Live2D Cubism Core
API リファレンス

Version r11
最終更新日 2023/03/10

更新履歴

更新日	版数	種別	内容
2018/03/27	r1	新規作成	新規作成。
2018/05/18	r2	修正	誤リンク修正 誤字修正 画像誤字修正
		追加	csmGetDrawableVertecPositionsで手に入る 頂点情報の特性について追記
2018/07/11	r3	修正	脱字修正 snippetの修正 曖昧さの回避による修正 目次のための整形修正 マスクの描画方法とそのアクセスに関して、 描画方法の指定を細かく修正 リンクの追加 繰り返し表記の修正 引数の表記にconstがある間違いを修正
2018/12/20	r4	追加	「moc3ファイルのバージョンについて」を追加 「パーツの親パーツを取得」を追加 csmGetPartParentPartIndicesのAPI説明を追加
		修正	曖昧さ回避のための修正 誤字修正
2019/02/12	r5	追加	「moc3ファイルのバージョンについて」にファイルバージョンを調べる方法について追加 csmGetLatestMocVersionのAPI説明を追加 csmGetMocVersionのAPI説明を追加 csmGetPartParentPartIndicesのAPIが追加されたCoreVersionの表記を追加
2019/08/01	r6	追加	moc3ファイルバージョンの定数を追加 ConstantFlagの要素追加に伴うスニペットの追記 マスクの反転のフラグについて説明を追加 マスク反転機能の説明追加 個別のAPIに利用可能バージョンの項目を追加
		修正	誤字修正
2019/09/04	r7	修正	Cubism Core および Cubism SDKの表記を調整
2021/02/26	r8	修正	csmGetDrawableIndexCountsで個数0のDrawableが存在する説明を追加しました。
		修正	csmGetDrawableIndicesで有効なアドレスが格納さ

			れないケースがある説明を追加しました。
2022/05/19	r9	追加	パラメータのキーを取得する機能の説明を追加しました。
		追加	乗算色、スクリーン色関連の説明を追加しました。
2022/07/07	r10	追加	パラメータの種類を取得する機能の説明を追加しました。
		追加	アートメッシュの親のパーツを取得する機能の説明を追加しました。
		追加	「moc3のファイルバージョン」「csmGetMocVersion」に、取得できるバージョンについての記述を更新しました。
2023/03/10	r11	追加	MOC3を検証する機能の説明を追加しました。

※　直近の修正、追加は黄色のハイライトで示しています。

目次

csmGetDrawableVertexUvs
csmGetDrawableIndexCounts
csmGetDrawableIndices
csmResetDrawableDynamicFlags
csmGetDrawableMultipleColors
csmGetDrawableScreenColors
csmGetDrawableParentPartIndices
csmHasMocConsistency

概要

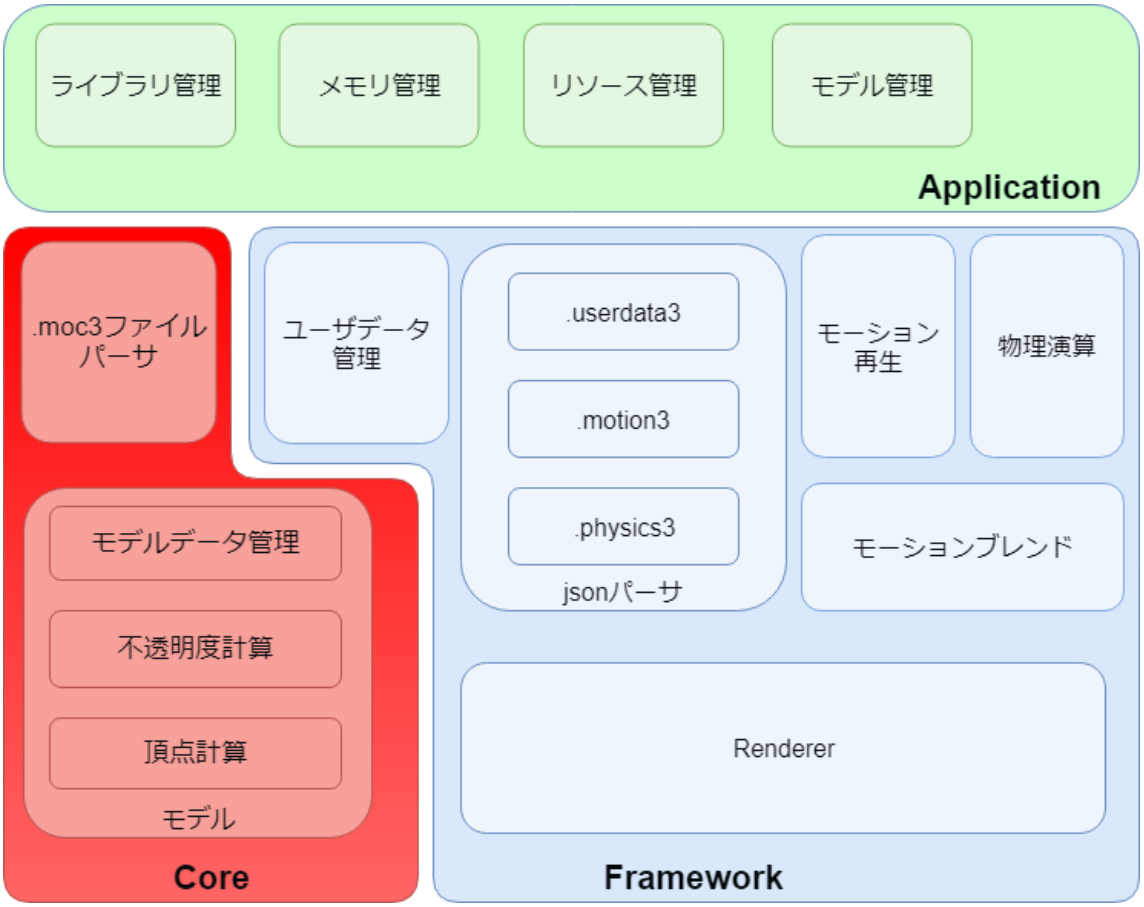
このドキュメントについて

本ドキュメントは Live2D Cubism SDK における Live2D Cubism Core (以降、Coreと呼びます) の役割、使用方法、APIの仕様について扱います。

対象者
・Live2D Cubism SDK の利用者
・Core を他言語(JavaやPhythonなど)から呼び出せるようにラッパーの実装を検討している者
・ゲームエンジンなど他のプログラムやプラットフォームへ組み込むことを検討している者

CoreとFrameworkの機能分類

以下の図はApplication、Core、Frameworkの関係と機能の役割を示したものです。
CoreはApplicationからもFrameworkからも使用されます。



・Coreとは

Coreは Live2D Cubism Editor を用いて作成されたモデル(.moc3ファイル)を扱うために必要なAPIを備えたライブラリです。特徴について下記に述べます。

・APIはC言語から構成。
・Core自身でメモリを確保・破棄しません。必要なメモリ量をCoreから要求されるため、利用する側で指定量を確保してCoreに渡す必要があります。
・描画機能は含みません。モデルのパラメータに応じて頂点情報を計算することがCoreの役目であり、利用する側は計算済みの頂点情報および描画に必要な情報(UVや不透明度など)をCoreから取得し、描画を行います。なお描画機能はFrameworkがリファレンス実装を提供しているため独自に実装する必要はありません。

上記特徴から、移植性の高くプラットフォームに依存しない設計になっています。

モデルの描画

Live2D Cubism 3 SDK 以降の Core は Live2D Cubism 2 SDKまでと違い、描画機能は分離されました。
描画を分離することで様々な環境へ開発者自身が新たにCubismを組み込めるという利点があります。
ユースケースの多い環境に対してはFrameworkで描画機能をリファンレンス実装として提供しておりますが、提供されていない環境であっても、CoreのAPIを利用して必要な頂点情報など3Dプリミティブ情報を取得し、環境に応じた描画APIを利用することで実現可能です。

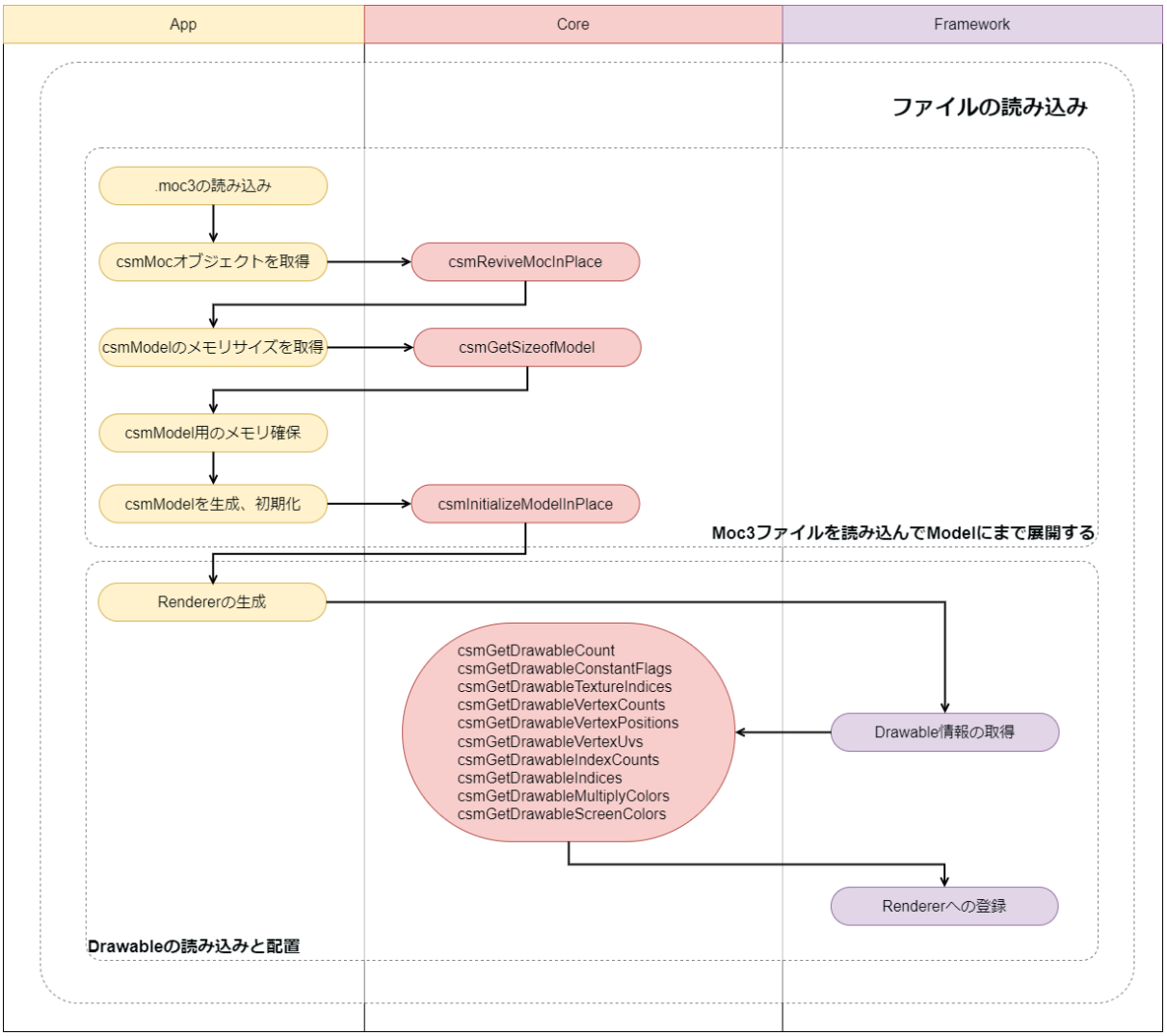
・Coreの提供する描画のためのデータ

Coreがモデルについて提供するデータはParameter、Part、Drawableの大きく3つに分類されます。

その中で描画に必要なデータの集まりがDrawableです。
Drawableが提供する頂点情報はxyの２次元のデータになります。
各要素の座標の指定は左下原点で、ポリゴンの表面は左回りです。
OpenGLの座標系に準じたデータになります。

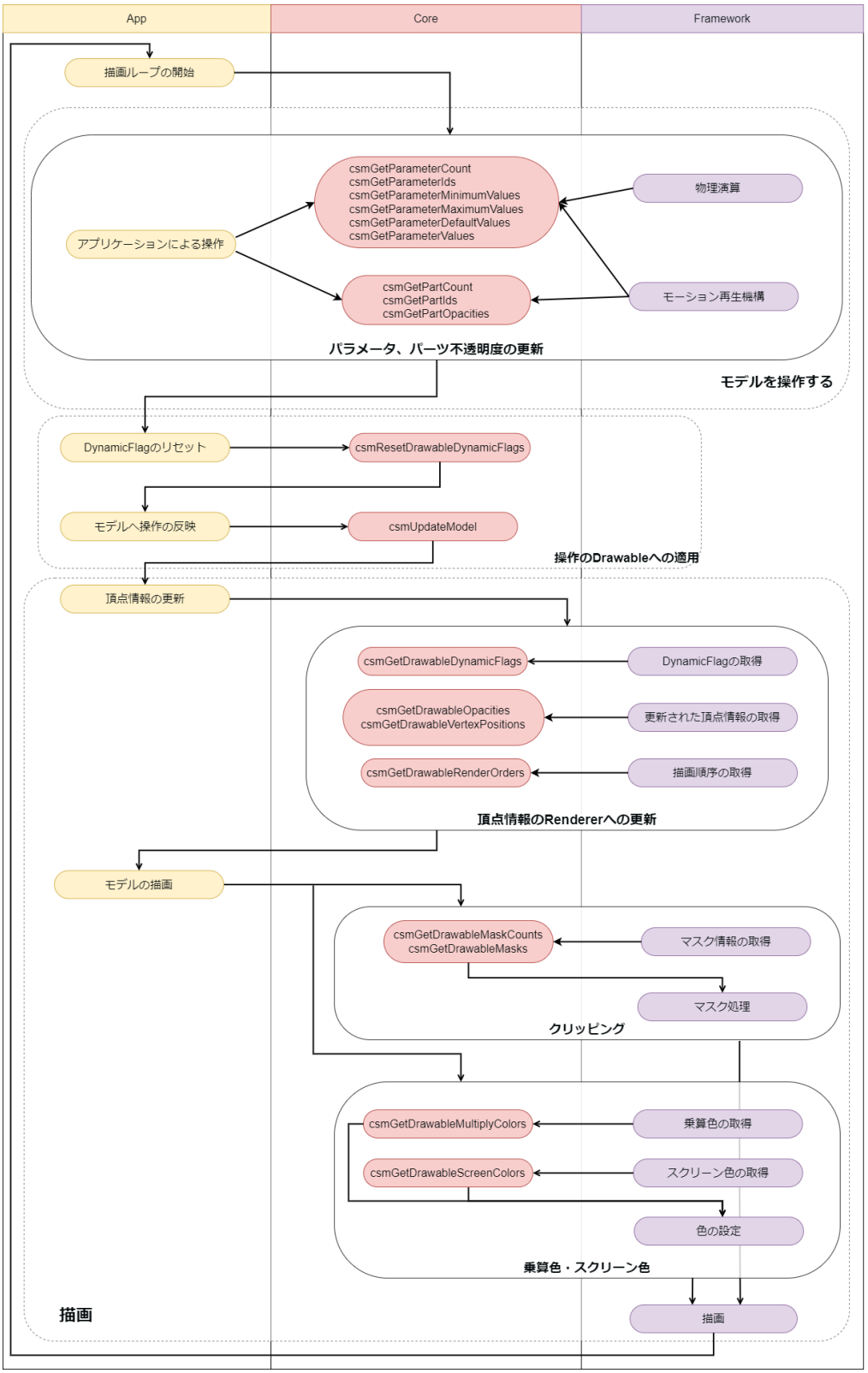
・描画サイクルとCoreの動作

下記の図はモデルファイル（.moc3)の読み込みの時の処理の流れを示したものです。



黄色のノードがApplication、紫のノードがFrameworkで処理すべき部分で
中央のCoreへ矢印が引かれているノードは、CoreのAPIを呼び出しを示しています。

下記の図は描画の更新サイクルを示したものです。



先ほどの図と同じく、黄色のノードがApplication、紫のノードがFrameworkで処理すべき部分で
中央のCoreへ矢印が引かれているノードは、CoreのAPIを呼び出しを示しています。
実線で囲まれた部分は簡易化している部分です。

シーンごとに見るAPIの使い方

Coreに関連する情報を取得する

・Coreのバージョンを取得する

現在利用しているCoreのバージョンを取得することができます。

```
snippet:
    csmVersion version = csmGetVersion();
```

バージョンの表記はMAJOR, MINOR, PATCHの3つから構成されます。
それぞれの運用ルールについて下記に示します。

メジャーバージョン (1byte)
Cubism Editorがメジャーバージョンアップするなどしてモデルデータ(.moc3ファイル)と後方互換性がなくなった時にインクリメントされます。

マイナーバージョン (1byte)
モデルデータの後方互換性を保ちつつ機能追加が追加されたときにインクリメントされます。

パッチ番号 (2byte)
不具合が修正されたときにインクリメントされます。メジャーバージョンあるいはマイナーバージョンが変更された場合、パッチ番号は0にリセットされます。

```
0x    00      00    0000
    Major  Minor  Patch
```

バージョンは4byteから構成され、単に符号なし整数として扱うことで、新しいCoreのバージョンの方が必ず大きい数値を指すことになります。

APIへのリンク

csmGetVersion

・Coreのログを出力する

Coreの内部のログを出力するために、ログ出力関数を登録することができます。
例えば、CoreのAPIを利用した際にエラーが起きた場合、登録されたログ出力関数を通してログが出されます。

登録可能なログ出力関数は下記のシグネチャになります。

```
snippet:
    /** Log handler.
     *
     * @param  message  Null-terminated string message to log.
     */
    typedef void (*csmLogFunction)(const char* message);
```

使用例：

```
snippet:
    void logPrint(const char* message)
    {
       printf(“[LOG] %s”, message);
    }

    // Set Cubism log handler.
    csmSetLogFunction(logPrint);
```

使用APIへのリンク

csmSetLogFunction
csmGetLogFunction

ファイルの読み込み

・moc3ファイルを読み込んでcsmModelオブジェクトにまで展開する

moc3にはモデル情報が格納されていますが、Coreで扱うにはcsmModelオブジェクトへ展開する必要があります。
csmModelに展開した後はcsmModelをキーにAPIを操作をすることになります。
csmMocとcsmModelのオブジェクト生成のためのメモリ領域は先頭アドレスが各指定サイズでアライメントされている必要があります。
アライメントサイズに関してはインクルードに記載されています。

moc3のロード

```
snippet:
/** アライメントサイズの定義 */
enum
{
        /** Necessary alignment for mocs (in bytes). */
        csmAlignofMoc = 64,
        /** Necessary alignment for models (in bytes). */
        csmAlignofModel = 16
};

void* mocMemory;
unsigned int mocSize;

//64byteアライメントされたメモリアドレスにファイルを読み込む。
//mocSizeには.moc3のファイルサイズが格納されます。
mocMemory = ReadBlobAligned("Koharu/Koharu.moc3", csmAlignofMoc, &mocSize);

csmMoc* moc = csmReviveMocInPlace(mocMemory, mocSize);
```

moc3からモデルを作成：

```
snippet:
unsigned int modelSize = csmGetSizeofModel(moc);

// モデルは16byteでアライメントする必要がある
void** modelMemory = AllocateAligned(modelSize, csmAlignofModel);

// モデルのインスタンスを作成
csmModel* model = csmInitializeModelInPlace(moc, modelMemory, modelSize);
```

・moc3の整合性の確認

csmHasMocConsistencyを使用することで、読み込ませる.moc3ファイルが不正なものでないかの整合性を確認することができます。
.moc3ファイルの整合性が確認できない場合は0を返します。

不特定の.moc3ファイルが読み込まれることが想定される場合、csmInitializeModelInPlaceでcsmMocのオブジェクトを生成する前に、読み込ませる.moc3ファイルの整合性の確認を行うことを推奨します。
ただし、整合性の確認はパフォーマンスに影響する可能性がありますのでご注意ください。

```
snipet:
void* mocMemory;
unsigned int mocSize;

//64byteアライメントされたメモリアドレスにファイルを読み込む。
//mocSizeには.moc3のファイルサイズが格納されます。
mocMemory = ReadBlobAligned("Koharu/Koharu.moc3", csmAlignofMoc, &mocSize);

// .moc3の整合性を確認
int consistency = csmHasMocConsistency(mocMemory, mocSize);

if(!consistency)
{
    // 整合性が確認できなければ処理しない
    return;
}

// モデルのインスタンスを作成
csmModel* model = csmInitializeModelInPlace(moc, modelMemory, modelSize);
```

使用APIへのリンク

csmHasMocConsistency

・moc3のファイルバージョン

moc3ファイルはファイル規格のバージョンが存在します。
Coreは自身の対応バージョン以下のmoc3ファイルに対して互換性を持ちます。
Coreが対応できるファイルバージョンはcsmGetLatestMocVersionで確認ができます。

```
/** moc3 file format version. */
enum
{
    /** unknown */
    csmMocVersion_Unknown = 0,
    /** moc3 file version 3.0.00 - 3.2.07 */
    csmMocVersion_30 = 1,
    /** moc3 file version 3.3.00 - 3.3.03 */
    csmMocVersion_33 = 2,
    /** moc3 file version 4.0.00 - 4.1.05 */
    csmMocVersion_40 = 3,
    /** moc3 file version 4.2.00 - */
    csmMocVersion_42 = 4
};

/** moc3 version identifier. */
typedef unsigned int csmMocVersion;

/**
 * Gets Moc file supported latest version.
 *
 * @return csmMocVersion (Moc file latest format version).
 */
csmApi csmMocVersion csmGetLatestMocVersion();

/**
* Gets Moc file format version.
*
* @param  address  Address of moc.
* @param  size     Size of moc (in bytes).
*
* @return csmMocVersion
*/
csmApi csmMocVersion csmGetMocVersion(const void* address, const unsigned int size);
```

読み込んだmoc3のファイルバージョンはcsmGetMocVersionを使用することで確認ができます。
moc3ファイルでない場合はcsmMocVersion_Unknown = 0を返します。
ファイルバージョンを確認するcsmGetMocVersionはcsmReviveMocInPlace実行の前後に関わらず、メモリアドレスに対して実行できます。
取得したファイルバージョンとCoreが扱えるバージョンを比較することで、読み込みが可能か確認することができます。

moc3のファイルバージョンを確認しながらのモデル生成

```
snippet:

    void* mocMemory;
    unsigned int mocSize;

    //64byteアライメントされたメモリアドレスにファイルを読み込む。
    //mocSizeには.moc3のファイルサイズが格納されます。
    mocMemory = ReadBlobAligned("Koharu/Koharu.moc3", csmAlignofMoc, &mocSize);

    const csmMocVersion fileVersion = csmGetMocVersion(mocMemory, mocSize);

    if( ( csmGetLatestMocVersion() < fileVersion ) ||
        ( fileVersion == 0 ) )
    {
        Log("can’t load moc3 file");
        return;
    }

    csmMoc* moc = csmReviveMocInPlace(mocMemory, mocSize);

    unsigned int modelSize = csmGetSizeofModel(moc);

    // モデルは16byteでアライメントする必要がある
    void** modelMemory = AllocateAligned(modelSize, csmAlignofModel);

    // モデルのインスタンスを作成
    csmModel* model = csmInitializeModelInPlace(moc, modelMemory, modelSize);
```

古いバージョンのCoreで新しいファイルの読み込みを試みた場合、csmReviveMocInPlaceの返り値がNULLなります。
このとき、CoreがcsmGetVersion()で得られるバージョン情報が03.03.0000以降(50528256)であればCoreのログの中に下記のようなメッセージが出力されます。

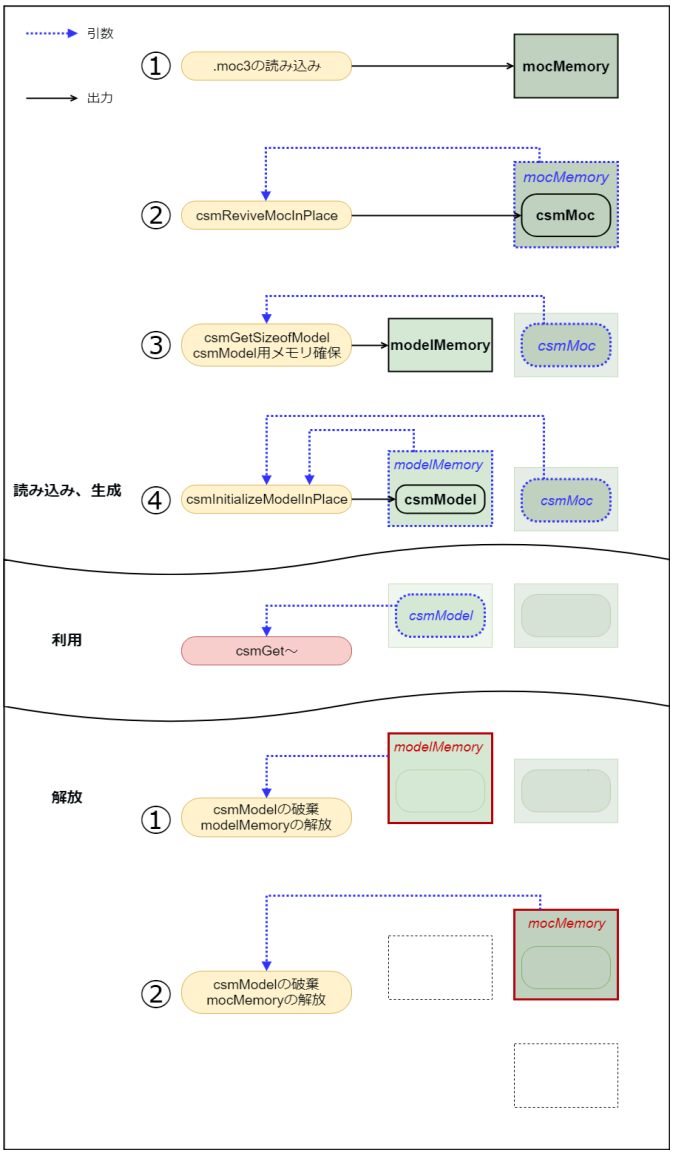
```
csmReviveMocInPlace is failed. The Core unsupport later than moc3 ver:2. This moc3 ver is 3.
```

Coreは最新のものを使用するようにしてください。

・csmMoc、csmModelの解放

csmReviveMocInPlace、csmInitializeModelInPlaceは入力されたメモリ空間内でのみ操作を行います。
返すアドレスは必ず用意したメモリ空間の中のものになります。
csmReviveMocInPlace、csmInitializeModelInPlaceで入力に使ったメモリ空間にcsmMoc、csmModelが存在するので、入力したメモリ空間は保持しつける必要があります。
また、csmMocは対応するcsmModelすべてが破棄されるまで保持する必要があります。
これはcsmModelがcsmMocを参照しているためです。
csmMoc、csmModelを破棄するとき、メモリの解放はcsmMocやcsmModelのアドレスではなく、生成に使ったmocMemoryやmodelMemoryを対象として解放してください。

下にメモリ確保、解放についての流れを示します。

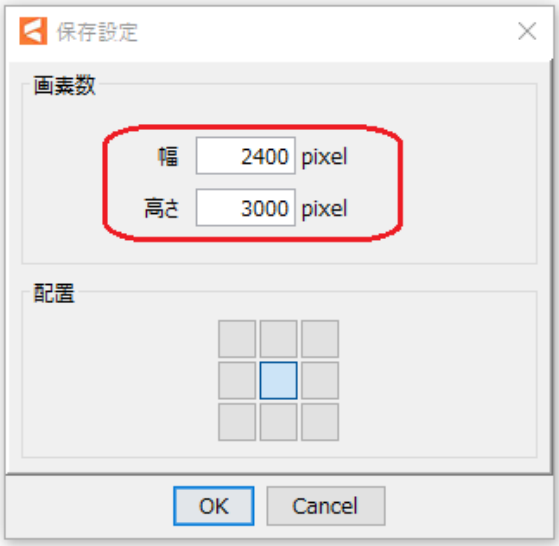


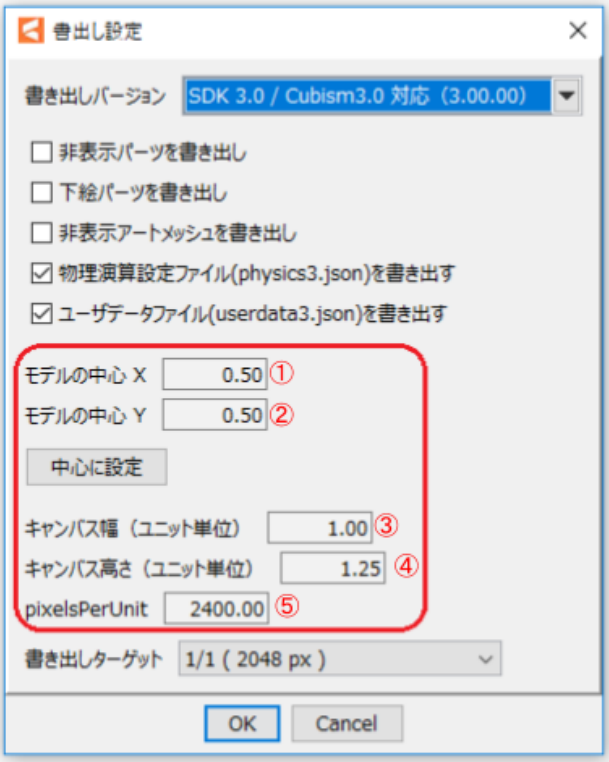
APIへのリンク

csmReviveMocInPlace
csmGetSizeofModel
csmInitializeModelInPlace
csmGetMocVersion
csmGetLatestMocVersion

・モデルの描画上サイズを得る

Editorで作業エリアとして表示されるキャンバスサイズ、
モデルファイルを出力するときに指定できる中央位置やユニットの位置について
取得することができます。





モデルのキャンバス情報へのアクセス

```
snippet:
  csmVector2 size;
  csmVector2 origin;
  float pixelsPerUnit;

  csmReadCanvasInfo(Sample.Model, &size, &origin, &pixelsPerUnit);

  printf(“size.X=%5.1f”,size.X); // size.X = 2400.0 = (3) * (5)
  printf(“size.Y=%5.1f”,size.Y); // size.Y = 3000.0 = (4) * (5)
  printf(“origin.X=%5.1f”,origin.X); // origin.X = 1200.0 = (1) * (5)
  printf(“origin.Y=%5.1f”,origin.Y); // origin.Y = 1500.0 = (2) * (5)
  printf(“pixelsPerUnit=%5.1f”,pixelsPerUnit); // pixelsPerUnit = 2400.0 =(5)
```

使用APIへのリンク

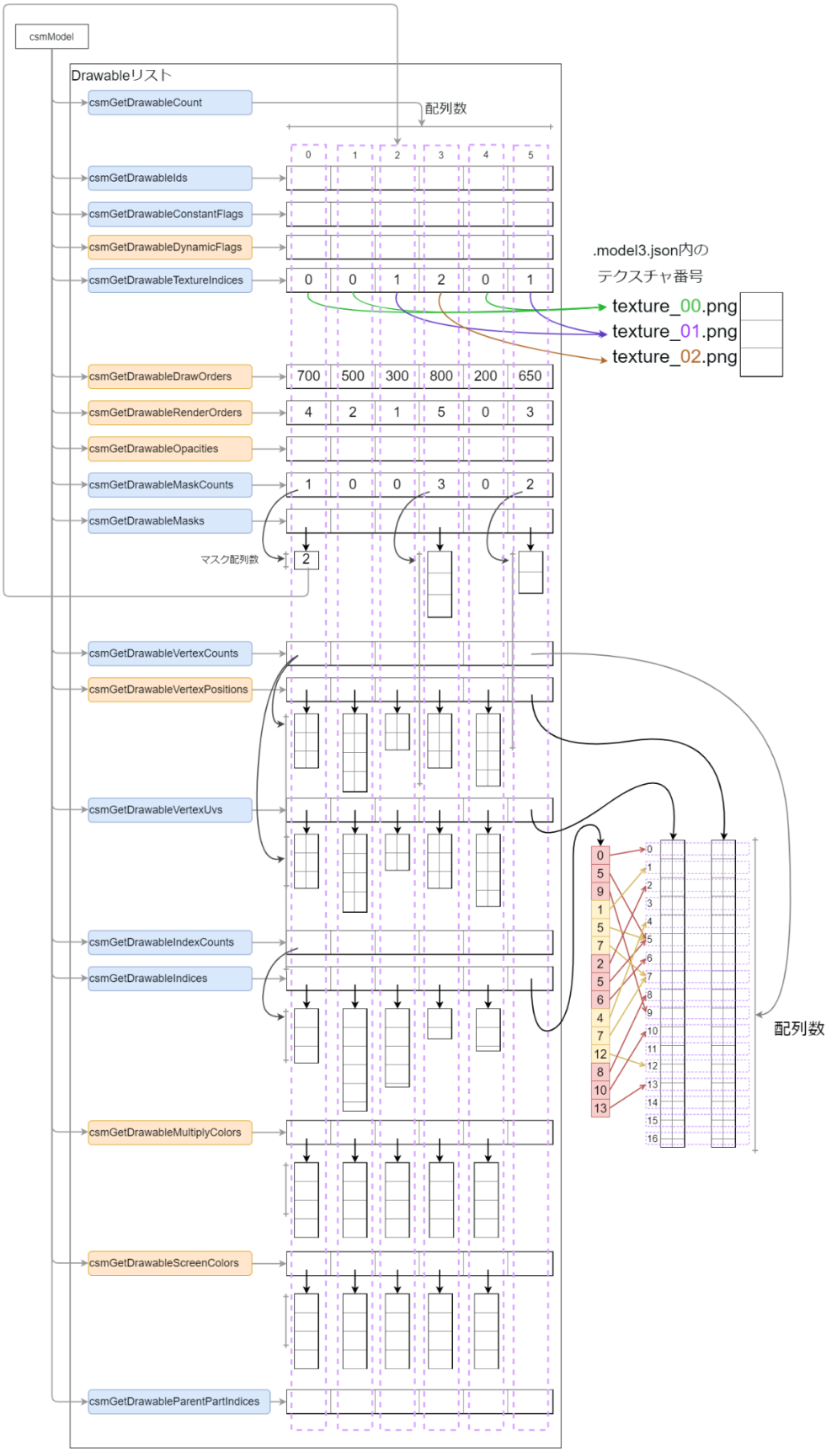
csmReadCanvasInfo

・Drawableの読み込みと配置

DrawableはCore内での一つの描画単位です。
DrawableはEditor上の１つのアートメッシュに対応するものです。
Drawableは描画する上で必要な情報を持っています。
moc3から読み込んだデータには変化しない静的な情報と、パラメータの値の変更によって変化する動的な情報があります。静的な情報は一度取得してアプリケーション側でキャッシュさせておくことが可能です。

csmGet[hogehoge]Countを持つグループはstructure of array(SOA)で、Countによって配列数を得られます。
csmGetDrawableTextureIndicesなどのAPIから得られるのは配列の先頭アドレスです。
配列内の並びは一致しており、目的のパラメータを探すときにはcsmGetcsmGetDrawableIdsで得られる配列から探します。
パラメータ、パーツなども同様の構造で表現されます。

下の図はDrawableリストの構造です。
csmUpdateModelが実行されたときに、
青のAPIは静的な項目で、オレンジのAPIは動的な項目です。

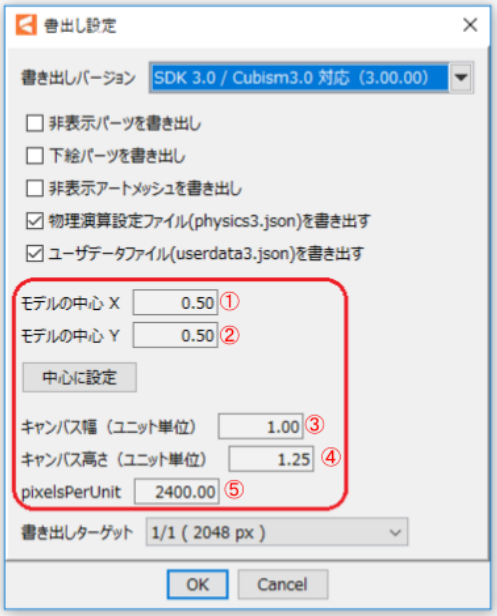


Drawableの読み込みではグラフィックスAPIへの登録や描画順序ソートのための構造生成など、Rendererに合わせた描画に向けての準備をします。
DrawableのSOAからAOS構造への変換

```
snippet:
// 初期化
drawableCount  = csmGetDrawableCount(model);
drawables      = Allocate(sizeof(Drawable) * drawableCount);


textureIndices   = csmGetDrawableTextureIndices(model);
constantFlags    = csmGetDrawableConstantFlags(model);

vertexCounts     = csmGetDrawableVertexCounts(model);
vertexPositons   = csmGetDrawableVertexPositions(model);
vertexUvs        = csmGetDrawableVertexUvs(model);

indexCounts      = csmGetDrawableIndexCounts(model);
vertexIndices    = csmGetDrawableIndices(model);

ids              = csmGetDrawableIds(model);
opacities        = csmGetDrawableOpacities(model);
drawOrders       = csmGetDrawableDrawOrders(model);
renderOrders     = csmGetDrawableRenderOrders(model);
dynamicFlags     = csmGetDrawableDynamicFlags(model);

maskCounts       = csmGetDrawableMaskCounts(model);
masks            = csmGetDrawableMasks(model);

// Initialize static drawable fields.
for (d = 0; d < drawableCount; ++d)
{
        drawables[d].TextureIndex       = textureIndices[d];

        if ((constantFlags[d] & csmBlendAdditive) == csmBlendAdditive)
        {
                drawables[d].BlendMode = csmAdditiveBlending;
        }
        else if ((constantFlags[d] & csmBlendMultiplicative) == csmBlendMultiplicative)
        {
                drawables[d].BlendMode = csmMultiplicativeBlending;
        }
        else
        {
                drawables[d].BlendMode = csmNormalBlending;
        }

        drawables[d].IsDoubleSided      =
                                (constantFlags[d] & csmIsDoubleSided) == csmIsDoubleSided;
        drawables[d].IsInvertedMask     =
                                (constantFlags[d] & csmIsInvertedMask) == csmIsInvertedMask;
```

```
        drawables[d].VertexCount        = vertexCounts[d];
        drawables[d].VertexPositions    = Allocate(sizeof(Vector3) * vertexCounts[d]);
        drawables[d].VertexUvs = Allocate(sizeof(Vector2) * vertexCounts[d]);

        //VertexPositionsとVertexUvsはともに2次元の表記になるが
        //vertexCountsはindicesと違い頂点数。
        for (i = 0; i < vertexCounts[d]; ++i)
        {
                drawables[d].VertexPositions[i].x = vertexPositons[d][i].X;
                drawables[d].VertexPositions[i].y = vertexPositons[d][i].Y;
                //Coreからくるvertexpositionはx,yの2つのみであることに注意
                drawables[d].VertexPositions[i].z = 0;

                drawables[d].VertexUvs[i].x = vertexUvs[d][i].X;
                drawables[d].VertexUvs[i].y = vertexUvs[d][i].Y;
        }

        //vertexIndices[d]はすべて三角形での表記 indexCounts[d]は必ず3の倍数になる。
        drawables[d].IndexCount         = indexCounts[d];
        drawables[d].Indices    = vertexIndices[d];   //一元配列として取得

        //VertexPositions,VertexUvs,vertexIndicesなどの値をグラフィックスAPIに登録する
        drawables[d].Mesh = MakeMesh(drawables[d].VertexCount,
                                     drawables[d].VertexPositions,
                                     drawables[d].VertexUvs,
                                     drawables[d].IndexCount,
                                     drawables[d].Indices);

        //その他のDrawable要素へのアクセス
        drawables[d].ID                 = ids[d];
        drawables[d].DrawOrder          = drawOrders[d];

        //下の3つは描画の更新時に重要
        drawables[d].Opacity    = opacities[d];
        drawables[d].RenderOrder = renderOrders[d];
        drawables[d].DynamicFlag = dynamicFlags[d];

        drawables[d].MaskCount = maskCounts[d];
        drawables[d].Masks    = Allocate(sizeof(int) * maskCounts[d]);
        for (m = 0; m < maskCounts[d]; ++m)
        {
                drawables[d].Masks[m]  = masks[d][m];

                //masksに入っている数字はDrawableのIndex
                drawables[d].MaskLinks = &drawables[(masks[d][m])];
        }
}
```

csmGetDrawableVertexPositionsで得られる頂点位置XYはCubism Editorからの組み込み向け出力時にキャンバス設定のPixelsPerUnitに影響を受けます。
XYの値はユニット単位で示され、以下の式によって求めることができます。



$$X = (localX / ⑤) - (①\text{x}③)$$
$$Y = (②\text{x}④) - (localY / ⑤)$$

縦横比が保たれた頂点位置が記録されています。
頂点位置がキャンバスの範囲を超えていても超えた状態での座標が記録されます。
詳しくはSDKマニュアルの「DrawableVertexPotionsの範囲」を確認してください。

使用APIへのリンク

csmGetDrawableCount
csmGetDrawableIds
csmGetDrawableConstantFlags
csmGetDrawableDynamicFlags
csmGetDrawableTextureIndices
csmGetDrawableDrawOrders
csmGetDrawableRenderOrders
csmGetDrawableOpacities
csmGetDrawableMaskCounts
csmGetDrawableMasks
csmGetDrawableVertexCounts
csmGetDrawableVertexPositions
csmGetDrawableVertexUvs
csmGetDrawableIndexCounts

csmGetDrawableIndices

・Drawableの親パーツを取得

パーツは木構造でできています。
この木構造はエディタ上の操作で作成され、moc3から生成されるcsmModelでも構造の情報を保持しています。
csmGetDrawableParentPartIndicesによってDrawableの親のPartを配置の順番の情報として確認することができます。
順番情報が-1を示すときにはRoot直下のパーツであることを示しています。

```
snippet:
// 初期化
partIds =csmGetPartIds(model);
drawableCount  = csmGetDrawableCount(model);
drawableParentPartIndices = csmGetDrawableParentPartIndices(model);
drawableIds = csmGetDrawableIds(model);

// drawableParentIndices[i] = -1であれば親なし
// drawableParentIndices[i] >= 0であれば取得したindexのdrawableParentIndices[i]が親のIDを指す
for (int i = 0; i < drawableCount; ++i)
{
    if(drawableParentPartIndices[i] == -1)
    {
        printf("drawableParentPartIndices[%d]:%s の親のパーツはありません。", i, drawableIds[i]);
    }
    else
    {
        printf("drawableParentPartIndices[%d]:%s の親のパーツは %s です。", i, drawableIds[i],
partIds[drawableParentPartIndices[i]]);
    }
}
```

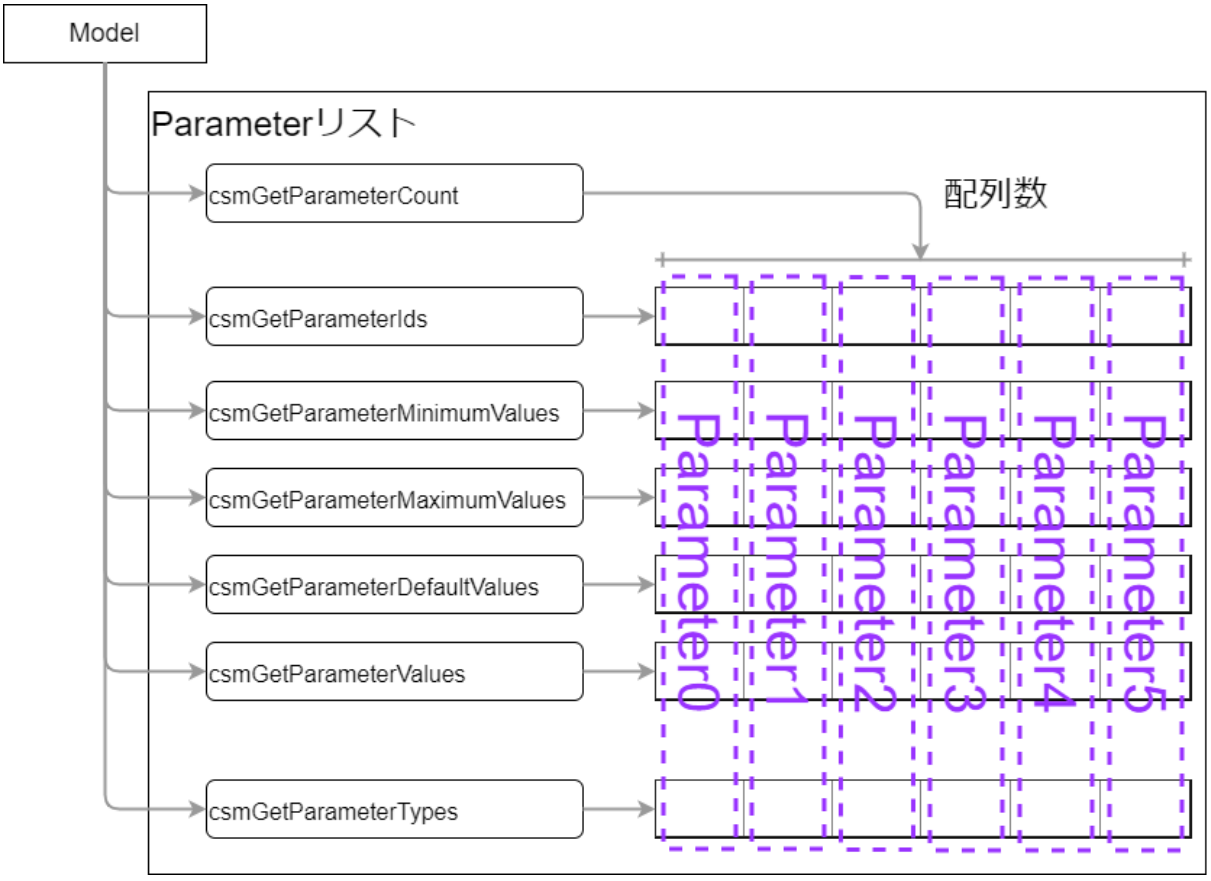
使用APIへのリンク

csmGetDrawableParentPartIndices

モデルを操作する

・パラメータの各要素を取得する

モデルの動きの操作にはパラメータの各要素を把握することが必要です。
各要素とは
・ID
・現在値
・最大値
・最小値
・初期値
の5つです。



各パラメータの要素へのアクセス

```
snipt:
  parameterCount = csmGetParameterCount(model);
  parameterIds = csmGetParameterIds(model);
  parameterValues = csmGetParameterValues(model);
  parameterMaximumValues = csmGetParameterMaximumValues(model);
  parameterMinimumValues = csmGetParameterMinimumValues(model);
  parameterDefaultValues = csmGetParameterDefaultValues(model);
  targetnum  = -1;

  for( i = 0; i < parameterCount ;++i)
  {
     if( strcmp(“ParamMouthOpenY”,parameterIds[i]) == 0 )
     {
        targetnum = i;
        break;
     }
  }
   //目的のIDは発見できなかった
   if(targetnum == -1 )
   {
      return;
   }

// モデルの”ParamMouthOpenY”パラメータの最小値、最大値、初期値が出てくる。
// min:0.0 max:1.0 default:0.0
 printf(“min:%3.1f max:%3.1f default:%3.1f”, parameterMinimumValues[targetnum]
                                                , parameterMaximumValues[targetnum]
                                                , parameterDefaultValues[targetnum] );
```

また、直接動きの操作には関わりませんが、ブレンドシェイプなどのパラメータに設定されている種別を取得することも可能です。

各パラメータに設定された種別の取得

```
snipt:
  /** Parameter types. */
  enum
  {
    /** Normal Parameter. */
    csmParameterType_Normal = 0,

    /** Parameter for blend shape. */
    csmParameterType_BlendShape = 1
  };

  /** Parameter type. */
  typedef int csmParameterType;
```

```
parameterCount = csmGetParameterCount(model);
parameterIds = csmGetParameterIds(model);
parameterTypes = csmGetParameterTypes(model);

for( i = 0; i < parameterCount ;++i)
{
  switch( parameterTypes[i] )
  {
    case csmParameterType_Normal :
      printf( ”%s : Normal\n”, parameterIds[i] );
      break;

    case csmParameterType_BlendShape :
      printf( ”%s : BlendShape\n”, parameterIds[i] );
      break;
  }
}
```

使用APIへのリンク

csmGetParameterCount
csmGetParameterIds
csmGetParameterValues
csmGetParameterMaximumValues
csmGetParameterMinimumValues
csmGetParameterDefaultValues
csmGetParameterTypes

・パラメータの操作

Cubismモデルへの操作においてパラメータの操作は、
パラメータの配列のアドレスを取得して、値を書き込むことで反映されます。

csmUpdateModel()を呼び出した時点でパラメータの最小値から最大値の範囲にクランプされます。
パラメータにリピート設定がされている場合は、クランプされることはありません。

```
snippet:
 //
  parameterIds = csmGetParameterIds(model);
  parameterValues = csmGetParameterValues(model);
  parameterDefaultValues = csmGetParameterDefaultValues(model);

    // 目的のIDに対応する配列位置を走査する
  targetIndex  = -1;

  for( i = 0; i < parameterCount ;++i)
  {
     if( strcmp(“ParamMouthOpenY”,parameterIds[i]) == 0 )
     {
        targetIndex = i;
        break;
     }
  }
   //目的のIDは発見できなかった
   if(targetIndex == -1 )
   {
     return;
   }


    //目的のパラメータに対して、基準値からの差を指定倍率で増大させる。
    parameterValues[targetIndex] =
                ( value - parameterDefaultValues[targetIndex] ) * multipleValues[targetIndex] +
                 parameterDefaultValues[targetIndex];
```

使用APIへのリンク

csmGetParameterValues
csmGetParameterDefaultValues

・パーツの不透明度の操作

パーツの不透明度の操作は、パラメータの操作と同様です。
配列のアドレスを取得し、そのメモリーに対し値を書き込むことで反映されます。
csmUpdateModelの処理によって0.0～1.0の範囲にクランプされます。

```
snippet:
  // Manipulate opacity
  partOpacities = csmGetPartOpacities(model);

    // Find parameter index.
  targetIndex  = -1;

  for( i = 0; i < parameterCount ;++i)
  {
      if( strcmp(“ParamMouthOpenY”,parameterIds[i]) == 0 )
      {
          targetIndex = i;
          break;
      }
  }
   //目的のIDは発見できなかった
   if(targetIndex == -1 )
   {
      return;
   }


    partOpacities[targetIndex] = value;
  }
```

使用APIへのリンク

csmGetPartOpacities

・パーツの親パーツを取得

パーツは木構造でできています。
この木構造はエディタ上の操作で作成され、moc3から生成されるcsmModelでも構造の情報を保持しています。
csmGetPartParentPartIndicesによってPartの親を配置の順番の情報として確認することができます。
順番情報が-1を示すときにはRoot直下のパーツであることを示しています。

snippet:

```
// パーツのIDリストを取得
const char** partIds =csmGetPartIds(model);

// パーツの親のindexリストを取得
const int* parentPartIndices =csmGetPartParentPartIndices(model);

// partParentIndex = -1であれば親なし
// partParentIndex >= 0であれば取得したindexのparentPartIndicesが親のIDを指す
for (int i = 0; i < partCount; ++i)
{
    if(partParentIndex[i] == -1)
    {
        printf("partParentIndex[%d]:%s の親のパーツはありません。", i, partIds[i]);
    }
    else
    {
        printf("partParentIndex[%d]:%s の親のパーツは %s です。", i, partIds[i], partIds[parentPartIndices[i]]);
    }
}
```

親パーツへの不透明度操作は子の不透明度へも適用されます

使用APIへのリンク

csmGetPartParentPartIndices

・操作をモデルへ適用する

パラメータやパーツの不透明度を変更したあと、実際のDrawableの頂点や不透明度に操作を反映しなければなりません。
この操作をするのがcsmUpdateModelです。
なお、詳細は「DynamicFlagのリセット」で説明しますが、描画に必要な情報のいずれが変更されたかを判断するためにcsmUpdateModel()の前にcsmResetDrawableDynamicFlags()を呼び出しておきます。

snippet:

```
// Update model.
csmUpdateModel(Model);
```

このとき影響があるのは
csmGetDrawableDynamicFlags
csmGetDrawableVertexPositions
csmGetDrawableDrawOrders
csmGetDrawableRenderOrders
csmGetDrawableOpacities
です。

使用APIへのリンク

csmUpdateModel
csmGetDrawableDynamicFlags
csmGetDrawableVertexPositions
csmGetDrawableDrawOrders
csmGetDrawableRenderOrders
csmGetDrawableOpacities

・DynamicFlagのリセット

csmResetDrawableDynamicFlagsは呼ばれた次のcsmUpdateModelで前回の値との違いをcsmGetDrawableDynamicFlagsへ書き込むように指示します。
この操作を行わなかったとき、csmGetDrawableDynamicFlagsで更新されるのはcsmIsVisibleの項目だけになります。
csmGetDrawableDynamicFlagsは描画のために実行されるcsmUpdateModelの直前に呼びます。

```
snippet:

  // Reset dynamic drawable flags.
  csmResetDrawableDynamicFlags(Sample.Model);
```

使用APIへのリンク

csmResetDrawableDynamicFlags

描画

・描画に必要なこと

描画ではモデルの操作を受けて次の処理ステップが必要です。
・Drawableの頂点の更新
・Drawableの不透明度の更新
・描画順序のソート
・Drawableの有効性の確認、有効でなければ描画しない
・マスク処理

また、Cubismでの描画には
・テクスチャの透明度同士の合成
・Additive合成、Multiplicative 合成
・カリングの有無
・クリッピングにおけるマスクの反転の有無
・乗算色
・スクリーン色
という要素があります。
Cubismモデルの描画を実装する際、これらをEditorと同じように再現する必要があります。

・描画の仕様

ConstantFlagsでの要素確認

Drawableごとの合成方法、カリングの有無、クリッピングにおけるマスクの反転の有無に関してはcsmGetDrawableConstantFlagsで取得できます。

得られたFlagの意味に関してはLive2DCubismCore.h内の定数を確認してください。

```
snippet:
    /** Bit masks for non-dynamic drawable flags. */
    enum
    {
    /** Additive blend mode mask. */
        csmBlendAdditive = 1 << 0,

        /** blend mode mask. */
        csmBlendMultiplicative = 1 << 1,

        /** Double-sidedness mask. */
        csmIsDoubleSided = 1 << 2,

        /**Clipping mask inversion mode mask. */
        csmIsInvertedMask = 1 << 3
    };
```

csmBlendAdditiveとcsmBlendMultiplicativeはどちらか片方の適用になります。

色合成についての計算式

各色要素が0.0から1.0で構成しているとした中で、
D=RGBA(Drgb,Da)を描画先に既存に入っている色情報
S=RGBA(Srgb,Sa)を描画する色情報としたとき、
出力結果O=RGBA(Orgb,Oa)は

Normal合成

$$Orgb = Drgb \times (1 - Sa) + Srgb$$
$$Oa = Da \times (1 - Sa) + Sa$$

Additive合成

$$Orgb = Drgb + Srgb$$
$$Oa = Da$$

Multiplicative合成

$$Orgb = Drgb \times (1 - Sa) + Srgb \times Drgb$$
$$Oa = Da$$

という計算になるように描画してください。

アルファ値のあるバッファなどに描画する際は透明背景にMultiplicative、Additiveを掛けると描画されないので注意してください。

カリングの方向とDrawableIndices

Coreから手に入るDrawableIndicesでは反時計回りが表面として表現されます。
使用する描画APIに合わせてカリング制御を調整してください。

クリッピングの仕様

クリッピングは描画ソースに対するすべてのマスクの合成後のアルファ値の乗算で行います。
複数のマスクの合成はDrawableの不透明度は1で固定し、合成方法の指定は無視してNormal合成を使って合成されます。テクスチャの不透明度は適用します。
カリングは通常の描画方法と同じく適用します。
マスクするDrawableにおいてマスクの反転が有効になっている場合は、合成後のマスクのアルファ値を反転させます。詳しくは「描画時にマスクを適用する」を参照してください。

・どの情報が更新されたか確認する

Drawableの頂点座標、不透明度、描画順など、変化があった項目だけ更新することで、処理全体の高速化が見込めます。更新された項目は、csmGetDrawableDynamicFlags で取得できます。

DynamicFlagの確認と頂点情報の更新、ソートフラグの処理

```
snippet:
for ( d = 0; d < csmGetDrawableCount(model); d++)
{
        dynamicFlags = csmGetDrawableDynamicFlags(model);

        isVisible = (dynamicFlags[d] & csmIsVisible) == csmIsVisible;

        if ((dynamicFlags[d] & csmVertexPositionsDidChange) ==   csmVertexPositionsDidChange)
        {
                /*    update vertexs   */
        }

        // Check whether drawables need to be sorted.
        sort = sort ||
              ((dynamicFlags[d] & csmRenderOrderDidChange) == csmRenderOrderDidChange);
}

if (sort)
{
        /*   render order need sort    */
}
```

csmGetDrawableDynamicFlagsによって得られる情報は以下の6つです。

snippet:

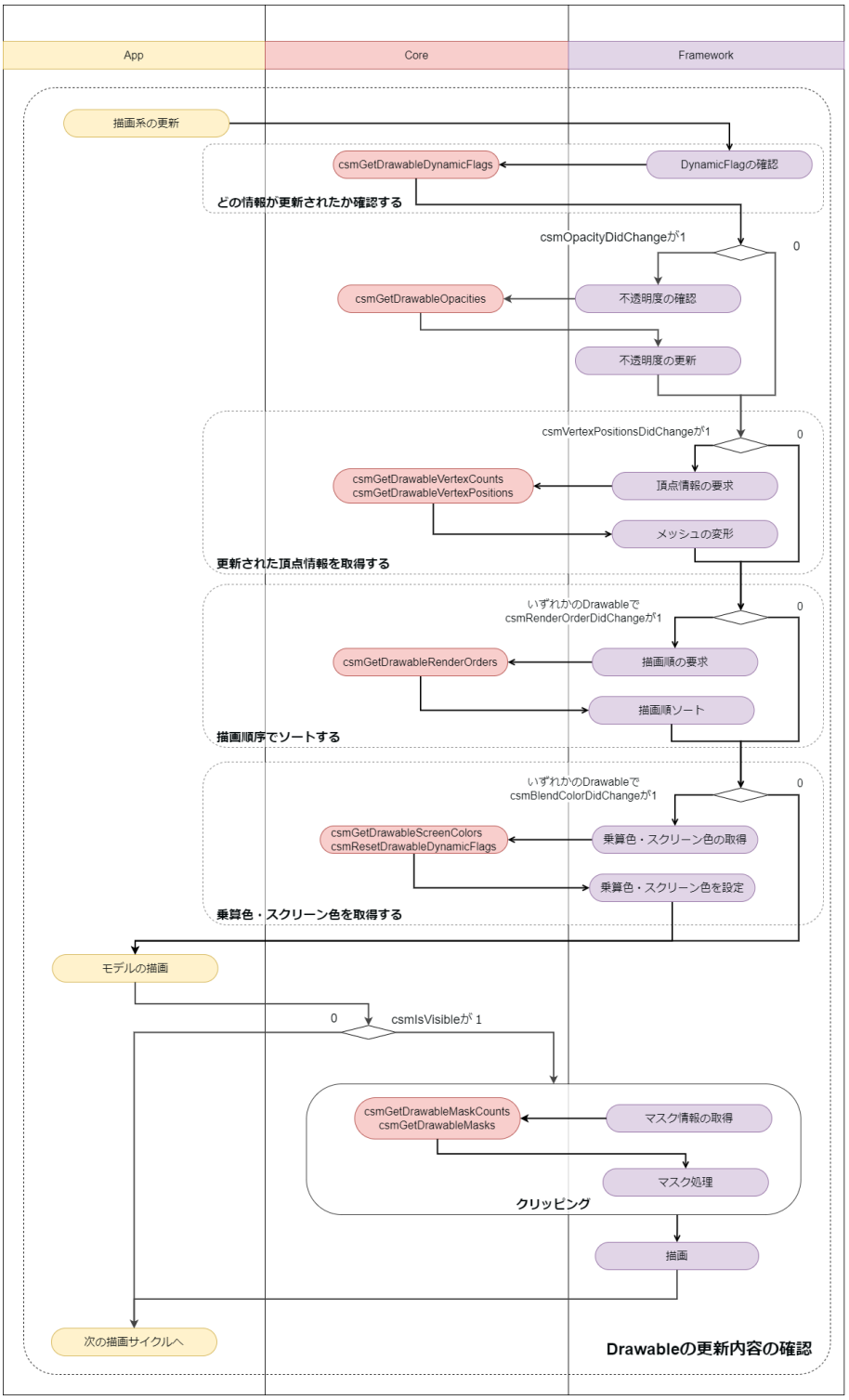
```
/** Bit masks for dynamic drawable flags. */
enum
{
    /** Flag set when visible. */
    csmIsVisible = 1 << 0,
    /** Flag set when visibility did change. */
    csmVisibilityDidChange = 1 << 1,
    /** Flag set when opacity did change. */
    csmOpacityDidChange = 1 << 2,
    /** Flag set when draw order did change. */
    csmDrawOrderDidChange = 1 << 3,
    /** Flag set when render order did change. */
    csmRenderOrderDidChange = 1 << 4,
    /** Flag set when vertex positions did change. */
    csmVertexPositionsDidChange = 1 << 5
};
```

各フラグに関しての解説

csmIsVisible	Drawableが表示になった時にビットが立ちます。 パラメータがキーの範囲外になっているか Drawableの不透明度の計算結果が0の時にビットが落ちます。
csmVisibilityDidChange	csmIsVisibleが以前の状態から変化があったときにビットが立ちます。
csmOpacityDidChange	Drawableの不透明度に変化があったときにビットが立ちます。
csmDrawOrderDidChange	Drawableの描画順に変化があったときにビットが立ちます。 描画の順序が入れ替わった時ではないことに注意してください。
csmRenderOrderDidChange	描画の順序に変更があったときにビットが立ちます。 描画の順序をソートする必要があります。
csmVertexPositionsDidChange	VertexPositionsが変化したときにビットが立ちます。
csmBlendColorDidChange	Drawableの乗算色・スクリーン色に変更があったときにビットが立ちます。 乗算色・スクリーン色のどちらかが変更されたのかまでは

	判定できないことに注意してください。

フラグの確認処理のフロー図



使用APIへのリンク

csmGetDrawableDynamicFlags

・更新された頂点情報を取得する

更新された頂点情報を受け取り、Rendererへと情報をコピーします。
初期化のときに読み込んだ頂点や不透明度を更新するだけです。

頂点情報と不透明度の更新

```
snippet:
  // Initialize locals.
  dynamicFlags = csmGetDrawableDynamicFlags(renderer->model)
  vertexPositions = csmGetDrawableVertexPositions(renderer->Model);
  opacities = csmGetDrawableOpacities(renderer->Model);

  for (d = 0; d < renderer->DrawableCount; ++d)
  {
    // Update 'inexpensive' data without checking flags.
    renderer->drawables[d].Opacity = opacities[d];

    // Do expensive updates only if necessary.
    if ((dynamicFlags[d] & csmVertexPositionsDidChange) ==
csmVertexPositionsDidChange))
    {
        //グラフィックスへ頂点情報の更新を行う
        for( i = 0; i < renderer->drawables[d].vertexCount; ++i)
        {
            renderer->drawables[d].vertexPositons[i].x = vertexPositions[d][i].x;
            renderer->drawables[d].vertexPositons[i].y = vertexPositions[d][i].y;
        }
        UpdateGraphicsVertexPosition( renderer->drawables[d] );
    }
  }
```

使用APIへのリンク

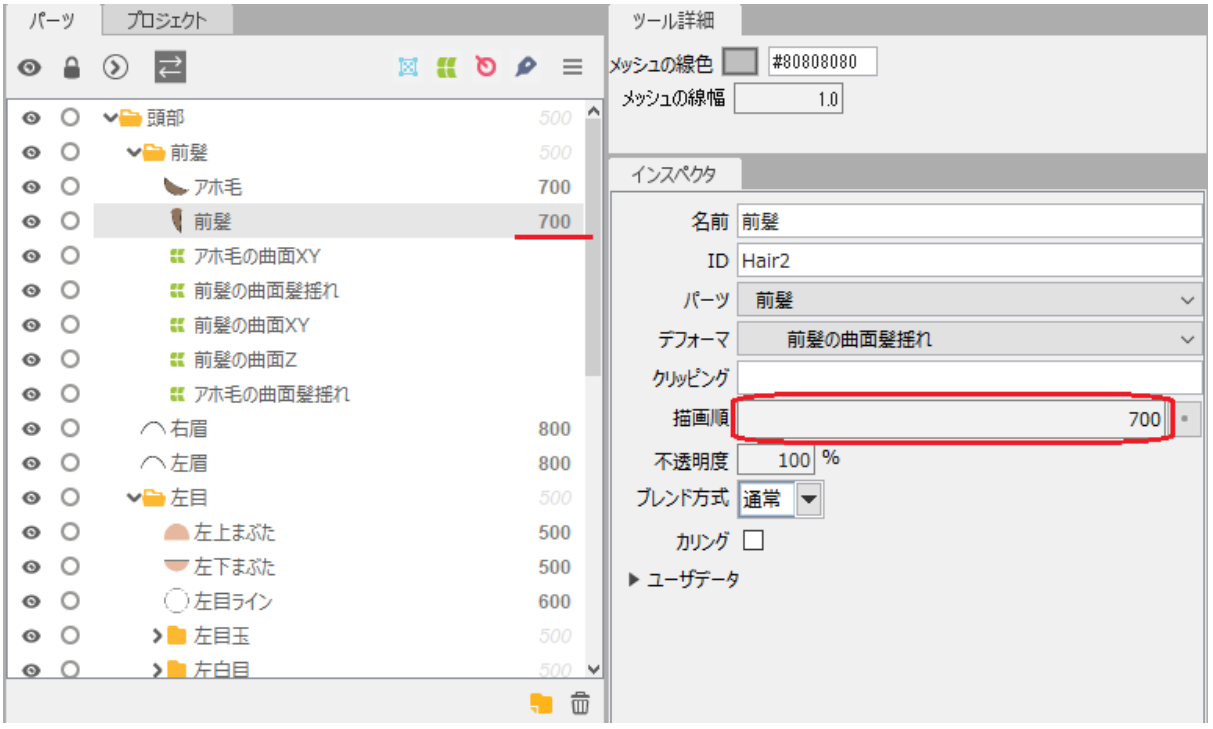
csmGetDrawableVertexPositions
csmGetDrawableDynamicFlags
csmGetDrawableOpacities

・Drawableの描画順序をソートする

パラメータの変化によってDrawOrderが変化し、
その結果として、RenderOrderが変化した場合、描画の呼び出し順序を変更する必要があります。

・描画順と描画順序

描画順（DrawOrder）と描画順序（RenderOrder）は似ているようで別のものです。
描画順とはEditor上でアートメッシュに描画の際の順序を決めるうえで参考にする値のことです。



csmGetDrawableDrawOrdersで出力される値はCubism Editorのインスペクタ上の値で、
描画順グループの計算は考慮されません。

描画順グループも考慮した、実際にDrawableを描画する順序を示したものが描画順序となります。
この描画順序の取得にはcsmGetDrawableRenderOrders()を呼び出します。

ソートへ向けての初期化時の処理

```
snippet:
  // Initialize static fields.
  for (d = 0, count = csmGetDrawableCount(model); d < count; ++d)
  {
    sortableDrawable[d].DrawableIndex = d;
  }
```

ソート用の評価関数

```
snippet:
static int CompareSortableDrawables(const void *a, const void *b)
{
  const SortableDrawable* drawableA = (const SortableDrawable*)a;
  const SortableDrawable* drawableB = (const SortableDrawable*)b;

  return (drawableA->RenderOrder > drawableB->RenderOrder) -
(drawableA->RenderOrder < drawableB->RenderOrder);
}
```

ソート

```
snippet:
  renderOrders = csmGetDrawableRenderOrders(model);
  count = csmGetDrawableCount(model);

  // Fetch render orders.
  for (d = 0; d < count; ++d)
  {
    sortableDrawable[d].RenderOrder = renderOrders[sortableDrawable[d].DrawableIndex];
  }

  // Sort.
  qsort(sortableDrawable, count, sizeof(SortableDrawable), CompareSortableDrawables);
```

描画時のソーティング順アクセス

```
snippet:
  for (d = 0, count = csmGetDrawableCount(model); d < count; ++d)
  {
    target = &drawable[sortableDrawable[d].DrawableIndex];
    drawing( target  );
  }
```

使用APIへのリンク

csmGetDrawableCount
csmGetDrawableDrawOrders
csmGetDrawableRenderOrders

・描画時にマスクを適用する

あるDrawableが、どのDrawableにマスクされているのかを調べるには
csmGetDrawableMaskCountsとcsmGetDrawableMasksを使用します。
d番目のDrawableが、いくつのマスク用Drawableでマスキングされているのかは、
csmGetDrawableMaskCounts[d]で取得できます。
d番目のDrawableをマスクするi枚目のDrawableの配列上の番号は
csmGetDrawableMasks[d][i]で得られます。

複数枚のマスクするDrawableがある場合、マスクへの描画はそれぞれのDrawableのアルファのみを合成します。
マスク用に合成する際は、そのDrawableのBlendモードにAdditiveやMultiplicativeが設定されていた場合でも、必ずNormal合成で行います。
カリングの設定は適用して合成します。
マスクとして使用しても、表現的にマスク用のDrawableを画面に表示させないことがあるため、マスク同士の合成の際には、そのDrawableに設定された不透明度の値は使用しません。

アルファ値の扱いが0.0-1.0の範囲の場合、反転マスクが設定されているDrawableのアルファ値を「1.0 - 合成したマスクのアルファ値」となるように設定することで、マスクを反転させてクリッピング描画を行います。

描画処理の中での基本的なマスク処理とマスクDrawableへのアクセス

snippet:

```
/* 呼ばれる関数などはすべて仮のものです。*/
int d;
int drawableCount = csmGetDrawableCount(model);
const int *maskCount = csmGetDrawableMaskCounts(model);
const int **masks = csmGetDrawableMasks(model);
const csmFlags *dynamicFlags = csmGetDrawableDynamicFlags(model);
const csmFlags *constantFlags = csmGetDrawableConstantFlags(model);

for (d = 0; d < drawableCount; ++d)
{
    /* Sorters[d].RenderOrderはcsmGetDrawableRenderOrderを使って
     * ソート済みの描画順が格納されているものとする。 */
    target = Sorters[d].RenderOrder;
    if (maskCount[d] > 0)
    {
        /* マスクがある場合の描画 */
        /* マスクバッファをリセットする */
        ResetMaskBuffer();

        /* 描画先をマスクバッファに変更 */
        RenderTarget(MASK);

        /* マスクの描画に合わせて共通の設定を行う */
        SetRenderingOpacity(1.0f);              //不透明度は1.0で固定
        SetRenderingMode(RENDER_MODE_NORMAL);   //合成方法は通常で固定
        for (i = 0; i < maskCount[target]; ++i)
        {
            int maskDrawableIndex = masks[target][i];
            /* maskDrawableIndexが-1となる場合は非表示などで出力されてないケースです。
             * -1である場合はcontinueでマスク描画をスキップする。*/
            if( maskDrawableIndex == -1 )
            {
                continue;
            }

            /* マスクのDynamicFlagのcsmVertexPositionsDidChangeが立っていない場合
             * 頂点情報が使用できないため、continueでマスク描画をスキップする。*/
            if ((dynamicFlags[maskDrawableIndex] & csmVertexPositionsDidChange) !=
                csmVertexPositionsDidChange)
            {
                continue;
            }

            Drawable maskingDrawable = drawable[maskDrawableIndex];
            /* カリングの設定とテクスチャはマスク対象の設定を使用 */
            SetCulling(maskingDrawable.culling);
            SetMainTexture(maskingDrawable.texture);
```

```
        /* 描画 */
        DrawElements();
    }
    /* 描画先を通常のバッファへ戻す */
    RenderTarget(MAIN);

    /* Drawableの各描画要素を指定 */
    Drawable targetDrawable = drawable[target];
    SetRenderingOpacity(targetDrawable.opacity);
    SetRenderingMode(targetDrawable.renderMode);
    SetCulling(targetDrawable.culling);
    SetMainTexture(targetDrawable.texture);

    bool isInvertedMask = (constantFlags[target] & csmIsInvertedMask) != csmIsInvertedMask;
    /* マスクの使用を指定(Shaderが異なる場合はこの段階で指定) */
    /* マスクの反転の有無によってShaderを変える */
    SetMaskTexture(MASK, isInvertedMask);

    /* 描画 */
    DrawElements();
  }
  else
  {
    /*マスクなしでの描画*/
    /* Drawableの各描画要素を指定 */
    Drawable targetDrawable = drawable[target];
    SetRenderingOpacity(targetDrawable.opacity);
    SetRenderingMode(targetDrawable.renderMode);
    SetCulling(targetDrawable.culling);
    SetMainTexture(targetDrawable.texture);

    /* マスクの"未"使用を指定 */
    SetMaskTexture(NULL);

    /* 描画 */
    DrawElements();
  }
}
```

使用APIへのリンク

csmGetDrawableMaskCounts
csmGetDrawableMasks

・乗算色、スクリーン色の取得、適用

あるDrawableの乗算色、スクリーン色は、csmGetDrawableMultiplyColors、csmGetDrawableScreenColorsを使用します。
d番目のDrawableに設定されている乗算色はcsmGetDrawableMultiplyColors[d]、スクリーン色はcsmGetDrawableScreenColors[d]で取得できます。
d番目のDrawableに設定されている各色はcsmVector4型で取得でき、XにRの値が、YにGの値が、ZにBの値が、WにAの値が格納されています。
ただし、Cubism 4.2 SDKではAの値は使用しておりません。

乗算色が設定されていない場合の初期値は(1.0f, 1.0f, 1.0f, 1.0f)が設定されます。
これは、乗算色を適用する際にRGBの各値が乗算されるため、元の色へ影響しない値としてこの初期値が設定されています。

スクリーン色が設定されていない場合の初期値は(0.0f, 0.0f, 0.0f, 1.0f)が設定されます。
これは、スクリーン色を適用する際にRGBの各値が加算されるため、元の色へ影響しない値としてこの初期値が設定されています。

乗算色、スクリーン色を取得し、それらの色をシェーダに適用

snippet:

```
/* 呼ばれる関数などはすべて仮のものです。*/

/* 乗算色 */
const csmVector4* multiplyColor = csmGetDrawableMultiplyColors(model);

/* スクリーン色 */
const csmVector4* screenColor = csmGetDrawableScreenColors(model);

/* シェーダに乗算色、スクリーン色を適用 */
CubismShader_OpenGLES2::GetInstance()->SetupShaderProgram(
  this, drawTextureId, vertexCount, vertexArray,
  uvArray, opacity, colorBlendMode, modelColorRGBA,
  multiplyColor[drawableIndex],    // 乗算色
  screenColor[drawableIndex],    // スクリーン色
  isPremultipliedAlpha, mpvMatrix, invertedMask
);
```

使用APIへのリンク

csmGetDrawableMultiplyColors
csmGetDrawableScreenColors

・パラメータのキーを取得する

パラメータに設定されたキーを取得するには、csmGetParameterKeyCounts、csmGetParameterKeyValuesを使用します。
d番目のParameterに設定されているキーの数はcsmGetParameterKeyCounts[d]で取得できます。
d番目のParameterに設定されているi個目のキーの位置はcsmGetParameterKeyValues[d][i]で得られます。

パラメータに設定されたキーとその数を取得

snippet:

```
/* 呼ばれる関数などはすべて仮のものです。*/

/* パラメータに設定されたキーの数 */
const int* keyCounts = csmGetParameterKeyCounts(_model);

/* パラメータに設定された各キーの位置を取得 */
const float** keyValues = csmGetParameterKeyValues(_model);

const csmChar** parameterIds =csmGetParameterIds(_model);
const csmInt32  parameterCount = csmGetParameterCount(_model);

for (csmInt32 i = 0; i < parameterCount; ++i)
{
  printf("%s : %d\n", drawableIds[i], keyCounts[i]);
  for (csmInt32 j = 0; j < keyCounts[i]; ++j)
  {
    printf("3.1%f\n", keyValues[i][j]);
  }
}
```

使用APIへのリンク

csmGetParameterKeyCounts
csmGetParameterKeyValues

個別のAPI

APIの命名規則

・SOA構造

csmGetXXXXCountというAPIがあった時は
csmGetXXXXYYYYsのAPI群で得られる配列は同じ並びで格納されています。

詳しくはDrawableの読み込みと配置をご覧ください。

・InPlace

InPlaceの付くcsmReviveMocInPlace、csmInitializeModelInPlaceは入力されたメモリ空間内でのみ操作を行う仕組みであるAPIであることを示してます。

詳しくはcsmMoc、csmModelの解放をご覧ください。

csmGetVersion

Coreのバージョン情報を返します。

引数

なし

返り値

・csmVersion(unsigned int)

バージョンの表記はMAJOR, MINOR, PATCHの3つから構成されます。
それぞれの運用ルールについて下記に示します。

メジャーバージョン (1byte)
Cubism Editorがメジャーバージョンアップするなどしてモデルデータ(.moc3ファイル)と後方互換性がなくなった時にインクリメントされます。

マイナーバージョン (1byte)
モデルデータの後方互換性を保ちつつ機能追加が追加されたときにインクリメントされます。

パッチ番号 (2byte)
不具合が修正されたときにインクリメントされます。メジャーバージョンあるいはマイナーバージョンが変更された場合、パッチ番号は0にリセットされます。

```
0x    00      00    0000
    Major  Minor  Patch
```

バージョンは4byteから構成され、単に符号なし整数として扱うことで、新しいCoreのバージョンの方が必ず大きい数値を指すことになります。

記載のある項目

Coreのバージョンを確認する

利用可能バージョン

3.0.00以上

csmGetLatestMocVersion

Coreが取り扱える最新のファイルバージョンを返します。

引数

なし

返り値

・csmMocVersion(unsigned int)

記載のある項目

moc3のファイルバージョン

利用可能バージョン

3.3.01以上

csmGetMocVersion

.moc3ファイルのロードされたメモリを参照し、moc3ファイルのバージョンを返します。

引数

・void* address
.moc3が読まれたデータ配列の頭のアドレス

・const unsigned int size
.moc3が読まれたデータ配列の長さ

返り値

・csmMocVersion(unsigned int)

```
/** moc3 file format version. */
enum
{
    /** unknown */
    csmMocVersion_Unknown = 0,
    /** moc3 file version 3.0.00 - 3.2.07 */
    csmMocVersion_30 = 1,
    /** moc3 file version 3.3.00 - 3.3.03 */
    csmMocVersion_33 = 2,
    /** moc3 file version 4.0.00 - 4.1.05 */
    csmMocVersion_40 = 3,
    /** moc3 file version 4.2.00 - */
    csmMocVersion_42 = 4
};

/** moc3 version identifier. */
typedef unsigned int csmMocVersion;
```

ロードされた内容がmoc3ファイルでない場合はcsmMocVersion_Unknownを返します。
エディタ側のバージョンアップによってLive2DCubismCore.hの定義以上の値が来る可能性があることに気を付けてください。
使用できるファイルバージョンか調べるにはcsmGetLatestMocVersionの返り値と比較してください。

記載のある項目

moc3のファイルバージョン

利用可能バージョン

3.3.01以上

csmGetLogFunction

保存されたログ用の関数のポインタを返します。

引数

なし

返り値

・csmLogFunction(アドレス)

ログ関数の型

```
snippet:
/** Log handler.
    *
    * @param  message  Null-terminated string message to log.
    */
   typedef void (*csmLogFunction)(const char* message);
```

記載のある項目

Coreのログを出力する

利用可能バージョン

3.0.00以上

csmSetLogFunction

ログ出力に使う関数を指定します。

引数

・csmLogFunction handler

```
snippet:
/** Log handler.
    *
    * @param  message  Null-terminated string message to log.
    */
   typedef void (*csmLogFunction)(const char* message);
```

返り値

なし

記載のある項目

Coreのログを出力する

利用可能バージョン

3.0.00以上

csmReviveMocInPlace

.moc3ファイルのロードされたメモリ内でcsmMoc構造体を再生します。
addressで渡すアドレスは既定のアライメントを満たしている必要があります。

インクルードファイルにあるアライメントサイズの記述

```
snippet:
    /** Alignment constraints. */
    enum
    {
    /** Necessary alignment for mocs (in bytes). */
    csmAlignofMoc = 64,
    };
```

再生されたcsmMoc構造体を解放するタイミングはcsmMocから生成されたcsmModelがすべて解放された後に行ってください。
詳しくはmoc3ファイルを読み込んでcsmModelオブジェクトにまで展開するをご覧ください。

引数

・void* address
.moc3が読まれたデータ配列の頭のアドレス
アライメントする必要があります。

・const unsigned int size
.moc3が読まれたデータ配列の長さ

返り値

・csmMoc*
csmMoc構造体へのアドレス
問題があるときにはNULLになる。

記載のある項目

Mocファイルを読み込んでModelにまで展開する

利用可能バージョン

3.0.00以上

csmGetSizeofModel

Moc構造体から生成されるModel構造体のサイズを返します。
メモリ確保用に使います。

引数

・const csmMoc* moc
Moc構造体へのアドレス

返り値

・unsigned int
Model構造体のサイズ

記載のある項目

Mocファイルを読み込んでModelにまで展開する

利用可能バージョン

3.0.00以上

csmInitializeModelInPlace

Moc構造体からModel構造体を初期化します。
メモリはアライメントされたものを用意する必要があります。

インクルードファイルにあるアライメントサイズの記述

```
snippet:
    /** Alignment constraints. */
    enum
    {
    /** Necessary alignment for models (in bytes). */
    csmAlignofModel = 16
    };
```

引数

・const csmMoc* moc
Moc構造体へのアドレス

・void* address
確保したメモリのアドレス

・const unsigned int size
確保したメモリーのサイズ

返り値

・csmModel*

記載のある項目

Mocファイルを読み込んでModelにまで展開する

利用可能バージョン

3.0.00以上

csmUpdateModel

パラメータやパーツの操作を頂点情報などに反映します。

引数

・csmModel* model
モデル構造体へのアドレス

返り値

なし

記載のある項目

操作をモデルへ適用する

利用可能バージョン

3.0.00以上

csmReadCanvasInfo

モデルのキャンバスサイズや中心点、ユニットサイズを返します。

引数

・const csmModel* model
モデル構造体へのアドレス

・csmVector2* outSizeInPixels
モデルのキャンバスサイズを格納するためのcsmVector2へのアドレス

・csmVector2* outOriginInPixels
モデルのキャンバスの中心点を格納するためのcsmVector2へのアドレス

・float* outPixelsPerUnit
モデルのユニットの大きさ単位

返り値

なし

記載のある項目

モデルの描画上サイズを得る

利用可能バージョン

3.0.00以上

csmGetParameterCount

モデルが保有するパラメータの個数を返します。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・int
保有するパラメータの数

記載のある項目

パラメータの各要素を取得する

利用可能バージョン

3.0.00以上

csmGetParameterIds

モデルが持つパラメータのIDが格納された配列アドレスを返します。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const char**
文字列アドレスが格納される配列へのアドレス

記載のある項目

パラメータの各要素を取得する

利用可能バージョン

3.0.00以上

csmGetParameterTypes

モデルが持つパラメータのIDが格納された配列アドレスを返します。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const csmParameterType*
パラメータの種別が格納される配列へのアドレス

```
/** Parameter types. */
enum
{
    /** Normal Parameter. */
    csmParameterType_Normal = 0,

    /** Parameter for blend shape. */
    csmParameterType_BlendShape = 1
};

/** Parameter type. */
typedef int csmParameterType;
```

記載のある項目

パラメータの各要素を取得する

利用可能バージョン

4.2.02以上

csmGetParameterMinimumValues

パラメータの最小値だけが集まった配列へのアドレスを返します。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const float*
最小値が格納された配列へのアドレス

記載のある項目

パラメータの各要素を取得する

利用可能バージョン

3.0.00以上

csmGetParameterMaximumValues

パラメータの最大値だけが集まった配列へのアドレスを返します。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const float*
最大値が格納された配列へのアドレス

記載のある項目

パラメータの各要素を取得する

利用可能バージョン

3.0.00以上

csmGetParameterDefaultValues

パラメータの初期値だけが集まった配列へのアドレスを返します。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const float*
初期値が格納された配列へのアドレス

記載のある項目

パラメータの各要素を取得する
パラメータの操作

利用可能バージョン

3.0.00以上

csmGetParameterValues

パラメータの現在値だけが集まった配列へのアドレスを返します。
この配列へ書き込むことによってモデルを操作します。

引数

・csmModel* model
モデル構造体へのアドレス

返り値

・const float*
現在値が格納された配列へのアドレス

記載のある項目

パラメータの各要素を取得する
パラメータの操作

利用可能バージョン

3.0.00以上

csmGetParameterKeyCounts

パラメータに設定されたキーの数が集まった配列へのアドレスを返します。

引数

・csmModel* model
モデル構造体へのアドレス

返り値

・const int*
パラメータに設定されたキーの数が格納された配列へのアドレス

記載のある項目

パラメータのキーを取得する

利用可能バージョン

4.1.00以上

csmGetParameterKeyValues

パラメータに設定されたキーの位置が格納されたジャグ配列へのアドレスを返します。

引数

・csmModel* model
モデル構造体へのアドレス

返り値

・const float**
パラメータに設定されたキーの位置のジャグ配列へのアドレス

記載のある項目

パラメータのキーを取得する

利用可能バージョン

4.1.00以上

csmGetPartCount

モデルが持つパーツの個数を返します。
http://docs.live2d.com/cubism-editor-manual/parts/

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・int
パーツの個数

記載のある項目

なし

利用可能バージョン

3.0.00以上

csmGetPartIds

モデルが持つパーツのIDが格納された配列へのアドレスを返します。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const char**
文字列アドレスが格納される配列へのアドレス

記載のある項目

なし

利用可能バージョン

3.0.00以上

csmGetPartOpacities

モデルが持つパーツの不透明度の現在値が格納された配列へのアドレスを返します。

引数

・csmModel* model
モデル構造体へのアドレス

返り値

・float*
パーツ不透明度の配列のアドレス

記載のある項目

パーツの不透明度の操作

利用可能バージョン

3.0.00以上

csmGetPartParentPartIndices

モデルが持つパーツの親に当たるパーツの配列位置が格納された配列へのアドレスを返します。
Rootに所属するPartには-1が格納されます。

引数

・csmModel* model
モデル構造体へのアドレス

返り値

・const int*
パーツの親の番号が格納された配列へのアドレス

記載のある項目

パーツの親パーツを取得

利用可能バージョン

3.3.00以上

csmGetDrawableCount

モデルが持っているDrawableの数を返す

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・int
モデルが持っているDrawableの数

記載のある項目

Drawableの読み込みと配置
Drawableの描画順序をソートする

利用可能バージョン

3.0.00以上

csmGetDrawableIds

モデルが持っているDrawableのIDが入った配列へのアドレスを返します。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const char**
文字列アドレスが格納される配列へのアドレス

記載のある項目

Drawableの読み込みと配置

利用可能バージョン

3.0.00以上

csmGetDrawableConstantFlags

モデルが保有するDrawableの動的に変更されないフラグが入った配列へのアドレスを返します。
ここに記載されるフラグは
Drawableの描画合成に関するフラグと
・描画の加算
・描画の乗算
Drawableのカリングに関するフラグ
・両面描画
Drawableのマスクに関するフラグ**(4.0.0より追加)**
・マスクの反転
の4つの要素になります。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const csmFlags*
フラグへの配列のアドレス

```
snippet:
    /** Bitfield. */
    typedef unsigned char csmFlags;
```

記載のある項目

Drawableの読み込みと配置

利用可能バージョン

3.0.00以上

csmGetDrawableDynamicFlags

モデルが持つDrawableの描画時に更新されるフラグを格納した配列へのアドレスを返します。
描画時に更新されるフラグの内容は
・描画の可視性
・描画の可視性の変化
・不透明度の変化
・描画順の変化
・描画順序の入れ替えの有無
・頂点情報の更新の有無
の6つの要素になります。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const csmFlags*
フラグへの配列のアドレス

```
snippet:
    /** Bitfield. */
    typedef unsigned char csmFlags;
```

記載のある項目

Drawableの読み込みと配置
操作をモデルへ適用する
どの情報が更新されたか確認する
更新された頂点情報を取得する

利用可能バージョン

3.0.00以上

csmGetDrawableTextureIndices

モデルが保有するDrawableが参照するテクスチャの番号の入った配列のアドレスを返します。
テクスチャ番号とはアートメッシュが所属するテクスチャアトラスの番号のことです。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const int*
テクスチャ番号の入った配列のアドレス

記載のある項目

Drawableの読み込みと配置

利用可能バージョン

3.0.00以上

csmGetDrawableDrawOrders

モデルが保有するDrawableの描画順が入った配列へのアドレスを返します。
この値は現在のパラメータ値に基づき、補間計算された結果が格納されています。
なお、描画順グループの影響は無視されます。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const int*
描画順が入った配列へのアドレス

記載のある項目

Drawableの読み込みと配置
操作をモデルへ適用する
Drawableの描画順序をソートする

利用可能バージョン

3.0.00以上

csmGetDrawableRenderOrders

モデルが保有するDrawableの描画の順序が入った配列へのアドレスを返します。
Cubism Editorの表示と同様の順序になります。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const int*
描画順序が入った配列へのアドレス

記載のある項目

Drawableの読み込みと配置
操作をモデルへ適用する
Drawableの描画順序をソートする

利用可能バージョン

3.0.00以上

csmGetDrawableOpacities

モデルが保有するDrawableの不透明度の値が入った配列へのアドレスを返します。
値は0.0～1.0になります。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const float*
不透明度が入った配列へのアドレス

記載のある項目

Drawableの読み込みと配置
操作をモデルへ適用する
どの情報が更新されたか確認する
更新された頂点情報を取得する

利用可能バージョン

3.0.00以上

csmGetDrawableMaskCounts

モデルが保有するDrawableのマスクの数が格納された配列へのアドレスを返します。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const int*
マスクの数が入った配列へのアドレス

記載のある項目

Drawableの読み込みと配置
描画時にマスクを適用する

利用可能バージョン

3.0.00以上

csmGetDrawableMasks

モデルが保有するDrawableのマスクのDrawable番号が格納されたジャグ配列のアドレスを返します。
csmGetDrawableMaskCountsで0の部分にも他のDrawableのマスクで使用するアドレス情報が入るので取り扱いに注意してください。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const int**
マスクの参照番号の入ったアドレスの配列へのアドレス

記載のある項目

Drawableの読み込みと配置
描画時にマスクを適用する

利用可能バージョン

3.0.00以上

csmGetDrawableVertexCounts

モデルが保有するDrawableの頂点の数が入った配列へのアドレスを返します。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const int*
Drawableの頂点の個数が入った配列へのアドレス

記載のある項目

Drawableの読み込みと配置
描画時にマスクを適用する

利用可能バージョン

3.0.00以上

csmGetDrawableVertexPositions

モデルが保有するDrawableの頂点が格納されたジャグ配列へのアドレスを返します。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const csmVector2**
頂点情報へのジャグ配列へのアドレス

```
 snippet:
/** 2 component vector. */
    typedef struct
    {
        /** First component. */
        float X;

        /** Second component. */
        float Y;
    }
    csmVector2;
```

記載のある項目

Drawableの読み込みと配置
操作をモデルへ適用する
どの情報が更新されたか確認する
更新された頂点情報を取得する

利用可能バージョン

3.0.00以上

csmGetDrawableVertexUvs

モデルが保有するDrawableのUV情報が入ったジャグ配列へのアドレスを返します。
頂点ごとに対応するので個数についてはcsmGetDrawableVertexCountsで取得します。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const csmVector2**
頂点情報へのジャグ配列へのアドレス

記載のある項目

Drawableの読み込みと配置

利用可能バージョン

3.0.00以上

csmGetDrawableIndexCounts

モデルが保有するDrawableの頂点に対するポリゴンの対応番号配列のサイズを格納した配列のアドレスを返します。
3角形の各角がどの頂点になるかを記述した配列になるため、
この配列に格納される値は必ず0か3の倍数になります。
スキニングの末端などで個数が0になることに注意してください。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const int*
ポリゴンの対応番号配列のサイズを格納した配列のアドレス

記載のある項目

Drawableの読み込みと配置

利用可能バージョン

3.0.00以上

csmGetDrawableIndices

モデルが保有するDrawableの頂点に対するポリゴンの対応番号のジャグ配列へのアドレスを返します。
格納されている番号は各Drawableで独立のものです。
csmGetDrawableIndexCountsで0の部分にも他のDrawableで使用する配列へのアドレス情報が入るので取り扱いに注意してください。

引数

・const csmModel* model
モデル構造体へのアドレス

返り値

・const unsigned short**
対応番号のジャグ配列へのアドレス

記載のある項目

Drawableの読み込みと配置

利用可能バージョン

3.0.00以上

csmResetDrawableDynamicFlags

次回のcsmUpdateModelの時に、csmGetDrawableDynamicFlagsで得られる情報を最新にするため、
すべてのフラグを下す処理をします。
呼び出しのタイミングは描画の処理が終わった後に呼びます。

引数

・csmModel* model
モデル構造体へのアドレス

返り値

なし

記載のある項目

DynamicFlagのリセット

利用可能バージョン

3.0.00以上

csmGetDrawableMultiplyColors

アートメッシュの乗算色の配列へのアドレスを返します。

引数

・csmModel* model
モデル構造体へのアドレス

返り値

・const csmVector4*
アートメッシュの乗算色のRGBA値が格納された配列へのアドレス
XがR、YがG、ZがBに対応（Wの値は現在未使用）

```
 snippet:
/** 4 component vector. */
    typedef struct
    {
        /** First component. */
        float X;

        /** Second component. */
        float Y;

        /** Third component. */
        float Z;

        /** Fourth component. */
        float W;
    }
    csmVector4;
```

記載のある項目

乗算色、スクリーン色を取得する

利用可能バージョン

4.2.00以上

csmGetDrawableScreenColors

アートメッシュのスクリーン色の配列へのアドレスを返します。

引数

・csmModel* model
モデル構造体へのアドレス

返り値

・const csmVector4*
アートメッシュのスクリーン色のRGBA値が格納された配列へのアドレス
XがR、YがG、ZがBに対応（Wの値は現在未使用）

```
snippet:
  /** 4 component vector. */
  typedef struct
  {
      /** 1st component. */
      float X;

      /** 2nd component. */
      float Y;

      /** 3rd component. */
      float Z;

      /** 4th component. */
      float W;
  } csmVector4;
```

記載のある項目

乗算色、スクリーン色を取得する

利用可能バージョン

4.2.00以上

csmGetDrawableParentPartIndices

Drawableの親のパーツの配列位置が格納された配列へのアドレスを返します。
Rootに所属するDrawableには-1が格納されます。

引数

・csmModel* model
モデル構造体へのアドレス

返り値

・const int*
Drawableの親の番号が格納された配列へのアドレス

記載のある項目

Drawableの親パーツを取得

利用可能バージョン

4.2.02以上

csmHasMocConsistency

.moc3ファイルの整合性を確認します。
メモリはアライメントされたものを用意する必要があります。

引数

・void* address
.moc3が読まれたデータ配列の頭のアドレス
アライメントする必要があります。

・const unsigned int size
.moc3が読まれたデータ配列の長さ

返り値

・int
.moc3の整合性
読み込んだ.moc3が有効であれば'1'、そうでなければ'0'になる。

記載のある項目

moc3の整合性の確認

利用可能バージョン

4.2.03以上