## 出力レイアウト 4: EDGE\_GEOM\_RSX\_VERTEX\_FORMAT\_EMPTY

バイトオフセット	フォーマット	成分の数	属性 ID
属性は含まれません			

*indexesFlavorAndSkinningFlavor* の上位の 4 ビットは、インデックスデータストリームの種類を示し、下位 4 ビットは、この頂点データチャンクに対して実行すべきスキニング処理を示します。

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

インデックス、およびスキニングフレーバーとして可能な値に関しては「インデックスデータの種類」および「スキニングモード」を参照してください。

*skinningMatrixFormat* は、スキニング行列がある場合に、そのスキニング行列のフォーマットを指定します。可能な値については、「スキニング行列フォーマット」を参照してください。

*numVertexes* には、この特定のジョブの頂点の数、*numIndexes* には、インデックスの数を格納します。どちらも、通常はオフラインのツールで生成されます。ユーザは `edgeGeomInitialize()` を呼び出した後で、[edgeGeomGetIndexCount\(\)](#) を呼び出して、インデックスの数にアクセスする必要があります。

*indexesOffset* には、インデックスデータの RSX® IO オフセットを格納します。この情報は、元の入力インデックスデータに変更がない、つまり、インデックスデータの圧縮やカリングが行われておらず、元のインデックスバッファを変更しないでレンダリングできる場合に利用できます。*indexesOffset* フィールドに -1 (0xFFFFFFFF) を設定することによって Edge で生成したインデックスバッファを使うことができます（今のところより一般的）。

## 関 連 項 目

[edgeGeomInitialize](#)

# EdgeGeomPpuConfigInfo

各セグメントをセットアップするための設定情報を格納する構造体

## 定 義

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

## メ ン バ

<i>spuConfigInfo</i>	SPU の設定情報
<i>indexes</i>	セグメントのインデックスデータへのポインタ
<i>indexesSizes[2]</i>	セグメントのインデックスデータの DMA サイズ
<i>spuVertexes[2]</i>	セグメントの 2 つの SPU 入力頂点データストリームへのポインタ
<i>spuVertexesSizes[6]</i>	セグメントの 2 つの SPU 入力頂点データストリームの DMA サイズ 配列の最初の 3 つの要素は 1 番目のストリームに関するもので、配列の次の 3 つの要素は 2 番目のストリームに関するものです
<i>rsxOnlyVertexes</i>	このセグメントの RSX®専用の頂点ストリームへのポインタ。これらの属性は RSX®へ直接送信されなければならない、SPU 入力ストリームは SPU を通じて送信されます
<i>rsxOnlyVertexesSize</i>	このセグメントの RSX®専用の頂点ストリームのサイズ (バイト)
<i>skinMatricesByteOffsets[2]</i>	アップロードする行列の 2 つのセクションの、ベース行列ポインタからのバイトオフセット
<i>skinMatricesSizes[2]</i>	アップロードする行列の 2 つのセクションのサイズ

<i>skinIndexesAndWeightsSizes[2]</i>	インデックスおよび重みデータの DMA サイズ
<i>shapeSize</i>	ブレンド形状データの DMA サイズ
<i>shapeFixedOffsetsSize</i>	固定小数点圧縮オフセットの DMA サイズ
<i>skinIndexesAndWeights</i>	インタレースされたスキニングインデックスと重みデータへのポインタ
<i>ioBufferSize</i>	この SPURS ジョブに必要な入出力バッファサイズ
<i>scratchSize</i>	この SPURS ジョブに必要なクワッドワードのスクラッチサイズ
<i>numBlendShapes</i>	このセグメントに定義されたブレンド形状の数。 <i>blendShapes</i> および <i>blendShapeSizes</i> メンバ配列は、ここに示された数の要素を持ちます
<i>blendShapeSizes</i>	このセグメントの各ブレンド形状バッファのサイズ (バイト) を含む配列へのポインタ。この配列の要素数は <i>numBlendShapes</i> で示されます
<i>blendShapes</i>	このセグメントの各ブレンド形状バッファを指すポインタの配列。未使用エントリは NULL となります。この配列の要素数は <i>numBlendShapes</i> で示されます
<i>fixedOffsetsSize[2]</i>	各頂点ストリームの固定小数点オフセットデータの DMA サイズ
<i>fixedOffsets[2]</i>	伸長の際に固定小数点属性の偏りを元の範囲に戻すために使われる、各頂点ストリームの固定小数点オフセットデータへのポインタ
<i>spuInputStreamDescs</i>	2 つの SPU 入力頂点ストリームに対する、オプションのストリーム記述構造体へのポインタ。各エントリは、対応するストリームがカスタム頂点フォーマットを使っている場合にのみ NULL 以外となります
<i>spuOutputStreamDesc</i>	二次入力頂点ストリームの SPU 出力ストリーム記述構造体へのポインタ。SPU 出力ストリームがカスタム頂点フォーマットを使っている場合にのみ NULL 以外となります
<i>rsxOnlyStreamDesc</i>	RSX®のみの頂点ストリームの、オプションのストリーム記述構造体へのポインタ。 <i>rsxVertexesSize</i> がゼロ以外の場合にのみ NULL 以外となります
<i>spuInputStreamDescSizes</i>	SPU 入力ストリーム記述のサイズ (バイト)
<i>spuOutputStreamDescSize</i>	SPU 出力ストリーム記述のサイズ (バイト)
<i>rsxOnlyStreamDescSize</i>	RSX®専用ストリーム記述のサイズ (バイト)

## 解 説

この構造体は、SPU ジョブを初期化する実行時 PPU 処理用の情報を格納するために使われ、オフラインのツールで生成する必要があります。

これらの構造体は頂点データストリーム、およびスキニング行列データストリームとして、それぞれ 2 つの異なるストリームを表します。固定小数点オフセットテーブルにも、2 つの異なる種類を含んでいます。

たとえば、2 番目の頂点データストリームの 3 番目の DMA タグは、次のようになります。

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

この構造体のインデックスデータストリーム、および *skinIndexesAndWeight* データストリームは、どちらもバッファサイズを記述する配列のエントリが 2 つずつありますが、ストリーム自体は 1 つだけであることを注意してください。たとえば、インデックスデータストリームの 2 番目の DMA タグは、次のようになります。

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

注意： [EdgeGeomPpuConfigInfo](#)が、Edgeジオメトリランタイムによって直接使われることはありません。この構造体が提供されているのは、Edge ジオメトリジョブを作成するために必要となる可能性があるデータのすべてをカプセル化するのに便利だからです。この目的のために Edge サンプルプログラムによって使われます。ユーザは、自分のニーズにもっとも適した Edge トランスポート構造体を作成されることをお勧めします。

## **関 連 項 目**

[EdgeGeomSpuConfigInfo](#)

# EdgeGeomBlendShapeInfo

形状ブレンドの情報を格納する構造体

## 定 義

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

## メ ン バ

<i>dmaTag</i>	形状ブレンドデータの DMA を設定する DMA タグ
<i>アルファ</i>	この形状ブレンドのブレンド比率
<i>padding</i>	構造体のパディング（現在未使用）

## 解 説

EdgeGeomBlendShapeInfo 構造体は、実行時 SPU 処理で形状ブレンド操作を実行するために必要な形状データの情報が含まれています。この構造体は、適用するブレンド形状が存在しない場合には不要です。*dmaTag* フィールドは、実行時 SPU 処理が、形状ブレンドデータを収集するために、List DMA GET を発行する際に使われます。このフィールドは、通常、以下のように計算することができます。

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

## 関 連 項 目

[EdgeGeomPpuConfigInfo](#)、[edgeGeomProcessBlendShapes](#)

# EdgeGeomViewportInfo

トライアングルカリング操作のビューポート設定を渡すための構造体

## 定義

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

## メンバ

<code>scissorArea[4]</code>	<code>cellGcmSetScissor()</code> に渡されるパラメータ
<code>depthRange[2]</code>	<code>cellGcmSetViewport()</code> に渡される Z クリップの最小および最大パラメータ
<code>viewProjectionMatrix[16]</code>	頂点データをワールド空間からデバイス座標に変換する行列
<code>viewportScales[4]</code>	<code>cellGcmSetViewport()</code> に渡されるスケール値
<code>viewportOffsets[4]</code>	<code>cellGcmSetViewport()</code> に渡されるオフセット値
<code>sampleFlavor</code>	使用する MSAA の種類。 CELL_GCM_SURFACE_CENTER_1 CELL_GCM_SURFACE_DIAGONAL_CENTERED_2 CELL_GCM_SURFACE_SQUARE_CENTERED_4 CELL_GCM_SURFACE_SQUARE_ROTATED_4 この値は、ターゲット曲面と同じ値に設定する必要があります。 MSAA に関して詳しくは <i>libgcm</i> リファレンスを参照
<code>pad</code>	バイトアライメントのためのパディング

## 解説

[EdgeGeomViewportInfo](#) データ構造には、三角形カリングを実行するために必要な情報が含まれています。この構造体は、SPU 入力ユーザデータの一部として設定される `cullingFlavor` の中に、実行すべき三角形カリングが存在しない場合には、不要です。

---

# EdgeGeomLocalToWorldMatrix

---

座標変換を実行するために、SPU にローカル・ワールド間変換行列を渡すための構造体

## 定 義

---

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

## メ ン バ

---

*matrixData*      特定のジオメトリセグメントに対してローカル・ワールド間変換を行う 4×3 行列

## 解 説

---

ワールド・ローカル間変換行列（一般にモデル行列と呼ばれる）は、オブジェクトをローカル空間からワールド空間に変換する 4x3 行列です。この行列、PPU がモデルビュー射影行列全体を計算する必要をなくし、代わりにその一部 SPU に委譲できるようにして、PPU の計算負荷を軽くします。三角形カリングが不要な場合、この構造体は不要です。



# EdgeGeomSharedBufferInfo

全 SPU が使用する共有バッファ制御のための情報を含む構造体

## 定 義

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

## メ ン バ

<i>startEa</i>	共有バッファ先頭の実効アドレス
<i>startOffset</i>	共有バッファ先頭の IO オフセット
<i>endEa</i>	バッファの最後のアドレス (先頭+サイズ)
<i>currentEa</i>	バッファ空き領域先頭の実効アドレス
<i>locationId</i>	バッファの位置 (CELL_GCM_LOCATION_MAIN、もしくは、CELL_GCM_LOCATION_LOCAL)
<i>failedAllocSize</i>	各フレームで実現できなかった割り当ての合計サイズ
<i>pad1[2]</i>	バイトアライメントのためのパディング
<i>pad2[24]</i>	バイトアライメントのためのパディング

## 解 説

EdgeGeomSharedBufferInfo 構造体は、アトミックロックを使って全 SPU の間で共有される、単純な出力バッファを記述します。このバッファが一杯のときには、それ以降の割り当ては失敗します。*startEa* フィールドと *endEa* フィールドが同じである場合には、バッファは空もしくは存在しないものと見なされます。

この構造体は、主に、ダブルバッファ、シングルバッファ、およびハイブリッドバッファ方式で使われます。各バッファ方ごとに構造体を設定する方法については、「[EdgeGeomOutputBufferInfo](#)」のセクションを参照してください。

SPU が、共有バッファから空き出力領域を割り当てるのに失敗した場合、*failedAllocSize* の値は、要求された割り当てサイズの分だけインクリメントされます。この値を監視することにより、アプリケーションは、バッファが小さすぎないかどうか、小さすぎる場合には、さらにどのぐらい大きくすればよいかを判断することができます。

## 関 連 項 目

[EdgeGeomOutputBufferInfo](#)

# EdgeGeomRingBufferInfo

1 つの SPU が使用するリングバッファを制御するための情報を含む構造体

## 定 義

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

## メ ン バ

<i>startEa</i>	リングバッファ先頭の実効アドレス
<i>startOffset</i>	バッファ先頭の IO オフセット
<i>endEa</i>	バッファの最後のアドレス (先頭+サイズ)
<i>currentEa</i>	バッファ空き領域先頭のアドレス
<i>locationId</i>	バッファの場所。CELL_GCM_LOCATION_MAIN、CELL_GCM_LOCATION_LOCAL のどちらか
<i>rsxLabelEa</i>	同期に使われる RSX®ラベルの実効アドレス
<i>cachedFree</i>	キャッシュされたフリーポインタ (0 に初期化されている必要有り)
<i>pad[25]</i>	バイトアライメントのためのパディング

## 解 説

EdgeGeomRingBufferInfo 構造体は、事実上無限の出力バッファを記述します。このバッファは、いつループバックすれば安全かを知るために、RSX®と同期を行います。割り当て領域が不足している場合、SPU は RSX®が必要な領域を解放できるだけのデータを消費するまでブロックします。startEa フィールドと endEa フィールドが同じの場合には、バッファは空もしくは存在しないものと見なされます。この構造体は、主にリングバッファとハイブリッドバッファ方式で使われます。さまざまなバッファ方式でどのように構造体を設定するのは、「[EdgeGeomOutputBufferInfo](#)」のセクションを参照してください。rsxLabelEa フィールドは、SPU が、RSX® によるバッファデータの消費量を監視するために使う RSX®ラベルの、ローカルメモリ中のアドレスです。SPU は、それぞれ専用の RSX® ラベルを必要とします。

## 注 意

RSX®ラベルの ID 0~63 は、OS によって予約されています。ラベル ID 64~255 が、利用できます。ただし、現在のところ、RSX®ラベルの予約を一括で管理するシステムがないので、どのラベルが同じアプリケーションの別の部分によって使われているかを確認する方法はありません。RSX®ラベルに関するより詳しい情報については、*libgcm* の概要ドキュメントをお読みください。

EdgeGeom ジョブが複数のリングバッファを使う場合、SpuConfigInfo の中で指定されるコマンドバッファホールサイズを更新して、リングバッファが 1 つ追加されるごとに 16 バイト追加されるようにする

必要があります（逆に言えば、使われるリングバッファが1つだけである場合には、追加は不要です）。  
各リングバッファの RSX®ラベルを書き込むコマンドはそこに配置されます。

## **関 連 項 目**

---

[EdgeGeomOutputBufferInfo](#)

# EdgeGeomOutputBufferInfo

## アプリケーションの出力バッファ方式を記述する構造体

### 定義

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

### メンバ

<i>sharedInfo</i>	単純な共有バッファを表します。このバッファは全 SPU に共有されます
<i>ringInfo[6]</i>	SPURS によってアドレス指定可能な、SPU ごとに存在する 6 つのリングバッファを表します

### 解説

この構造体は、アプリケーションの出力バッファ方式を記述します。この中には、[EdgeGeomSharedBufferInfo](#) オブジェクト 1 つと、（アドレス指定可能な SPU ごとに 1 つの）6 つの [EdgeGeomRingBufferInfo](#) オブジェクトの配列 1 つの、2 つの要素が含まれています。*startEa* フィールドと *endEa* フィールドが同じである場合には、バッファは空もしくは存在しないものと見なされます。この構造体では、ダブルバッファ、シングルバッファ、リングバッファ、ハイブリッドバッファの 4 種類の異なるバッファ方式をサポートしています。

#### ダブルバッファを実装するための擬似コード：

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

#### シングルバッファを実装するための擬似コード：

このバッファは、SPU が使用する前に、フレームごとにリセットする必要があります。

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

---

```
cellGcmAddressToOffset(buffer, &info.sharedInfo.startOffset);
```

**リングバッファを実装するための擬似コード：**

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

**ハイブリッドバッファを実装するための擬似コード：**

ハイブリッドバッファの実装は、シングルバッファとリングバッファの両方を設定するのと同じぐらい簡単です。ランタイムは、設定された構造体の存在に応じて、インテリジェントにやるべきことを判断します。このシステムでは、間違った使い方というものはありませんが、これが最も推奨される方式です。

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

**関 連 項 目**

---

[edgeGeomAllocateOutputSpace](#)、[EdgeGeomAllocationInfo](#)

# EdgeGeomAllocationInfo

[edgeGeomAllocateOutputSpace\(\)](#)によって予約される割り当て領域を記述する構造体

## 定 義

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

## メ ン バ

<i>ea</i>	割り当てメモリの実効アドレス
<i>offset</i>	割り当て領域の RSX® オフセット
<i>location</i>	割り当て領域の RSX® 位置 (CELL_GCM_LOCATION_MAIN、もしくは、CELL_GCM_LOCATION_LOCAL)
<i>endEa</i>	割り当て領域の最後 (リングバッファのみ)
<i>rsxLabelEa</i>	RSX® ラベルの実効アドレス (リングバッファのみ)
<i>isVertexData</i>	<a href="#">edgeGeomOutputIndexes()</a> 、もしくは <a href="#">edgeGeomOutputVertexes()</a> に渡す場合には True

## 解 説

この構造体は、[edgeGeomAllocateOutputSpace\(\)](#)の呼び出しによって予約される割り当て領域を記述します (アプリケーションの出力バッファ方式)。この構造体には 6 つの要素がありましたが、そのうちの 2 つはリングバッファを利用する際にのみ使います。

*location* は、割り当て領域が、メインメモリと RSX® ローカルメモリのどちらに存在するかを表します。*isVertexData* と *rsxLabelEa* は、キャッシュされたデータがリングバッファから割り当てられた領域に格納されたときに、頂点キャッシュを無効にする必要があるかどうかを判定するために使われます。

## 関 連 項 目

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

---

# EdgeGeomLocation

---

CELL および RSX®の両方から見て、メモリ上のどこに対象が存在するかを記述する構造体

## 定 義

---

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

## メ ン バ

---

<i>ea</i>	割り当てメモリの実効アドレス
<i>offset</i>	割り当て領域の RSX® オフセット
<i>location</i>	割り当て領域の RSX® 位置 (CELL_GCM_LOCATION_MAIN、もしくは、CELL_GCM_LOCATION_LOCAL)

## 解 説

---

*location* は、割り当て領域が、メインメモリと RSX® ローカルメモリのどちらに存在するかを表します。

## 関 連 項 目

---

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

# EdgeGeomCullingResults

各テストケースによってカリングされた三角形の数を格納する構造体

## 定 義

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

## メ ン バ

<i>numOccludedTriangles</i>	オクルージョンテストによってカリングされた三角形の数
<i>numOutsideFrustumTriangles</i>	ビューフラスタムの外にある三角形の数
<i>numNoPixelTriangles</i>	標本点テストにパスしない三角形の数
<i>numBackFacingTriangles</i>	裏向きの三角形の数
<i>totalNumCulledTriangles</i>	カリングされた三角形の総数
<i>pad</i>	バイトアライメントのためのパディング

## 解 説

この構造体は、プロファイリング目的のために提供されています。この構造体を、Cバージョンのオクルージョン関数や三角形カリング関数で利用すると、構造体はテスト時にカリングされた三角形の数を含むように更新されます。カリングされた値は、カリングされた三角形の総数ですが、三角形の中には複数のテストに落ちるものもあるので、かならずしも構造体の他のメンバの和にはなりません。オクルージョンカリングされた三角形は、他のテストの対象にはなりません。

パフォーマンスのために、カリング結果は、以下の条件の両方が満たされない限り、記録されません。

- カリング関数のC実装が使われている (edgegeom\_config.h を参照)
- `EDGE_GEOM_DEBUG` が定義されている。

また、Edge カリング関数はカウンタをインクリメントするだけであることに注意してください。必要に応じて、カウンタをカリングの前にゼロに初期化するのは、ユーザ側の責任です。

## 関 連 項 目

edgeGeomCullTriangles、edgeGeomCullOccludedTriangles



# EdgeGeomVertexStreamDescription

コールバック関数や頂点ストリームの記述も含め、カスタムのフレーバ情報を保存する構造体

## 定 義

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

## メ ン バ

<i>numAttributes</i>	頂点ストリーム中の頂点属性の数
<i>stride</i>	ストリーム中の単一の頂点のサイズ（単位はバイト）。各頂点の中や後ろにパディングが追加されるため、このサイズは、個々の属性を合計したサイズよりも大きくなる場合があります。
<i>numBlocks</i>	<i>blocks</i> 配列中のエントリの数
<i>padding</i>	未使用
<i>blocks</i>	8 バイトの属性記述ブロックを保持する可変サイズの配列。構造体の後ろにはみ出ます。慣例により、この配列中の最初の <i>numAttributes</i> エントリは、EdgeGeomAttributeBlock オブジェクトです（ストリーム中の各頂点属性のためのオブジェクト）。残りの ( <i>numBlocks</i> - <i>numAttributes</i> ) エントリは、EdgeGeomAttributeFixedBlock オブジェクトです（ストリーム中の各固定小数点の頂点属性のためのオブジェクト）。

## 解 説

この構造体は、ストリーム中の各頂点属性についての情報も含め、頂点ストリームの構成を記述するためのものです。8 バイトのヘッダブロックの後ろに、8 バイトの属性データブロックが可変サイズで続きます。ヘッダブロックには、ストリームの頂点ストライドと頂点属性カウントとともに、可変サイズのブロック配列中にある 8 バイトのブロックの数が含まれます。

また *blocks* 配列には、2 種類のオブジェクトが含まれます。最初の *numAttributes* エントリは *EdgeGeomAttributeBlock* オブジェクトで、それぞれが 1 つの頂点属性を記述します。これらの属性ブロックの順番は定義されておらず、各頂点中の属性の順番とかならずしも関係ありません。それぞれの頂点ブロックには、属性のバイトオフセット、データ形式、要素数、Edge 属性 ID、バイト単位のサイズ、それに頂点プログラムスロットのインデックスに関する情報が含まれます。属性の種類が固定小数点の場合、属性ブロックには、(*blocks* 配列の先頭から) 関連する *EdgeGeomAttributeFixedBlock* へのバイトオフセットも含まれます。これらのブロックは *blocks[numAttributes]* から始まり、固定小数点属性の各要素に関する詳しい精度情報が含まれます。

構造体の定義では、技術的に *blocks* 配列のエントリは 0 である点に注意してください。ブロックは、個々の *EdgeGeomVertexStreamDescription* オブジェクトがメモリ中で連続することを保証しながら、構造体の後ろにはみ出します。この副作用として、*sizeof(EdgeGeomVertexStreamDescription)* は、ほとんど意味のない値となります（ヘッダブロックのサイズだけを考慮するため、常に 8 バイトを返します）。ストリームに関する記述の実際のサイズを求めるには、以下の式を利用してください。

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

# EdgeGeomCustomVertexFormatInfo

コールバック関数や頂点ストリーム記述などのカスタムフレーバー情報を格納する構造体

## 定義

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

## メンバ

<i>inputStreamDescA</i>	一次入力ストリーム記述構造体へのポインタ
<i>inputStreamDescB</i>	二次入力ストリーム記述構造体へのポインタ。二次頂点ストリームがない場合には、NULL を指定する必要があります
<i>outputStreamDesc</i>	出力ストリーム記述構造体へのポインタ
<i>blendStreamDesc</i>	ブレンド形状デルタストリーム記述構造体へのポインタ
<i>decompressInputCallbackA</i>	一次入力頂点ストリームを伸張するコールバック関数
<i>decompressInputCallbackB</i>	二次入力頂点ストリームを伸張するコールバック関数
<i>decompressBlendCallback</i>	入力頂点ストリームデータを伸張するコールバック関数
<i>outputStrideCallback</i>	カスタム出力フレーバー中における、頂点データのストライド長を計算するコールバック関数
<i>compressOutputCallback</i>	出力頂点データストリームを圧縮するコールバック関数
<i>setVertexDataArraysCallback</i>	頂点データストリームをセットアップして、コマンドバッファホールに書き込むグラフィックコマンドを生成するコールバック関数

## 解説

このコンテナ構造体には、ユーザが、関連するカスタム頂点フォーマット情報のすべてを書き込む必要があります。Edgeでは、頂点ストリームの圧縮・伸張方法として、ストリーム記述構造体、およびカスタムコールバック関数という2種類の方法をサポートしています。セグメントでカスタムフォーマットが使われていない（つまり、[EdgeGeomSpuConfigInfo](#)構造体に含まれているのが組み込みのフォーマットIDだけである）場合、この構造体のフィールドはすべて無視されるので、値は任意でかまいません（ただし、NULLが最も安全）。

カスタム頂点フォーマット用のストリーム記述構造体はツールによって生成されます。Edge ランタイムは、ストリーム記述を解析して、ストリームを適宜圧縮・伸張します。

またユーザは、代わりにさらに柔軟性が必要とされる場合、*decompressInputCallback*、*decompressBlendCallback*、*compressOutputCallback*を利用して、EDGE SPU ELF の中でリンクされるカスタム SPU コードで、対応する関数をオーバーロードすることもできます。

`compressOutputCallback`を提供する場合には、`outputStrideCallback`、および  
`setVertexDataArraysCallback`も提供する必要があります。  
特定の操作（入力、出力、ブレンディングなど）に対して、コールバックとストリーム記述の両方が提供  
された場合には、コールバックが優先されます。

---

# EdgeGeomCustomTransformVertexesForCullCallbackInfo

---

カスタム EdgeGeomTransformVertexesForCullCallback、およびユーザデータへのポインタを格納する構造体

## 定 義

---

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

## メ ン バ

---

<i>transformCallback</i>	頂点を変換するためのコールバック
<i>transformCallbackUserData</i>	コールバックに必要な任意のユーザデータへのポインタ

## 解 説

---

この構造体は、三角形カリングの前にカスタム頂点変換を行うために必要な、コールバックおよびユーザデータへのポインタが含まれます。この構造体へのポインタは、edgeGeomInitialize() に渡されるべきです。

## 関 連 項 目

---

edgeGeomInitialize, edgeGeomTransformVertexes, EdgeGeomTransformVertexesForCullCallback

---

# EdgeGeomSpuContext

---

SPU のコンテキスト固有のデータを格納する構造体

## 定 義

---

```
#include <edgegeom_structs.h>
struct EdgeGeomSpuContext
{
    /* 省略されています */
};
```

## 解 説

---

この構造体には、SPUのコンテキスト固有のデータが含まれています。アプリケーションは、この構造体に領域を割り当ててから、[edgeGeomInitialize\(\)](#) に渡して初期化する必要があります。

## 関 連 項 目

---

[edgeGeomInitialize](#)

## SPU実行時の関数

# edgeGeomInitialize

実行時 SPU ジオメトリ処理を初期化します。

## 定 義

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

## 引 数

<i>ctx</i>	SPU へのポインタ
<i>spuConfigInfo</i>	構成情報へのポインタ。 このデータはコンテキストにコピーされます
<i>scratchBuffer</i>	SPURS スクラッチバッファの先頭へのポインタ
<i>scratchBufferSize</i>	SPURS スクラッチバッファのサイズ (バイト)
<i>ioBuffer</i>	SPURS I/O バッファの先頭へのポインタ
<i>ioBufferSize</i>	SPURS I/O バッファのサイズ (バイト)
<i>dmaTag</i>	データ転送に使われる DMA タグ
<i>inViewportInfo</i>	ビューポート情報構造体へのポインタ。 このデータはコンテキストにコピーされます
<i>inLocalToWorldMatrix</i>	ローカル・ワールド間変換行列へのポインタ。 このデータはコンテキストにコピーされます
<i>customFormatInfo</i>	カスタム頂点フォーマット情報へのポインタ。 このデータはコンテキストにコピーされます
<i>customTransformInfo</i>	三角形カリングに使われるデフォルトの変換を置き換える、カスタム頂点変換用のコールバックおよびユーザデータへのポインタ
<i>gcmControlEa</i>	メインメモリ中の GCM 制御構造の実効アドレス

## 返 り 値

なし

## 解 説

この関数は、Edge ジオメトリライブラリを初期化する関数であり、いかなる処理操作の実行よりも先に、呼び出す必要があります。

この関数は、基本的に以下のようなことを行います。

- [EdgeGeomViewportInfo](#)をコンテキストにコピーします



- [EdgeGeomLocalToWorldMatrix](#)をコンテキストにコピーします
- [EdgeGeomSpuConfigInfo](#)をコンテキストにコピーします
- 固定オフセットをコンテキストにコピーします
- 頂点ユニフォームテーブルへのポインタを設定します
- ユーザ/カスタムコールバックを設定します
- **フリーポインタ**を I0 バッファの最後に設定します

フリーポインタや EdgeGeometry SPU ローカルストレージ管理に関する、より詳細な情報に関しては、「*PlayStation®EEEdge ライブラリ概要*」を参照してください。

# edgeGeomValidateBufferOrder

DMA リストの中に配置されたバッファの順序を検証します。

## 定 義

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

## 引 数

<i>pOutputStreamDesc</i>	出力データストリームの設定を表すストリーム記述構造体へのポインタ
<i>pIndexes</i>	入力インデックスデータへのポインタ
<i>pSkinMatrices</i>	入力スキニング行列へのポインタ
<i>pSkinWeights</i>	入力スキニング重みおよびインデックスへのポインタ
<i>pVertexesA</i>	一番目の頂点データストリームへのポインタ
<i>pVertexesB</i>	二番目の頂点データストリームへのポインタ
<i>pViewportInfo</i>	ビューポート情報構造体へのポインタ
<i>pLocalToWorld</i>	ローカル・ワールド間変換行列へのポインタ
<i>pSpuConfigInfo</i>	SPU 設定情報構造体へのポインタ
<i>pFixedOffsetsA</i>	伸長の際に固定小数点属性の偏りを元の範囲に戻すために使われる、固定小数点オフセットデータへのポインタ。これは 1 番目の頂点ストリーム用のデータです
<i>pFixedOffsetsB</i>	伸長の際に固定小数点属性の偏りを元の範囲に戻すために使われる、固定小数点オフセットデータへのポインタ。これは 2 番目の頂点ストリーム用のデータです
<i>pInputStreamDescA</i>	入力データストリーム A の設定を表すストリーム記述構造体へのポインタ
<i>pInputStreamDescB</i>	入力データストリーム B の設定を表すストリーム記述構造体へのポインタ

## 返 り 値

返り値 0 は、エラーや警告がないことを示しています。

それ以外の場合には、発生したエラー条件のコードすべての論理和を返します。以下の表は、エラー条件とコードの一覧を示しています。

マクロ	値	解説
EDGE_GEOM_VALIDATE_WARNING_VIEWPORT_INFO	0x00000001	パフォーマンス警告
EDGE_GEOM_VALIDATE_WARNING_LOCAL_TO_WORLD	0x00000002	パフォーマンス警告

マクロ	値	解説
EDGE_GEOM_VALIDATE_WARNING_SPU_CONFIG_INFO	0x00000004	パフォーマンス警告
EDGE_GEOM_VALIDATE_WARNING_FIXED_OFFSETS	0x00000008	パフォーマンス警告
EDGE_GEOM_VALIDATE_WARNING_CUSTOM_INPUT_FORMAT	0x00000010	パフォーマンス警告
EDGE_GEOM_VALIDATE_WARNING_FAST_PATH	0x00000020	パフォーマンス警告
EDGE_GEOM_VALIDATE_ERROR_SKINNING_MATRICES	0x00010000	致命的エラーの原因になります
EDGE_GEOM_VALIDATE_ERROR_SKINNING_WEIGHTS	0x00020000	致命的エラーの原因になります
EDGE_GEOM_VALIDATE_ERROR_INDEXES	0x00040000	致命的エラーの原因になります
EDGE_GEOM_VALIDATE_ERROR_CUSTOM_OUTPUT_FORMAT	0x00080000	致命的エラーの原因になります
EDGE_GEOM_VALIDATE_ERROR_FAST_PATH	0x00100000	致命的エラーの原因になります

## 解 説

この関数は、バッファ順序の妥当性をチェックします。バッファの配置ミスは、致命的エラーになることもあれば、単なるパフォーマンス警告で済むこともあります。正しいバッファ順序に関する詳しい情報については、「*PlayStation® Edge ライブラリ概要*」を参照してください。

以下のマクロを使うと、関数によって報告されたのがエラーなのか警告なのかを、すばやく調べることができます。

マクロ	値
EDGE_GEOM_VALIDATE_ERROR_MASK	0xFFFF0000
EDGE_GEOM_VALIDATE_WARNING_MASK	0x0000FFFF

## 注 意

この関数がバッファの順序を検証するのは、Edge ジオメトリをビルドする際に EDGE\_GEOM\_DEBUG が定義された場合だけです。

---

# edgeGeomGetSpuConfigInfo

---

SPU 構成情報のローカルコピーを取得します。

## 定 義

---

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

## 引 数

---

<code>ctx</code>	コンテキストへのポインタ
------------------	--------------

## 返 り 値

---

コンテキストの SPU 設定情報へのポインタを返します。

## 解 説

---

この関数は、初期化に渡された [EdgeGeomSpuConfigInfo](#) のコンテキストのコピーへのポインタを返します。

---

# edgeGeomGetViewportInfo

---

ビューポート情報のローカルコピーを取得します。

## 定 義

---

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

## 引 数

---

<code>ctx</code>	コンテキストへのポインタ
------------------	--------------

## 返 り 値

---

ビューポート情報のコンテキストのコピーを返します。

## 解 説

---

初期化に渡された [EdgeGeomViewportInfo](#) のコンテキストのコピーへのポインタを返します。

---

# edgeGeomGetLocalToWorldMatrix

---

ローカル・ワールド間変換行列のローカルコピーを取得します。

## 定 義

---

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

## 引 数

---

<code>ctx</code>	コンテキストへのポインタ
------------------	--------------

## 返 り 値

---

ローカル・ワールド間変換行列のコンテキストのコピーを返します。

## 解 説

---

初期化に渡された [edgeGeomGetLocalToWorldMatrix](#) のコンテキストのコピーへのポインタを返します。

---

# edgeGeomGetUniformTable

---

指定されたユニフォームテーブルを取得します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>index</i>	ユニフォームテーブルのインデックス

## 返 り 値

---

スラッシュメモリ中の指定されたユニフォームテーブルを返します。インデックスは、一様テーブル配列への絶対インデックスです。特定の頂点属性の一様テーブルを取得するには、[edgeGeomGetUniformTableByAttribute\(\)](#)を使います。

## 解 説

---

ユニフォームテーブルは、[edgeGeomInitialize\(\)](#)関数に渡されたスラッシュバッファの先頭から配置されます。

このテーブル中のデータの実際の初期化は、[edgeGeomDecompressVertexes\(\)](#)関数の中で行われます。

ユニフォームテーブルの順序は、頂点データ入力フレーバーによって異なります。テーブルは、伸張されたときの順序で、属性に割り当てられます。

## 注 意

---

実行時のパフォーマンスを維持するため、この関数の中にはエラーチェックはありません。

## 関 連 項 目

---

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

---

# edgeGeomGetUniformTableByAttribute

---

指定された一様テーブルを取得します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>attrId</i>	一様テーブルを返す属性 ID

## 返 り 値

---

指定された頂点属性に対応するスクラッチメモリ中の一様テーブルを返します。特定の絶対インデックスをもつ一様テーブルにアクセスするには、[edgeGeomGetUniformTable\(\)](#)を使います。

## 解 説

---

一様テーブルは、[edgeGeomInitialize\(\)](#)に渡されたスクラッチバッファの先頭から始まるように配置されます。

このテーブル中のデータの実際の初期化は、[edgeGeomDecompressVertexes\(\)](#)関数の中で行われます。

## 注 意

---

実行時のパフォーマンスを維持するため、この関数の中にはエラーチェックはありません。

## 関 連 項 目

---

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



## 定義

## 引 数

**返 り 値**

## 解説

## 關 連 項 目

- 41 -

## 定義

## 引 数

**返　り　値**

## 解説

## 關 連 項 目

- 42 -

---

# edgeGeomAssignUniformTable

---

利用できる次の（未割り当ての）一様テーブルを、指定された頂点属性 ID に割り当てます。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>attrId</i>	利用できる次の一様テーブルに割り当てる頂点属性 ID

## 返 り 値

---

成功した場合の返り値は、新たに割り当てられた一様テーブルのインデックス（常に 0～16）になります。利用できる一様テーブルが残っていない場合には、この関数は-1 を返します。

## 解 説

---

この関数は、利用できる最初の一様テーブルを探し、それを指定された頂点属性 ID に割り当てます。この関数は、通常呼び出す必要はありません。なぜなら、Edge ジオメトリは、頂点伸長処理の際に、関連する一様テーブルのすべてを自動的に設定するからです。けれども、ユーザが実行時に SPU 上で新しい頂点属性を生成する場合（接線と法線の外積をとることにより従法線テーブルを作成する場合など）、従法線属性にテーブルを割り当てるために、この関数を呼び出す必要があります。

## 注 意

---

この関数は、新しい一様テーブルを割り当てません。テーブルの数は、初期化時に、EdgeGeomSpuConfigInfo の *flagsAndUniformTableCount* メンバによって固定されます。新しい一様テーブルをマップする前に、未使用の一様テーブルの割り当てを解除する必要がある場合があります。

## 関 連 項 目

---

[edgeGeomUnassignUniformTable](#)

---

# edgeGeomUnassignUniformTable

---

指定された頂点属性一様テーブルをアンマップして、別の属性に割り当て直せるようにします。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>attrId</i>	一様テーブルを割り当てなおす必要のある頂点属性の ID

## 返 り 値

---

成功した場合の返り値は、新たに割り当て解除された一様テーブルのインデックス（常に 0～16）になります。指定された属性 ID が見つからない場合には、この関数は-1 を返します。

## 解 説

---

この関数は、指定された頂点属性IDに関連付けられた一様テーブルを特定して、そのテーブルを未使用のものとしてマークします。これにより、edgeGeomAssignUniformTable [\(Q\)](#) を使ってテーブルを新しい頂点属性に割り当て直すことが可能になります。

## 関 連 項 目

---

[edgeGeomAssignUniformTable](#)

## edgeGeomGetPositionUniformTable

位置データと関連付けられたユニフォームテーブルを取得します。

## 定義

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

## 引 数

ctx	コンテキストへのポインタ
-----	--------------

返 り 値

スクラッチメモリ中の位置ユニフォームテーブルへのポインタを返します。

## 解説

この関数は、頂点位置ユニフォームテーブルへのポインタを返します。  
このテーブルのデータは、入力データが正しく伸張されていれば、`vec_float4` フォーマットになっているはずです。

## 注意

この関数が有効なのは、頂点の伸長や、[edgeGeomSetPositionUniformTable\(\)](#)による位置ユニフォームテーブルの明示的な設定の後に呼び出された場合だけです。

## 關 連 項 目

edgeGeomSetPositionUniformTable

---

# edgeGeomSetPositionUniformTable

---

位置データと関連付けられたユニフォームテーブルを設定します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>table</i>	指定された位置ユニフォームテーブルへのポインタ

## 返 り 値

---

なし

## 解 説

---

ユーザは、この関数を呼び出すことにより、コンテキストの位置ユニフォームテーブルポインタを、指定された場所に設定することができます。

このポインタは、ほとんどの環境では、Edge ランタイムによって適切に設定されるので、この呼び出しは通常不要です。

この内部的な位置ユニフォームテーブルのポインタは、ライブラリの他の関数によって使用されます。たとえば [edgeGeomProcessBlendShapes\(\)](#)、[edgeGeomCullOccludedTriangles\(\)](#)、[edgeGeomCullTriangles\(\)](#) など、頂点位置データを参照する関数です。

## 関 連 項 目

---

[edgeGeomGetPositionUniformTable](#)

---

# edgeGeomGetTransformUniformTable

---

変換後の位置と関連付けられたユニフォームテーブルを取得します。

## 定 義

---

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

## 引 数

---

*ctx*                      コンテキストへのポインタ

## 返 り 値

---

変換後の位置と関連付けられたユニフォームテーブルを返します。

## 解 説

---

この関数は、変換後の位置を含むユニフォームテーブルを返します。  
このテーブルが有効なのは、カリングフラグをEDGE\_GEOM\_CULL\_NONEに設定せず  
に、[edgeGeomCullOccludedTriangles\(\)](#) または [edgeGeomCullTriangles\(\)](#) 関数を呼び出した後だけです。

## 注 意

---

この変換後の位置テーブルは、かならず、ユニフォームテーブルの中の最後にあるはずです。  
通常、このユニフォームテーブルに含まれるデータが有効なのは、  
[edgeGeomCullOccludedTriangles\(\)](#)、[edgeGeomCullTriangles\(\)](#) の後だけです。

## 関 連 項 目

---

[edgeGeomCullOccludedTriangles](#)、[edgeGeomCullTriangles](#)

---

# edgeGeomGetNormalUniformTable

---

法線と関連付けられたユニフォームテーブルを取得します。

## 定 義

---

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

## 引 数

---

<code>ctx</code>	コンテキストへのポインタ
------------------	--------------

## 返 り 値

---

法線データと関連付けられたユニフォームテーブルを返します。

## 解 説

---

この関数は、法線を含むユニフォームテーブルへのポインタを返します。

この法線ユニフォームテーブルが利用できるのは、[edgeGeomDecompressVertexes\(\)](#) 関数を呼び出した後です。このポインタは、ほとんどの環境では、Edgeランタイムによって適切に設定されるので、この呼び出しは通常不要です。

## 注 意

---

この関数が有効なのは、頂点の伸長の後か、[edgeGeomSetNormalUniformTable\(\)](#) を明示的に呼び出した後だけです。

## 関 連 項 目

---

[edgeGeomSetNormalUniformTable](#)、[edgeGeomDecompressVertexes](#)



---

# edgeGeomSetNormalUniformTable

---

頂点法線と関連付けられたユニフォームテーブルを設定します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>table</i>	ユニフォームテーブルへのポインタ

## 返 り 値

---

なし

## 解 説

---

この関数は、そのコンテキストの法線ユニフォームテーブルへのポインタを設定します。この関数を使うと、ユーザ独自の法線テーブルを設定することができます。このポインタは、ほとんどの環境では、Edgeランタイムによって適切に設定されるので、この呼び出しは通常不要です。

## 関 連 項 目

---

[edgeGeomGetNormalUniformTable](#)

---

# edgeGeomGetTangentUniformTable

---

頂点接線と関連付けられたユニフォームテーブルを取得します。

## 定 義

---

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

## 引 数

---

*ctx*                      コンテキストへのポインタ

## 返 り 値

---

頂点接線と関連付けられたユニフォームテーブルを返します。

## 解 説

---

この関数は、接線データを含むユニフォームテーブルへのポインタを返します。

## 注 意

---

返されたポインタに含まれるデータが有効なのは、頂点の伸長や、[edgeGeomSetTangentUniformTable\(\)](#) を明示的に呼び出した後だけです。

## 関 連 項 目

---

[edgeGeomSetTangentUniformTable](#)

---

# edgeGeomSetTangentUniformTable

---

頂点法線と関連付けられたユニフォームテーブルを取得します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>table</i>	ユニフォームテーブルへのポインタ

## 返 り 値

---

なし

## 解 説

---

この関数は、コンテキストの接線テーブルポインタを、指定した接線テーブルを指すように設定できます。このポインタは、ほとんどの環境では、Edge ランタイムによって適切に設定されるので、この呼び出しは通常不要です。

## 関 連 項 目

---

[edgeGeomGetTangentUniformTable](#)

## edgeGeomGetBinormalUniformTable

頂点従法線と関連付けられたユニフォームテーブルを取得します。

## 定義

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

## 引 数

<code>ctx</code>	コンテキストへのポインタ
------------------	--------------

**返　り　値**

頂点従法線と関連付けられたユニフォームテーブルを返します。

## 解説

この関数は、従法線データを含むユニフォームテーブルを返します。

## 注意

返されたポインタに含まれるデータが有効なのは、頂点の伸長後か、[edgeGeomSetBinormalUniformTable\(\)](#)を明示的に呼び出した後だけです。

## 關 連 項 目

edgeGeomSetBinormalUniformTable

---

# edgeGeomSetBinormalUniformTable

---

頂点従法線に関連付けられたユニフォームテーブルを設定します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>table</i>	ユニフォームテーブルへのポインタ

## 返 り 値

---

なし

## 解 説

---

この関数は、コンテキストの従法線テーブルポインタを、指定した従法線テーブルを指すように設定します。

このポインタは、ほとんどの環境では、Edge ランタイムによって適切に設定されるので、この呼び出しは通常不要です。

## 関 連 項 目

---

[edgeGeomGetBinormalUniformTable](#)

## edgeGeomGetIndexTable

伸張されたインデックステーブルへのポインタを返します。

## 定義

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

## 引 数

ctx	コンテキストへのポインタ
-----	--------------

返り値

伸張されたインデックステーブルへのポインタ。この関数が [edgeGeomDecompressIndexes\(\)](#) より先に呼び出された場合、返り値は未定義です。

### 注意

インデックステーブル内の要素数は、`edgeGeomGetIndexCount()`を呼び出せば、いつでも好きなときに取得することができます。また、オクルージョンカリング後の要素数は [`edgeGeomCullOcludedTriangles\(\)`](#)、三角形カリング後の要素数は [`edgeGeomCullTriangles\(\)`](#) によって返されます。

## 關 連 項 目

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

---

# edgeGeomGetIndexCount

---

インデックステーブル内のインデックスの数を返します。

## 定 義

---

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

## 引 数

---

<code>ctx</code>	コンテキストへのポインタ
------------------	--------------

## 返 り 値

---

インデックステーブル内のインデックスの数。

## 解 説

---

この関数は、ユーザに対し、内部的に保持されたインデックステーブル内のインデックス数へのアクセスを提供します。この数は、[edgeGeomCullOccludedTriangles\(\)](#)、および [edgeGeomCullTriangles\(\)](#) によって更新されます。初期値は、[edgeGeomInitialize\(\)](#) に渡された [EdgeGeomSpuConfigInfo](#) からコピーされます。ユーザがインデックス数を変更した場合には、かならず、[edgeGeomSetIndexCount\(\)](#) を呼び出すことによって、Edgeの内部変数に反映させる必要があります。

## 関 連 項 目

---

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

---

# edgeGeomSetIndexCount

---

インデックステーブル内のインデックスの数を設定します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>count</i>	新しいインデックス数

## 返 り 値

---

なし

## 解 説

---

この関数は、ユーザに対し、内部的に保持されたインデックステーブル内のインデックス数を設定する手段を提供します。ユーザは、インデックス数を変更するようなユーザ処理を行った後に、かならず、この関数でインデックス数を更新する必要があります。現在のインデックス数には、[edgeGeomGetIndexCount\(\)](#)によってアクセスできます。

## 関 連 項 目

---

[edgeGeomGetIndexCount](#)



---

# edgeGeomGetVertexCount

---

頂点テーブル内の頂点の数を返します。

## 定 義

---

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

## 引 数

---

<code>ctx</code>	コンテキストへのポインタ
------------------	--------------

## 返 り 値

---

頂点テーブル内の頂点の数。

## 解 説

---

この関数は、ユーザに対し、内部的に保持された頂点テーブル内の頂点数へのアクセスを提供します。初期値は、[edgeGeomInitialize\(\)](#)に渡された[EdgeGeomSpuConfigInfo](#)からコピーされます。ユーザが頂点数を変更した場合には、[edgeGeomSetVertexCount\(\)](#)呼び出しによって、Edge内に通知する必要があります。

## 関 連 項 目

---

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

---

# edgeGeomSetVertexCount

---

頂点テーブル内の頂点の数を設定します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>count</i>	新しい頂点の数

## 返 り 値

---

なし。

## 解 説

---

この関数は、ユーザに対し、内部的に保持されたインデックステーブル内の頂点数を設定する手段を提供します。ユーザは、頂点数を変更するようなユーザ処理を行った後に、かならず、この関数で頂点数を更新する必要があります。現在の頂点数には、`edgeGeomGetVertexCount()` からアクセスすることができます。

## 関 連 項 目

---

[edgeGeomGetVertexCount](#)

---

# edgeGeomGetFreePtr

---

SPURS IO バッファメモリの使用中の最後の位置を示すポインタを取得します。

## 定 義

---

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

## 引 数

---

<code>ctx</code>	コンテキストへのポインタ
------------------	--------------

## 返 り 値

---

SPURS IO バッファメモリの使用中の最後の位置を示すポインタを返します。

## 解 説

---

この関数は、現在のフリーポインタを返します。フリーポインタに関する詳細は「*PlayStation®Edge ライブラリ概要*」を参照してください。

このフリーポインタは、入出力バッファの「空き領域の先頭」を指しています。

たとえば、普通は、[edgeGeomDecompressVertexes\(\)](#) 関数を呼び出した後に、圧縮後の入力頂点データをIO バッファに保持する必要はありません。したがって、フリーポインタは、[edgeGeomDecompressVertexes\(\)](#) 関数の最後の入力頂点バッファの先頭に設定されます。

## 注 意

---

この関数は、[edgeGeomInitialize\(\)](#) が終了しないと有効になりません。

## 関 連 項 目

---

[edgeGeomInitialize](#)、[edgeGeomDecompressVertexes](#)

---

# edgeGeomSetFreePtr

---

フリーポインタを指定されたアドレスに設定します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>ptr</i>	設定するポインタ

## 返 り 値

---

なし

## 解 説

---

この関数は、ユーザは、現在のフリーポインタを、特定のバッファを指すように設定します。

## 注 意

---

このポインタの指すバッファは、ライブラリによって内部的に使われます。空き領域の詳細については、[edgeGeomGetFreePtr](#)のセクションと「PlayStation®Edgeライブラリ概要」を参照してください。

## 関 連 項 目

---

[edgeGeomGetFreePtr](#)

---

# edgeGeomIsAllocatedFromRingBuffer

---

出力領域がリングバッファから割り当てられるかどうかを調べます。

## 定 義

---

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

## 引 数

---

*info*                      割り当て情報構造体へのポインタ

## 返 り 値

---

*info* の指す領域がリングバッファから割り当てられている場合には true を返します。それ以外の場合は false を返します。

## 解 説

---

この関数は、出力領域がリングバッファから割り当てられるかどうかを判定します。この関数は、RSX® にキャッシュされたデータを明示的に無効化する必要があるかどうかを判断する際に便利です。

## 注 意

---

Edge ジオメトリのデフォルトの使用パターンでは、この関数を呼び出す必要はありません。リングバッファを使っている場合、Edge は、頂点キャッシュの無効化を内部的に管理します。けれども、頂点やテクスチャのデータを出力に割り当てているのがユーザコードである場合には、リングバッファが使われているか、したがってその対応するキャッシュを無効にする必要があるかどうかを、この関数を使って判断することができます。

## 関 連 項 目

---

[edgeGeomAllocateOutputSpace](#)

---

# edgeGeomGetOutputVertexStride

---

指定された出力フレーバーのストライドサイズを取得します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>outputFormatId</i>	ストライドを取得する RSX®出力頂点フォーマットの ID

## 返 り 値

---

単一の出力頂点のサイズをバイトで返します。内部に出力フォーマットがなく、出力ストライドコールバック関数が [EdgeGeomSpuContext](#) に登録されてない場合、0 を返します。

## 解 説

---

この関数は、指定された出力頂点フォーマットが内部的に存在する場合には、その出力フレーバーに適したストライドを返します。それ以外の場合には、この関数は、出力ストライドコールバック関数 ([EdgeGeomGetOutputVertexStrideCallback\(\)](#)) が設定されていることを確認して、呼び出します。コールバック関数が存在しない場合には、この関数は 0 を返します。

# edgeGeomDecompressVertexes

EdgeGeomSpuConfigInfo で指定された入力頂点フォーマット ID に従って入力頂点データを伸張します。

## 定 義

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

## 引 数

<i>ctx</i>	コンテキストへのポインタ。
<i>vertexesA</i>	1 番目の入力頂点データストリームへのポインタ。
<i>fixedOffsetsA</i>	1 番目の頂点ストリームを伸張する際に使われる、固定オフセットバッファへのポインタ。
<i>vertexesB</i>	2 番目の入力頂点データストリームへのポインタ。この値は、2 番目の入力頂点ストリームがない場合には、0 でもかまいません。
<i>fixedOffsetsB</i>	2 番目の頂点ストリームを伸張する際に使われる、固定オフセットバッファへのポインタ。

## 返 り 値

なし

## 解 説

この関数は、まず、[edgeGeomInitialize\(\)](#) 関数の引数として指定された、コンテキストの [EdgeGeomSpuConfigInfo](#) 変数から、入力フォーマット ID を抽出します。

この関数は、伸長を済んだ後、フリーポインタを入力頂点バッファの先頭に設定します。

*fixedOffsets\** 引数は、このセグメントのためにツールが用意した、固定小数点オフセットのテーブルを参照している必要があります。このテーブルには、入力プレーバー中の固定小数点属性ごとに、4 つの浮動小数点オフセットが含まれています。これらのオフセットは、元の float に変換されてから、伸張後の固定小数点属性値に加算され、属性値を元の正しい範囲に戻します。

**注意：** 静的にリンクされた新しい入力頂点フォーマットへのサポートを追加するには、内部関数 [edgeGeomDecompressVertexes\(\)](#) の中のスイッチ文に新しい case ブロックを加える必要があります。この case ブロック内で使われているマクロ (edgegeom\_decompress.h で定義された) は、ほぼ最適な伸長コードの明確かつ簡潔な実装を可能にします。

**注意：** EDGE\_GEOM\_DECOMPRESS マクロはコンパイル時定数を活用しているので、最適化コンパイラは可能な限り事前計算を行うことができます。したがって、このマクロの引数がすべてコンパイル時定数 (数値定数もしくは #define 定数) になっている必要があります。実際には、コンパイル時にパラメータ値はわかっているはずなので、このことは問題にはならないはずです。

case ブロックの中で呼び出す必要がある最初のマクロは、EDGE\_GEOM\_DECOMPRESS\_INIT\_GLOBAL です。このマクロの唯一のパラメータは、頂点ストライドの合計です。

次に、伸張する属性を列挙します。伸張マクロでサポートしている属性の種類は、以下の通りです。

属性種別	解説
F32	32 ビット浮動小数点
F16	16 ビット浮動小数点
U8	8 ビット符号付き整数
I16	16 ビット符号付き整数
U8N	8 ビット符号なし整数。[0.0..1.0]の範囲を表すように正規化。
I16N	16 ビット符号付き整数。[-1.0..1.0]の範囲を表すように正規化。
X11Y11Z10N	32 ビットのパックされた 3 成分ベクトル 基本的に、X 用の I11N、s 用の I11N、Z 用の I10N が、単一の 32 ビット値にパックされます
FIXED_POINT	各ベクトル成分を固定小数点値にエンコードする Edge 独自のフォーマットです。詳しくは、「PlayStation®Edge ライブラリ概要」の「ジオメトリデータ圧縮」のデータ圧縮に関するセクションを参照してください。
UNIT_VECTOR	単位ベクトルを 24 ビットにパックする Edge 独自のフォーマット。詳しくは、「PlayStation®Edge ライブラリ概要」のデータ圧縮に関するセクションを参照してください。

各属性について、下記の一覧の中から対応するマクロを、属性が頂点の中に出現する順序で呼び出します。

- EDGE\_GEOM\_DECOMPRESS\_INIT\_F32(attrId, attrOffset, attrComponentCount)
- EDGE\_GEOM\_DECOMPRESS\_INIT\_F16(attrId, attrOffset, attrComponentCount)
- EDGE\_GEOM\_DECOMPRESS\_INIT\_U8(attrId, attrOffset, attrComponentCount)
- EDGE\_GEOM\_DECOMPRESS\_INIT\_I16(attrId, attrOffset, attrComponentCount)
- EDGE\_GEOM\_DECOMPRESS\_INIT\_U8N(attrId, attrOffset, attrComponentCount)
- EDGE\_GEOM\_DECOMPRESS\_INIT\_I16N(attrId, attrOffset, attrComponentCount)
- EDGE\_GEOM\_DECOMPRESS\_INIT\_X11Y11Z10N(attrId, attrOffset, attrComponentCount)
- EDGE\_GEOM\_DECOMPRESS\_INIT\_FIXED\_POINT(attrId, attrOffset, attrComponentCount, integerBits1, mantissaBits1, integerBits2, mantissaBits2, integerBits3, mantissaBits3, integerBits4, mantissaBits4)
- EDGE\_GEOM\_DECOMPRESS\_INIT\_UNIT\_VECTOR(attrId, attrOffset, attrComponentCount)

上記のマクロは、以下のようなパラメータをとります。

attrId                      EDGE\_GEOM\_ATTRIBUTE\_ID\_\* 列挙型の属性を一意に特定する数値。  
attrOffset                  属性の頂点中でのバイトオフセット。  
attrComponentCount      属性中の成分の数。[1-4]の範囲に入っている必要があります。

EDGE\_GEOM\_DECOMPRESS\_INIT\_FIXED\_POINT はさらに、各成分の整数成分および仮数成分のビット深度を記述する 8 つのパラメータを受け取ります。属性によって使われるビット数の合計は、8 で割り切れる必要があります。使用しない成分には、両方のビット深度に 0 を使います。

全属性の初期化が済んだら、入力ストリーム中の各頂点についてループを実行するメインの do-while ループに入る前に、EDGE\_GEOM\_DECOMPRESS\_LOAD\_COMMON() を呼び出す必要があります。ループ条件としては、!EDGE\_GEOM\_DECOMPRESS\_LOOP\_DONE() を使います。



ループの中では、まず、EDGE\_GEOM\_DECOMPRESS\_LOOP\_START() を呼び出して、ループ本体の準備を行います。

次に、各属性を伸張します。各属性について、下記の一覧の中から対応するマクロを呼び出します(*attrId* パラメータの意味は、前と同じです)。

- EDGE\_GEOM\_DECOMPRESS\_LOOP\_F32(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_LOOP\_F16(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_LOOP\_U8(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_LOOP\_I16(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_LOOP\_U8N(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_LOOP\_I16N(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_LOOP\_X11Y11Z10N(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_LOOP\_FIXED\_POINT(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_LOOP\_UNIT\_VECTOR(*attrId*)

EDGE\_GEOM\_DECOMPRESS\_LOOP\_END() を呼び出してループを終了します。

ループが終わったら、各属性型に対して適切なマクロを呼び出して、伸長処理を完成させる必要があります。以下のリストは、各属性型に対応する完了処理マクロです(*attrId* パラメータの意味は以前と同じ)

- EDGE\_GEOM\_DECOMPRESS\_FINALIZE\_F32(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_FINALIZE\_F16(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_FINALIZE\_U8(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_FINALIZE\_I16(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_FINALIZE\_U8N(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_FINALIZE\_I16N(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_FINALIZE\_X11Y11Z10N(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_FINALIZE\_FIXED\_POINT(*attrId*)
- EDGE\_GEOM\_DECOMPRESS\_FINALIZE\_UNIT\_VECTOR(*attrId*)

以下は、頂点ストライドが 30 バイトで、5 つの属性を含む入力ストリーム全体の例です。3 成分の固定小数点位置 (x=12.20、y=12.20、z=4.20)、3 成分 X11Y11Z10N 法線、4 成分 I16N 接線 (w はフリックベクトル)、単位ベクトル従法線、および 2 成分 I16N テクスチャ座標。

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

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

    EDGE_GEOM_DECOMPRESS_LOAD_COMMON();

    do
    {
        EDGE_GEOM_DECOMPRESS_LOOP_START();

        EDGE_GEOM_DECOMPRESS_LOOP_FIXED_POINT(
            EDGE_GEOM_ATTRIBUTE_ID_POSITION);
        EDGE_GEOM_DECOMPRESS_LOOP_X11Y11Z10N(
```

```
        EDGE_GEOM_ATTRIBUTE_ID_NORMAL);  
    EDGE_GEOM_DECOMPRESS_LOOP_I16N(EDGE_GEOM_ATTRIBUTE_ID_TANGENT);  
    EDGE_GEOM_DECOMPRESS_LOOP_UNIT_VECTOR(  
        EDGE_GEOM_ATTRIBUTE_ID_BINORMAL);  
    EDGE_GEOM_DECOMPRESS_LOOP_F16(EDGE_GEOM_ATTRIBUTE_ID_UV0);  
  
    EDGE_GEOM_DECOMPRESS_LOOP_END();  
}  
while(!EDGE_GEOM_DECOMPRESS_LOOP_DONE());  
  
    EDGE_GEOM_DECOMPRESS_FINALIZE_FIXED_POINT(EDGE_GEOM_ATTRIBUTE_ID_POSITION);  
    EDGE_GEOM_DECOMPRESS_FINALIZE_X11Y11Z10N(EDGE_GEOM_ATTRIBUTE_ID_NORMAL);  
    EDGE_GEOM_DECOMPRESS_FINALIZE_I16N(EDGE_GEOM_ATTRIBUTE_ID_TANGENT);  
    EDGE_GEOM_DECOMPRESS_FINALIZE_UNIT_VECTOR(EDGE_GEOM_ATTRIBUTE_ID_BINORMAL);  
    EDGE_GEOM_DECOMPRESS_FINALIZE_F16(EDGE_GEOM_ATTRIBUTE_ID_UV0);  
  
    break;  
}
```

## 関 連 項 目

[EdgeGeomSpuConfigInfo](#)

---

# edgeGeomDecompressIndexes

---

入力インデックスデータストリームを伸張します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>indexes</i>	入力インデックスデータストリームへのポインタ

## 返 り 値

---

なし

## 解 説

---

この関数は、まず、[edgeGeomInitialize\(\)](#)関数の引数として指定された、コンテキストの [EdgeGeomSpuConfigInfo](#)から、*indexesFlavorAndSkinningFlavor*フォーマットを抽出します。そのあと、この関数は、入力インデックスデータを伸張して、伸張されたデータを元の入力インデックスバッファの上に上書きします。この新しいインデックスバッファのサイズは、元のインデックスバッファ以上になるので、この関数は、呼び出し時のフリーポインタが、入力インデックスバッファの最後を指していることを前提とします。そして、この関数の呼び出し後、 freespaceポインタは、伸張されたインデックスバッファの最後に設定されます。

## 関 連 項 目

---

[EdgeGeomSpuConfigInfo](#)

---

# edgeGeomProcessBlendShapes

---

形状ブレンド操作を実行します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>numShapes</i>	ブレンドする形状の数
<i>shapeInfosEa</i>	形状情報データ ( <a href="#">EdgeGeomBlendShapeInfo()</a> ) の実効アドレス (EA)

## 返 り 値

---

なし

## 解 説

---

この関数は、頂点ユニフォームテーブルのデータに対して形状ブレンド処理を実行します。

この関数は、形状ごとに、以下の処理を行います。

- [EdgeGeomBlendShapeInfo\(\)](#) 中の *shapeInfoEa* からフリーポインタにDMA転送を行います
- [EdgeGeomBlendShapeInfo](#).*dmaTag* の中のリストDMAをバリア付きで実行する
- 上の DMA が完了するのを待ちます
- [edgeGeomDecompressVertexes\(\)](#) を呼び出して、形状フレーバーに応じてデータを伸張り、[EdgeGeomBlendShapeInfo](#).*alpha* を使ってユニフォームテーブルとブレンドします。
- 次の形状を処理するために *shapeInfoEa* をインクリメントします

## 関 連 項 目

---

[EdgeGeomBlendShapeInfo](#)、[edgeGeomDecompressVertexes](#)

---

# edgeGeomNormalizeUniformTable

---

指定されたユニフォームテーブルを正規化します。

## 定 義

---

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

## 引 数

---

<i>pUniform</i>	正規化対象のユニフォームテーブル
<i>vertexCount</i>	頂点の数

## 返 り 値

---

なし

## 解 説

---

この関数は、指定されたユニフォームテーブルの各クワッドワードの X、Y、Z 成分によって作られるベクトルを、各頂点の w 値を維持することに特に留意しながら、正規化します。

---

# edgeGeomSkinVertexes

---

スキニング操作を実行します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>matrices</i>	スキン行列データ (4x3 float) へのポインタ
<i>matrixCount</i>	行列の数
<i>indexesAndWeights</i>	入力インデックスおよび重みデータのストリーム

## 返 り 値

---

なし

## 解 説

---

この関数は、[EdgeGeomSpuConfigInfo\(\)](#) で定義された *indexesFlavorAndSkinningFlavor* を使って、ユニフォームテーブルをスキニングします。

この関数は、実際に頂点をスキニングする前に、入力スキン行列に対して、内部的に特殊な変換を行います。*matrixCount* 引数を使う理由はこのためで、ユーザは、この関数を呼び出した後でスキン行列データを使いたい場合には、この内部変換について意識する必要があります。

この関数の終了時に、フリーポインタは、スキン行列 (*matrices*) の先頭に設定されます。

---

# edgeGeomTransformVertexes

---

指定されたユニフォームテーブルを変換します。

## 定 義

---

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

## 引 数

---

<i>numVertexes</i>	頂点の数
<i>inVertexes</i>	入力ユニフォームテーブル
<i>outVertexes</i>	変換後のユニフォームテーブルの出力先
<i>matrix</i>	<i>inVertexes</i> の変換に使われる 4x4 行列

## 返 り 値

---

なし

## 解 説

---

この関数は、指定されたユニフォームテーブルに、渡された 4x4 行列をかけます。

## 注 意

---

この関数は、「入出力データが同じバッファ上で」実行しても安全です。たとえば、*inVertexes* と *outVertexes* を同じポインタにすることは有効です。

# edgeGeomCullOccludedTriangles

オクルーダーによって隠される三角形をカリングします。

## 定 義

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

## 引 数

<i>ctx</i>	コンテキストへのポインタ。
<i>occludersCount</i>	オクルーダーの数 (8 以下)
<i>occludersEA</i>	オクルーダーのデータが位置するメインメモリ中の実効アドレス
<i>indexBias</i>	インデックスがゼロベースになるように調整します。
<i>detailedResults</i>	(オプション) プロファイリングのために、各カリングテストに失敗した三角形の数を追跡します。このパラメータは、使用するのが C 実装のカリング関数でないか、もしくは、EDGE_GEOM_DEBUG が定義されていない場合には、無視されます。

## 返 り 値

可視インデックスの数を返します。

## 解 説

*occludersCount* がゼロ以外の場合、この関数は以下の処理を実行します。

- オクルーダーをローカルメモリに DMA 転送します
- 各オクルーダーおよび各頂点の座標をスクリーン座標に変換します
- 各頂点をチェックして、オクルーダーによって隠されているかを判定します
- その頂点がスクリーン空間中で 1 つ以上のオクルーダーの背後にある三角形をカリングします
- 可視三角形インデックスの数を返します。

各オクルーダーは、4 個の頂点を表す 16 個の float によって定義されます。各オクルーダーのデータは、16 個の float の配列です。この関数は、最高で 8 個のオクルーダーをサポートします。

*indexBias* は、最初のインデックス値に加算した結果が 0 になるように設定する必要があります。インデックスバイアスは、より大きなジオメトリのサブセット（したがって、インデックスがゼロから始まらない可能性のある）としてロードされたインデックスや頂点データ用に提供されます。

この関数のデバッグビルドバージョンは、EdgeGeomCullingResults 構造体が渡された場合、結果の構造体のオクルージョンおよびカリングメンバを、遮蔽された三角形ごとにインクリメントします。



## **注 意**

---

この関数は、[edgeGeomCullTriangles\(\)](#)の前に呼び出しておく必要があります。

# edgeGeomCullTriangles

カリングフレーバーで指定したメソッドでトライアングルカリングします。

## 定 義

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

## 引 数

<i>ctx</i>	コンテキストへのポインタ
<i>cullingFlavor</i>	カリング操作のフレーバー。 「 <a href="#">トライアングルカリングモード</a> 」を参照。
<i>indexBias</i>	インデックスがゼロベースになるように調整します。
<i>detailedResults</i>	(オプション) プロファイリングのために、各カリングテストに失敗した三角形の数を追跡します。このパラメータは、使用するのがC実装のカリング関数でないか、もしくは、EDGE_GEOM_DEBUG が定義されていない場合には、無視されます。

## 返 り 値

可視インデックスの数を返します。

## 解 説

この関数は、カリングフレーバーが EDGE\_GEOM\_CULL\_NONE に設定されていない場合、以下の処理を行います。

- [EdgeGeomViewportInfo.viewProjectionMatrix](#)に [EdgeGeomLocalToWorldMatrix](#)をかけます。
- 位置を投影空間に変換して得られた x、y 座標を w 座標で割ります。
- ビューポートのスケールとオフセットを適用して、位置をクリップ空間に変換します。
- カリングテスト用の事前計算を実行して、その結果を z、w 成分に格納します。
- 変換後の位置 x、y、およびカリングテスト値 z、w を、変換ユニフォームテーブルに保存します。
- 三角形を指定されたカリングフレーバーでカリングし、出力インデックスを現在の出力ポインタに書き込みます。
- フリーポインタを、I/O バッファ先頭+可視インデックスの数×2 を、次の 16 バイトの倍数まで切り上げた位置に設定します。
- 可視三角形インデックスの数を返します。

ただし、カリングフレーバーが EDGE\_GEOM\_CULL\_NONE に設定された場合には、この関数は EdgeGeomSpuConfigInfo 構造体の中に指定されたインデックスの数を返します。また、[EdgeGeomSpuConfigInfo\(\)](#) の *indexesOffset* が -1 に設定されなかった場合、この関数はフリーポインタをインデックスデータの先頭に設定します。*indexesOffset* が -1 である場合には、フリースペースポインタは変更されません。

*indexBias* は、最初のインデックス値に加算した結果が 0 になるように設定する必要があります。インデックスバイアスは、より大きなジオメトリのサブセット（したがって、インデックスがゼロから始まらない可能性のある）としてロードされたインデックスや頂点データ用に提供されます。

この関数のデバッグビルドバージョンは、EdgeGeomCullingResults 構造体が渡された場合、結果の構造体のメンバを、テストされた三角形ごとにインクリメントします。

---

# edgeGeomCalculateDefaultOutputSize

---

標準の操作で使われる出力サイズを計算します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ。
<i>numIndexes</i>	目に見えるインデックスの数。この値は通常、 <a href="#">edgeGeomCullOccludedTriangles()</a> または <a href="#">edgeGeomCullTriangles()</a> によって返される値です。

## 返 り 値

---

[edgeGeomAllocateOutputSpace\(\)](#) に渡す標準割り当てサイズを返します。

## 解 説

---

標準の操作で使われる出力サイズを計算します。

## 関 連 項 目

---

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

# edgeGeomAllocateOutputSpace

出力領域をメインもしくはローカルメモリから割り当てます。

## 定 義

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

## 引 数

<i>ctx</i>	コンテキストへのポインタ。
<i>outputBufferInfoEa</i>	<a href="#">EdgeGeomOutputBufferInfo</a> 構造体の実効アドレス。
<i>allocSize</i>	バッファ情報から割り当てられるサイズ（バイト）。
<i>outInfo</i>	成功した場合の割り当て情報が書き込まれます。
<i>spuId</i>	SPU の ID 通常は、 <code>cellSpursGetCurrentSpuId()</code> から取得されます。「libspurs コアリファレンス」に記述されています。

## 返 り 値

割り当てに成功した場合には、true を返します。  
それ以外の場合には、false を返します。

## 解 説

この関数は、メインもしくはローカルメモリに、ライブラリがインデックス、頂点、コマンドバッファなどを出力するために利用できる空き領域を割り当てを試みます。

(*outputBufferInfoEa* & 0x2) がセットされた場合、この関数は *outputBufferInfoEa* & 0xFFFFFFFFFC に等しいアドレスを持っている静的出力バッファがあると仮定します。

*outputBufferInfoEa* は、EDGE\_GEOM\_DIRECT\_OUTPUT\_TO\_MAIN\_MEMORY または EDGE\_GEOM\_DIRECT\_OUTPUT\_TO\_LOCAL\_MEMORY を用いてマスクされなければなりません。

でないと、この関数は [EdgeGeomOutputBufferInfo](#) の設定に応じて、共有バッファもしくはリングバッファから動的に領域を割り当てを試みます。

[EdgeGeomOutputBufferInfo](#) のセクションで説明したように、*startEa* フィールドと *endEa* フィールドが同じである場合には、バッファは空もしくは存在しないものと見なされます。

シングルのバッファ出力方式が使われるのは、共有バッファだけが空でない場合だけです。

リングバッファ出力方式が使われるのは、リングバッファだけが空でない場合だけです。

最終的に、どちらのバッファも空でない場合には、リングバッファ出力方式が使われます。その際に、SPU が RSX® を待たなければならなくなるときには、オーバーフロー領域として共有バッファも使われます。リングバッファ方式が使われた場合には、この関数は、RSX® が特定のメモリを消費するのを関数内で待つので、割り当てに成功するまでリターンしません。

*spuId* には、どのリングバッファ（特に、現在の SPU と対応するもの）から割り当てを指定します。

出力バッファ方式に関する詳しい情報については、「*PlayStation®Edge* ライブラリ概要」を参照してください。

## **注 意**

---

割り当てに失敗すると、ライブラリ内のすべての残りの処理も中断されます。これはグラフィックエラーを引き起こし、RSX®をクラッシュさせる場合があります。

---

# edgeGeomUseOutputSpace

---

割り当てられた領域を使って、対応する EdgeGeomAllocationInfo データを更新します。

## 定 義

---

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

## 引 数

---

<i>info</i>	使用後に更新する構造体
<i>size</i>	使用する出力領域の量
<i>ea</i>	使用する領域の先頭の実効アドレス
<i>offset</i>	使用する領域の先頭の RSX® オフセット

## 返 り 値

---

なし。

## 解 説

---

この関数は、割り当てられた領域が複数の独立した出力データのために使われる場合に、その領域を管理するために提供されます。同じリングバッファから複数の領域を割り当てるとデッドロックの原因になるので、代わりに、単一の領域とこの関数を使うようにしてください。この関数は、割り当てられた領域の中の残り領域の実効アドレスの先頭、および RSX® オフセットを更新します。

---

# edgeGeomOutputIndexes

---

IO バッファのインデックスをメインもしくはローカルメモリに出力します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>numIndexes</i>	目に見えるインデックスの数。この値は通常、 <a href="#">edgeGeomCullOccludedTriangles()</a> 、 <a href="#">edgeGeomCullTriangles()</a> によって返される値です。
<i>info</i>	頂点を配置する場所を示す割り当て情報
<i>outLoc</i>	インデックスの位置

## 返 り 値

---

なし

## 解 説

---

この関数は、インデックスを出力データアドレスに DMA 転送してから、出力データアドレスに、可視インデックスの数×2 を、次の 16 バイトの倍数まで切り上げた数を加えます。

この関数を呼び出した後、*info* の実効アドレスとオフセットは、それぞれ適切な数だけインクリメントされます。したがって、大きな領域をまとめて割り当てて、なおかつ、その中にさまざまな型のデータを持たせることができます。

## 注 意

---

ユーザがコマンドバッファを生成して、コマンドバッファホール型の同期を利用する際には、この関数は、[edgeGeomBeginCommandBufferHole\(\)](#) を呼び出す前に呼び出す必要があります。

## 関 連 項 目

---

[edgeGeomCullTriangles](#)、[edgeGeomCullOccludedTriangles](#)



# edgeGeomCompressVertexes

ユニフォームテーブルの中のデータを、[EdgeGeomSpuConfigInfo](#)で指定された出力頂点フォーマットに圧縮します。

## 定 義

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

## 引 数

ctx                      コンテキストへのポインタ

## 返 り 値

圧縮頂点データへのポインタを返します。

## 解 説

この関数は、ユニフォームテーブルを出力頂点フォーマットに圧縮して、出力頂点を出力ポインタに書き込みます。圧縮が済むと、この関数は、フリーポインタに、出力フォーマットのストライドサイズ×頂点の数を、次の 128 バイトの倍数まで切り上げた数を加算します。

## 注 意

静的にリンクされた新しい出力頂点フォーマットへのサポートを追加するには、[edgeGeomCompressVertexes\(\)](#)の中のスイッチ文に新しいcaseブロックを加えます。このcaseブロック内で使われているマクロ（edgegeom\_compress.hで定義された）は、ほぼ最適な圧縮コードの明確かつ簡潔な実装を可能にします。

**注意：** EDGE\_GEOM\_COMPRESS マクロは大いにコンパイル時定数に依存しているので、最適化コンパイラは可能な限り事前計算を行うことができます。したがって、このマクロの引数がすべてコンパイル時定数（数字定数もしくは#define 定数）になっている必要があります。実際には、コンパイル時にパラメータ値はわかっているはずなので、このことは問題にはならないはずです。

case ブロックの中で呼び出す必要がある最初のマクロは、EDGE\_GEOM\_COMPRESS\_INIT\_GLOBAL です。このマクロの唯一のパラメータは、頂点ストライドの合計です。（バイト）

次に、圧縮する属性を列挙します。圧縮マクロでサポートしている属性の種類は、以下の通りです。

属性種別	解説
F32	32 ビット浮動小数点
F16	16 ビット浮動小数点
U8	8 ビット符号付き整数
I16	16 ビット符号付き整数
U8N	8 ビット符号なし整数。[0.0...1.0]の範囲を表すように正規化
I16N	16 ビット符号付き整数。[-1.0...1.0]の範囲を表すように正規化
X11Y11Z10N	32 ビットのパックされた 3 成分ベクトル。 基本的に、X 用の I11N、Y 用の I11N、Z 用の I10N が 1 つの 32 ビット値にパックされます。

各属性について、下記の一覧の中から対応するマクロを、属性が頂点の中に出現する順序で呼び出します。

- `EDGE_GEOM_COMPRESS_INIT_F32(attrId, attrOffset, attrSize)`
- `EDGE_GEOM_COMPRESS_INIT_F16(attrId, attrOffset, attrSize)`
- `EDGE_GEOM_COMPRESS_INIT_U8(attrId, attrOffset, attrSize)`
- `EDGE_GEOM_COMPRESS_INIT_I16(attrId, attrOffset, attrSize)`
- `EDGE_GEOM_COMPRESS_INIT_U8N(attrId, attrOffset, attrSize)`
- `EDGE_GEOM_COMPRESS_INIT_I16N(attrId, attrOffset, attrSize)`
- `EDGE_GEOM_COMPRESS_INIT_X11Y11Z10N(attrId, attrOffset, attrSize)`

上記のマクロは、以下のようなパラメータをとります。

<code>attrId</code>	<code>EDGE_GEOM_ATTRIBUTE_ID_*</code> 列挙型の属性を一意に特定する数値
<code>attrOffset</code>	属性の頂点中でのバイトオフセット
<code>attrSize</code>	属性のサイズ (バイト)

次に、ループ条件を `!EDGE_GEOM_COMPRESS_LOOP_DONE()` とする `do-while` ループを挿入します。

ループの中では、まず、`EDGE_GEOM_COMPRESS_LOOP_START()` を呼び出して、ループ本体の準備を行います。

次に、各属性を圧縮します。各属性について、下記の一覧の中から対応するマクロを呼び出します(`attrId` パラメータの意味は、前と同じです)。

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

`EDGE_GEOM_COMPRESS_LOOP_END()` を呼び出してループを終了します。

以下は、頂点ストライドが 32 バイトで、5 つの属性を含む出力ストリーム全体の例です。3 成分の F32 位置、3 成分の X11Y11Z10N 法線、4 成分の I16N 接線、(w はフリップベクトル)、3 成分の X11Y11Z10N 従法線、2 成分の I16N テクスチャ座標。

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

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

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

        EDGE_GEOM_COMPRESS_LOOP_END();
    }
}
```

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

---

# edgeGeomOutputVertexes

---

IO バッファの圧縮頂点データをメインもしくはローカルメモリに出力します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>info</i>	頂点を配置する場所を示す割り当て情報
<i>outLoc</i>	ここにインデックスの位置が格納されます

## 返 り 値

---

なし

## 解 説

---

この関数は、出力頂点を出力データアドレスに DMA 転送してから、出力データアドレスに、出力フォーマットプレーバーによって指定された出力頂点ストライド×頂点の数を、次の 128 バイトの倍数まで切り上げた数を加算します。

この関数が呼び出された後、*info* の実効アドレスとオフセットは、それぞれ適切な数だけインクリメントされます。したがって、大きな領域をまとめて割り当てて、なおかつ、その中にさまざまな型のデータを持たせることができます。

## 注 意

---

ユーザがコマンドバッファを生成して、コマンドバッファホール型の同期を利用する際には、この関数は、[edgeGeomBeginCommandBufferHole\(\)](#) を呼び出す前に呼び出す必要があります。

## 関 連 項 目

---

[edgeGeomBeginCommandBufferHole](#)

---

# edgeGeomBeginCommandBufferHole

---

メインメモリ中にコマンドバッファを生成して、コマンドバッファホールに書き込みを行います。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>gcmCtx</i>	この関数によって設定されるコンテキストデータです。RSX®コマンドを書き込むため必要な情報が含まれています。 詳細は「libgcm リファレンス」を参照のこと。
<i>holeEa</i>	書き込む必要のあるコマンドバッファの実効アドレス

## 返 り 値

---

なし

## 解 説

---

コマンドバッファホールに書き込みを行うためのコマンドバッファの生成の開始をします。以降のコマンドは、メインメモリ常駐のコマンドバッファに合わせてアラインメントされた、*gcmCtx* コンテキストの中で実行されます。

## 関 連 項 目

---

[EdgeGeomSpuConfigInfo](#)

---

# edgeGeomSetVertexDataArrays

---

RSX®用の頂点データフォーマットとオフセットを設定するコマンドを挿入します。

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ。
<i>gcmCtx</i>	コマンドが配置されるコンテキスト。 詳細は「libgcm リファレンス」を参照のこと。
<i>vtxLoc</i>	コマンドバッファホルの書き込みが正しく行えるように、頂点の場所を指定します。

## 返 り 値

---

なし

## 解 説

---

指定されたコマンドコンテキストにRSX®用の頂点データフォーマットとオフセットを設定するコマンドを挿入します。[EdgeGeomSpuConfigInfo](#)が示す出力頂点フォーマットが定義済みのフォーマットではなく、なおかつ、[EdgeGeomSetVertexDataArraysCallback\(\)](#)が[EdgeGeomVertexStreamDescription](#)構造体に提供されている場合には、このコールバック関数で頂点データコマンドを書き込む必要があります。

## 関 連 項 目

---

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

# edgeGeomEndCommandBufferHole

コマンドバッファホールを埋め、その結果を DMA を使ってメインメモリ中の対応するバッファに書き込みます。

## 定 義

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

## 引 数

<i>ctx</i>	コンテキストへのポインタ
<i>gcmCtx</i>	コマンドバッファホールに書き込まれるコマンドが格納されたコンテキスト
<i>holeEa</i>	書き込む必要のあるコマンドバッファの実効アドレス
<i>infos</i>	割り当て情報の配列。 配列中の割り当ては、それぞれ、独立したエントリとして指定する必要があります。 リングバッファリングを使わない場合には、NULL を指定してかまいません。
<i>numInfos</i>	<i>infos</i> 配列の要素数。 リングバッファリングを使わない場合には、ゼロを指定してかまいません。

## 返 り 値

なし

## 解 説

SPUローカルストア中のコマンドバッファホールを、未使用領域にSkip NOPを書き込んで埋めます。書き込み後のコマンドバッファは、メインメモリ上の対応する領域にDMA転送されます。また、リングバッファが使われている場合には、この関数は、出力データが使われた後、[EdgeGeomAllocationInfo](#)配列の各要素に対応するRSX®ラベルを更新するためのコマンドを書き込みます。

## 関 連 項 目

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

# コールバック関数



---

# EdgeGeomGetOutputVertexStrideCallback

---

カスタム出力フォーマットフレーバーのストライドサイズを提供するコールバック関数

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>outputFormatId</i>	カスタム出力フォーマット ID

## 返 り 値

---

指定された RSX®頂点フォーマットのストライドサイズ

## 解 説

---

[EdgeGeomGetOutputVertexStrideCallback\(\)](#) は、[edgeGeomGetOutputVertexStride\(\)](#) 関数がカスタム出力フォーマットを検出するたびに呼び出されます。

この関数ポインタを設定するには、[EdgeGeomVertexStreamDescription](#) 構造体を使います。

## 関 連 項 目

---

[edgeGeomGetOutputVertexStride](#)

---

# EdgeGeomDecompressVertexStreamCallback

---

頂点データストリームを指定されたカスタム入力頂点フォーマットで伸張するコールバック関数

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>vertexes</i>	伸張する入力頂点データストリームへのポインタ
<i>numVertexes</i>	頂点の数
<i>fixedOffsets</i>	2 番目の頂点ストリームを伸長する際に使われる、固定小数点オフセットテーブルへのポインタ。ストリーム中に固定小数点属性が存在しない場合には、NULL でもかまいません
<i>inputFormatId</i>	ストリームの頂点フォーマット ID

## 返 り 値

---

使用されていません。

## 解 説

---

このコールバック関数は、入力頂点データストリームがカスタム入力頂点フォーマットを使って圧縮される際に、[edgeGeomDecompressVertexes\(\)](#) 関数から呼び出されます。

この関数ポインタを設定するには、[EdgeGeomVertexStreamDescription](#) 構造体を使います。

## 関 連 項 目

---

[edgeGeomDecompressVertexes](#)

---

# EdgeGeomBlendVertexStreamCallback

---

カスタム頂点デルタフォーマットに対して形状ブレンド操作を実行するコールバック関数

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>vertexesB</i>	(先に伸張された頂点データとブレンドするための) 入力頂点デルタストリームへのポインタ
<i>numVertexes</i>	頂点の数
<i>fixedOffsets</i>	データ伸長に使われる固定オフセットバッファへのポインタ
<i>deltaFormatId</i>	カスタム頂点デルタフォーマット ID
<i>alpha</i>	この形状ブレンドのブレンド比率

## 返 り 値

---

使用されていません。

## 解 説

---

このコールバック関数は、カスタムの形状ブレンド操作を実行するために、[edgeGeomProcessBlendShapes\(\)](#) 関数からカスタム頂点デルタフォーマットに対して呼び出されます。結果情報は [edgeGeomGetUniformTable](#) を呼ぶことによって得られたユニフォームテーブルに書き込まなければなりません。

この関数ポインタを設定するには、[EdgeGeomVertexStreamDescription](#) 構造体を使います。

## 関 連 項 目

---

[edgeGeomProcessBlendShapes](#)

---

# EdgeGeomCompressVertexStreamCallback

---

頂点データをカスタム出力フォーマットに圧縮するコールバック関数

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>numVertexes</i>	頂点の数
<i>outVertexes</i>	圧縮データを出力する I/O バッファの上の空き領域
<i>flavor</i>	カスタム出力頂点フォーマット ID

## 返 り 値

---

使用されていません。

## 解 説

---

このコールバック関数は、頂点ユニフォームテーブルの中の圧縮データを、カスタム出力フォーマットに圧縮するために、[edgeGeomCompressVertexes\(\)](#)から呼び出されます。

この関数ポインタを設定するには、[EdgeGeomVertexStreamDescription](#)構造体を使います。

## 関 連 項 目

---

[edgeGeomCompressVertexes](#)、[EdgeGeomVertexStreamDescription](#)

# EdgeGeomSetVertexDataArraysCallback

カスタム出力フォーマット用のコマンドバッファを生成するコールバック関数

## 定 義

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

## 引 数

<code>ctx</code>	コンテキストへのポインタ
<code>outputFormatId</code>	カスタム頂点出力フォーマット ID
<code>gcmCtx</code>	コマンドが配置されるコンテキスト。 詳細は「libgcm リファレンス」を参照
<code>outputLocation</code>	最終的な頂点データの場所。
<code>vertexOffset</code>	CELL_GCM_LOCATION_MAIN、もしくは、CELL_GCM_LOCATION_LOCAL メインもしくはローカルメモリ上の頂点データの開始アドレスの I0 オフセット。

## 返 り 値

使用されていません。

## 解 説

このコールバック関数は、カスタム出力頂点データフォーマット用のコマンドバッファを生成するために、[edgeGeomSetVertexDataArrays\(\)](#) 関数から呼び出されます。

この処理は、`cellGcmSetVertexDataArray` を属性ごとに呼び出すことによって実現できます。

この関数ポインタを設定するには、[EdgeGeomVertexStreamDescription](#) 構造体を使います。

## 注 意

この関数によって出力されたコマンドバッファは、コマンドバッファホールに書き込むために、メインメモリに DMA 転送されます。したがって、コマンドバッファの生成方法には、細心の注意を払ってください。コマンドバッファホールの詳細については、「*PlayStation®Edge ライブラリ概要*」を参照してください。

## 関 連 項 目

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

---

# EdgeGeomTransformVertexesForCullCallback

---

三角形カリングの前に頂点位置を変換するためのコールバック関数

## 定 義

---

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

## 引 数

---

<i>ctx</i>	コンテキストへのポインタ
<i>userData</i>	ユーザデータへのポインタ

## 返 り 値

---

なし。

## 解 説

---

そう指定された場合、[EdgeGeomTransformVertexesForCullCallback\(\)](#)がデフォルトの頂点変換関数 [edgeGeomTransformVertexes\(\)](#) の代わりに呼び出されます。

このカリング関数は、変換後の頂点が以下の形式になると想定します。

- x 成分 - 変換後の x 座標
- y 成分 - 変換後の y 座標
- z 成分 - 頂点フラスタムテストの結果。ビット 31-26: z 最小値未満、x 最小値未満、y 最小値未満、x 最大値より大きい、y 最大値より大きい、z 最大値より大きい。
- w 成分 - ビット 32: 変換された w が正の場合にセットされます。ビット 30-16: 量子化された x。ビット 15-0: 量子化された y。

## 定数

# トライアングルカリングモード

実行するトライアングルカリングテストのさまざまな組合せを表す定数

## 定 義

マクロ	値	解説
EDGE_GEOM_CULL_NONE	0	トライアングルカリングを実行しません。
EDGE_GEOM_CULL_FRUSTUM	1	フラスタムおよびピクセル中心トライアングルカリングを実行します。
EDGE_GEOM_CULL_BACKFACES_AND_FRUSTUM	2	裏面、フラスタム、およびピクセル中心トライアングルカリングを実行します。
EDGE_GEOM_CULL_FRONTFACES_AND_FRUSTUM	3	表面、フラスタム、およびピクセル中心トライアングルカリングを実行します。

## 解 説

この定数は `cullingFlavor` 引数の [edgeGeomCullTriangles\(\)](#) に渡されます。



# スキニングモード

実行するさまざまな種類の行列パレットスキニングを表す定数

## 定 義

マクロ	値	解説
EDGE_GEOM_SKIN_NONE	0	スキニングを実行しません。
EDGE_GEOM_SKIN_NO_SCALING	1	単位行列スキニング。
EDGE_GEOM_SKIN_UNIFORM_SCALING	2	スキニングを実行します。
EDGE_GEOM_SKIN_NON_UNIFORM_SCALING	3	スキニングを実行して、正規変換のための余因子行列を計算します。
EDGE_GEOM_SKIN_SINGLE_BONE_NO_SCALING	4	シングルボーン単位行列スキニング。
EDGE_GEOM_SKIN_SINGLE_BONE_UNIFORM_SCALING	5	シングルボーンスキニング。
EDGE_GEOM_SKIN_SINGLE_BONE_NON_UNIFORM_SCALING	6	シングルボーンスキニング。正規変換のための余因子行列を計算します。

## 解 説

[EdgeGeomSpuConfigInfo](#)の中の*indexesFlavorAndSkinningFlavor*によって使われます。

# インデックスデータの種類

セグメントで使われるさまざまなインデックスデータの種類を表す定数

## 定 義

マクロ	値	解説
EDGE_GEOM_INDEXES_U16_TRIANGLE_LIST_CW	0	時計回りの三角形順序をもつ 16 ビットのインデックスデータ。
EDGE_GEOM_INDEXES_U16_TRIANGLE_LIST_CCW	1	反時計回りの三角形順序をもつ 16 ビットのインデックスデータ。
EDGE_GEOM_INDEXES_SW_TRIANGLE_LIST_CW	2	時計回りの三角形順序をもつソフトウェア圧縮されたインデックスデータ。
EDGE_GEOM_INDEXES_SW_TRIANGLE_LIST_CCW	3	反時計回りの三角形順序をもつソフトウェア圧縮されたインデックスデータ。

## 解 説

[EdgeGeomSpuConfigInfo](#)の中の*indexesFlavorAndSkinningFlavor*によって使われます。

# スキニング行列フォーマット

サポートされたスキニング行列フォーマットを表す定数

## 定 義

マクロ	値	解説
EDGE_GEOM_MATRIX_3x4_ROW_MAJOR	0	これは Edge のネイティブの行列型で、メモリの使用量に関して最も効率的です。この行列は、「DirectX スタイル」の 4x4 行列の上 3 行です。4 行目は常に [0, 0, 0, 1] なので、明示的に格納する必要はありません。
EDGE_GEOM_MATRIX_4x4_ROW_MAJOR	1	「DirectX スタイル」の行優先形式の 4x4 行列を指定します。この型が提供されるのは、主に便宜上の理由からです。4x4 行列は、3x4 行列より 33% 多くのメモリを使用することは言うまでもありません。
EDGE_GEOM_MATRIX_4x4_COLUMN_MAJOR	2	「OpenGL スタイル」の列優先形式の 4x4 行列を指定します。この型が提供されるのは、主に便宜上の理由からです。4x4 行列は、3x4 行列より 33% 多くのメモリを使用することは言うまでもありません。

## 解 説

[EdgeGeomSpuConfigInfo](#)の中の *skinningMatrixFormat* によって使われます。

# ジオメトリ設定フラグ

ジオメトリパイプラインのさまざまな設定を表す定数。このフラグは、[EdgeGeomSpuConfigInfo](#)の *flagsAndUniformTableCount* メンバに保存されます。

## 定 義

マクロ	値	解説
EDGE_GEOM_FLAG_STATIC_GEOMETRY_FAST_PATH	0x10	この値は、入力ジオメトリがSPU ジョブによって変更されていないことを示します。これにより、一部の Edge 関数は、不要な処理を実行せずに直ちにリターンすることができます。
EDGE_GEOM_FLAG_INCLUDES_EXTRA_UNIFORM_TABLE	0x80	このフラグは、セグメントがカリングもしくはカスタムブレンド形状フレーバーを使っている場合に、ツールによって設定する必要があります。これらの処理は、どちらも、一時データを格納するための追加の一樣テーブルが必要です。このテーブルのサイズは、Edge パーティショナーの中でセグメントを作成する際に指定する必要があります。

## 解 説

[EdgeGeomSpuConfigInfo](#)中の *flagsAndUniformTableCount* の上位の 4 ビットによって使われます。

## 出力モード

メインメモリもしくはビデオメモリ中での静的な出力バッファの使い方を示す定数。これらの定数は、[edgeGeomAllocateOutputSpace\(\)](#)に渡される出力バッファ情報の実効アドレスに対するマスクとして使う必要があります。

### 定 義

マクロ	値	解説
EDGE_GEOM_DIRECT_OUTPUT_TO_LOCAL_MEMORY	0x2	この定数は、 <a href="#">edgeGeomAllocateOutputSpace()</a> に指定されたビデオメモリ中のアドレスに直接出力するようにEdgeに指示します。
EDGE_GEOM_DIRECT_OUTPUT_TO_MAIN_MEMORY	0x3	この定数は、 <a href="#">edgeGeomAllocateOutputSpace()</a> に指定されたメインメモリ中のアドレスに直接出力するようにEdgeに指示します。

### 解 説

この定数は、[EdgeGeomSpuConfigInfo\(\)](#)の中にあります。

Filename: Edge\_Geometry\_Library-Reference\_j.doc  
Directory: C:\Documents and Settings\MTESHIMA\My Documents\Working  
Files\7-19-10\Edge 1.2.0  
Template: C:\sony\yoshi\templates\libref\_V1.1\_j.dot  
Title: Edge\_Geometry\_Library-Reference\_j.doc  
Subject:  
Author: SCE Document Group  
Keywords:  
Comments:  
Creation Date: 7/19/2010 2:54:00 PM  
Change Number: 3  
Last Saved On: 7/19/2010 2:55:00 PM  
Last Saved By: mteshima  
Total Editing Time: 6 Minutes  
Last Printed On: 7/19/2010 3:02:00 PM  
As of Last Complete Printing  
Number of Pages: 101  
Number of Words: 27,896 (approx.)  
Number of Characters: 57,747 (approx.)