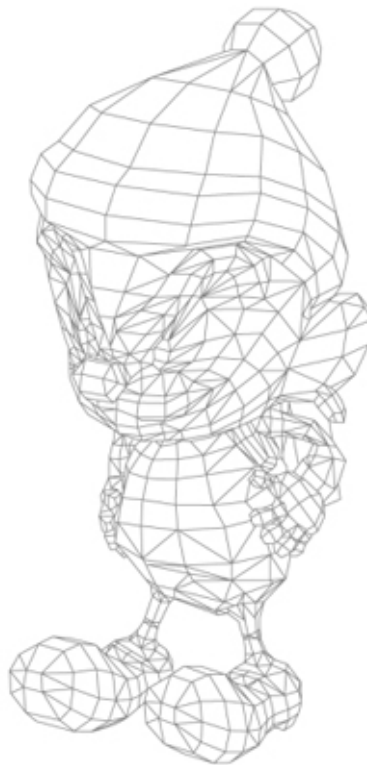

MascotCapsuleV3

Tutorial Document for BREW

日本語版



Ver.2.0

—更新履歴—

2006年7月28日

・ Ver. 1.0.0 リリース

更新内容

- 新規作成

2006年11月27日

・ Ver. 2.0 リリース

更新内容

- リニューアルしました。

目次

はじめに.....	1
チュートリアルの目的	1
サンプルのウォークスルー	1
ディレクトリ/ファイル構成	1
Step1 – 3D モデルの表示とテクスチャの利用	3
Step1-1 オブジェクト類の初期化 MascotCapsule_InitAppData().....	3
MascotCapsule オブジェクトの作成.....	3
アロケータの初期化	3
Figure オブジェクトの初期化.....	4
Texture オブジェクトの初期化.....	4
Render オブジェクトの初期化.....	4
Render オブジェクトの初期化.....	5
Step1-2 視点位置の決定 MascotCapsule_TransCamera().....	6
視点の位置.....	6
視線の向き.....	6
視点の上方向.....	7
視点の有効範囲.....	8
Step1-3 オブジェクトの描画 MascotCapsule_Draw().....	9
1,テクスチャを Render にセット	9
2,figure を描画登録.....	9
3,描画内容を VRAM に flush()	9
Step1-4 オブジェクトの終了処理 MascotCapsule_FreeAppData()	10
Step2 – 視点の平行移動	11
ズームイン/ズームアウト.....	11
左右移動	11
上下移動	11
設定したパラメータの反映	12
Step3 – モデルの拡大、縮小	13
行列の扱い.....	13
単位行列	13
4096 と単位行列について	13
拡大	14
縮小	14
Step4 – 照明の利用	15
ライトの設定	15
属性の取得と設定	15
環境光の設定	15
平行光源の設定.....	16
モデルデータの属性	16
Step5 – 平行投影と透視投影	17
平行投影と透視投影の切替	17

Step6 – 視点の回転	18
回転行列の取得	18
[回転行列のイメージ図].....	18
[X 方向への回転].....	18
[Y 方向への回転].....	18
回転行列と視点行列の演算	19
Step7 – アニメーション	20
アニメーションの初期化、読み込み	20
ActTable オブジェクトと Figure オブジェクトの関連付け.....	20
MascotCapsule で使用するデータ	20
フレーム毎のアニメーション処理	21
Step8 – 複数アニメーション	22
アニメーションの初期化、読み込み	22
アニメーションの切替	23
Step9 – 複数 3D モデル	24
Figure の初期化、MBAC の読み込み	24
Figure の描画	24
Step10 – プリミティブの描画	25
データを用意する	25
頂点情報	25
法線情報	25
テクスチャ位置情報	26
色情報	26
描画する	26

はじめに

チュートリアルの目的

本チュートリアルは、MascotCapsule アプリケーションを作成する際の手順や、作成する際の注意点を実践的に学ぶことを目的としています。

3D モデルを表示するところから開始し、カメラ、アニメーション、ライトなどの使い方を小さなサンプルを通して理解することができます。

なお、本チュートリアルでは、アプリケーション作成の手順を重視していますので、MascotCapsule の仕様や API の詳細については解説いたしません。これらの情報については弊社で提供しております、プログラマーズマニュアル、リファレンスマニュアルをあわせて参照して頂くようお願い致します。

サンプルのウォークスルー

このステップでは、本チュートリアルで使用する sample プログラムの基本的な内容について取り上げます。なお、BREW プログラミングの知識、C 言語など前提となる知識については本チュートリアルでは解説を行いませんので、別途情報を入手してください。

ディレクトリ/ファイル構成

/sample_xx	/allocator.c(h)	メモリアロケータの実装です。
	/FileIStream.c(h)	ファイルストリームインターフェイスの実装です。
	/sample.c(h)	サンプルプログラムのメインとなるファイルです。
	/include/	MascotCapsuleV3 に必要となるヘッダーファイル群です。
	/data/	サンプルで使用するデータ群です。

data ディレクトリにあるデータの解説

ファイル	説明
*.tra	アクションデータが格納されています。市販のモデリングツールから Plugin を使って出力します。テキストファイルです。
*.mtra	上記、TRA ファイルをコンバータによりバイナリ化したファイルです。アプリケーション動作時にはバイナリ化されている必要があります。
*.bac	モデルデータが格納されています。市販のモデリングツールから Plugin を使って出力します。テキストファイルです。
*.mbac	上記、BAC ファイルをコンバータによりバイナリ化したファイルです。アプリケーション動作時にはバイナリ化されている必要があります。
*.bmp	モデルに貼り付ける Texture や、sphere マップなどの画像ファイルです。

※より詳しい解説は弊社プログラマーズマニュアルを参照してください。

sample.c の構成

本チュートリアルでは理解をしやすいするために、ほとんどの内容を sample.c および sample.h にまとめています。sample.c には BREW アプリケーションが必須とする処理なども含まれています。ここでは、関数単位でこれらについて簡単に解説します。

[System 系関数]

関数名の先頭に” System_” というプレフィックスが付いているものは、BREW のシステムが必要としている処理群であり、MascotCapsule に限らず必要となる処理です。

System_HandleEvent()	イベントハンドラ。イベントに関する処理を行います。
System_FrameProc()	フレームプロシージャ。毎フレーム行いたい処理を記述します。
System_KeyProc()	キープロシージャ。キー入力を判別します。
System_Suspend()	サスペンドハンドラ。端末がサスペンド(休止)した際の処理を記述。
System_Resume()	レジュームハンドラ。端末がレジューム(再開)した際の処理を記述。

[MascotCapsule 系関数]

関数名の先頭に” MascotCapsule_” というプレフィックスが付いているものは、MascotCapsule で必要となる処理です。

MascotCapsule_InitAppData()	MascotCapsule で使用するデータ類を初期化します。
MascotCapsule_FreeAppData()	MascotCapsule で使用するデータ類を終了処理します。
MascotCapsule_Draw()	3D オブジェクトを描画します。
MascotCapsule_Animate()	3D オブジェクトのアニメーション更新を行います。
MascotCapsule_KeyCheck()	入力されたキーに応じたパラメータの変更を行います。
MascotCapsule_TransCamera()	視点の設定や位置の更新処理を行います。
MascotCapsule_DrawPrimitives()	プリミティブ描画を行います。Step10 で必要となります。

Step1 ・ 3D モデルの表示とテクスチャの利用

Step1 では単純に 3D キャラクタの表示を行います。
 ただ 3D オブジェクトを表示するだけでも、必ず以下の手順を踏む必要があります。

Step1-1,オブジェクト類の初期化、データの読み込み

Step1-2,視点位置の決定

Step1-3,オブジェクトの描画

Step1-4,オブジェクトの終了処理

Step1-1 オブジェクト類の初期化 MascotCapsule_InitAppData()

MascotCapsule オブジェクトの初期化やデータの読み込みは、MascotCapsule_InitAppData()関数内で行われています。

MascotCapsule オブジェクトの作成

MascotCapsule の機能を利用するためには、まず MascotCapsule オブジェクトを作成する必要があります。作成には BREW API の ISHELL_CreateInstance()を使用します。

```
pMe->mpMicro3D = NULL;
if (ISHELL_CreateInstance(pMe->aeeApplet.m_plShell, AEECLSID_IM3D_B31,
                          (void **)&pMe->mpMicro3D))
    return FALSE;
```

MascotCapsule のオブジェクトの多くは初期化が必要です。
 初期化は、それぞれのオブジェクト用に用意された_Initialize()関数を使用します。

アロケータの初期化

アロケータはメモリのアロケートを行うためのものです。多くの MascotCapsule オブジェクトは初期化の際にメモリを確保します。そのため、メモリ確保を行うアロケータを引数として渡す必要があります。

```
Allocator_initialize( &pMe->allocator, pApp->m_plShell, pMe );
ialloc = Allocator_getAllocator( &pMe->allocator );
```

Figure オブジェクトの初期化

Figure オブジェクトは 3D オブジェクトの外観(形)を扱います。

Figure にはモデルデータ(MBAC)が対応します。この 2 つの関連付けはデータ読み込み時に行われます。

```
IMICRO3D_Figure_initialize( pMe->mpMicro3D, &pMe->figure, ialloc );
...
Filestream_initialize( &fis, pFileMgr, pFile, pMe);
rval = IMICRO3D_Figure_loadMbacData( pMe->mpMicro3D, &pMe->figure,
                                     Filestream_getIstream( &fis ) );
```

Texture オブジェクトの初期化

Texture オブジェクトは画像データ(bmp)と対応します。この 2 つの関連付けもデータ読み込み時に行われます。MascotCapsule で扱える画像のサイズは 256*256 が最大サイズです。

```
IMICRO3D_Texture_initialize( pMe->mpMicro3D, &pMe->texture[0], ialloc );
IMICRO3D_Texture_initialize( pMe->mpMicro3D, &pMe->texture[1], ialloc );
.....
Filestream_initialize( &fis, pFileMgr, pFile, pMe );
rval = IMICRO3D_Texture_loadBmpData( pMe->mpMicro3D, &pMe->texture[i],
                                     Filestream_getIstream( &fis ),TEXTURE_TYPE_NORMAL );
```

Render オブジェクトの初期化

Render オブジェクトは 3D 描画全般に必要なオブジェクトです。ほとんどの描画系 API のコールの際に Render オブジェクトを引数として必要とします。Render オブジェクトの初期化時には、VRAM の設定、クリッピング領域の指定、スクリーンセンターの指定も行います。

Render オブジェクトの初期化

```
IMICRO3D_Render_initialize( pMe->mpMicro3D, &pMe->render, ialloc );
```

[Render オブジェクトと VRAM の関連付け]

BREW の場合、通常は DIB が VRAM となります。

VRAM を取得するには BREW API の IDISPLAY_GetDeviceBitmap() を利用します。

```
// Get the screen bitmap
IDISPLAY_GetDeviceBitmap(pApp->m_pIDisplay, &pMe->pBitmap);

// Get AEEDIB in order to access frame buffer
IBITMAP_QueryInterface(pMe->pBitmap, AEECLSID_DIB, (void **)&pMe->pAEEDIB);
```

IMICRO3D_Render_setVram() は VRAM 領域(描画領域)の指定をします。

以降、ここで指定された領域に 3D 描画を行うようになります。

```
IMICRO3D_Render_setVram( pMe->mpMicro3D, &pMe->render,
                        GetBitmapWidth( pMe ),
                        GetBitmapHeight( pMe ),
                        GetBitmapPitch( pMe ),
                        (pMe->pAEEDIB->pBmp));
```

[クリッピング領域の指定]

クリッピング領域とは描画する画面の範囲(大きさ)の指定です。

```
IMICRO3D_Render_setClipRect( pMe->mpMicro3D, &pMe->render, 0, 0,
                             GetBitmapWidth( pMe ), GetBitmapHeight( pMe ) );
```

[スクリーンセンターの決定]

スクリーンの中心を指定します。通常は縦、横をそれぞれ 2 で割った数値になるでしょう。

```
IMICRO3D_Render_setScreenCenter( pMe->mpMicro3D, &pMe->render,
                                  (GetBitmapWidth( pMe ) >> 1), (GetBitmapHeight( pMe ) >> 1) );
```

Step1-2 視点位置の決定 MascotCapsule_TransCamera()

3D を扱う上で視点は非常に重要な意味を持っています。視点とはカメラと置き換えて考えることができます。つまり、この視点から見た内容が最終的に画面上に描画されることになります。(MascotCapsule では厳密にはカメラというオブジェクトはありません。)
視点の場所と位置は MascotCapsule_InitAppData()内で次のように指定しています。

視点の位置

MascotCapsule の座標系は右手系です。今回は、視点の位置はスクリーンを正面にした場合、x は横方向、y は縦方向、z は奥行きとしています。z は手前が+方向、奥が-方向となります。(下図参照)

```
pMe->cam_frame.pos.x = 0;  
pMe->cam_frame.pos.y = 4096;  
pMe->cam_frame.pos.z = 4096 * 4;
```

視線の向き

上記、視点の位置で視点は手前側に設置されているので、奥側(-方向)を向くように調整しています。また、視点位置が y 方向にプラスされているため、視線はやや下向き(-方向)にしています。

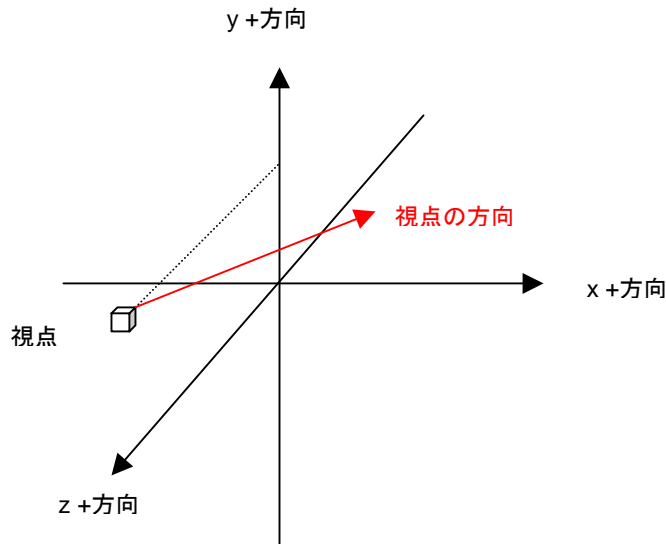
```
pMe->cam_frame.dir.x = 0;  
pMe->cam_frame.dir.y = -4096 / 4;  
pMe->cam_frame.dir.z = -4096;
```

視点の上方向

このパラメータはカメラのどちらが上を向くか、ですが、通常は y 方向に 4096 を指定します。

```
pMe->cam_frame.up.x = 0;
pMe->cam_frame.up.y = 4096;
pMe->cam_frame.up.z = 0;
```

[視点のイメージ]



MascotCapsule_TransCamera()内では以下のように現在の視点変換行列を pMe->view_trans に対して設定しています。(pMe->view_trans はいわばカメラ位置となります)

```
IMICRO3D_Atrans3i_setViewTrans( pMe->mpMicro3D,
&pMe->view_trans, &pMe->cam_frame.pos,
&pMe->cam_frame.dir, &pMe->cam_frame.up );
```

この視点情報 view_trans を Render にセットします。これは Render に対して、「この位置から見た 3D 描画するように」という指示をしていると考えられます。より詳細に言うならば、内部では視点変換行列を元にしたアフィン変換が行われています。

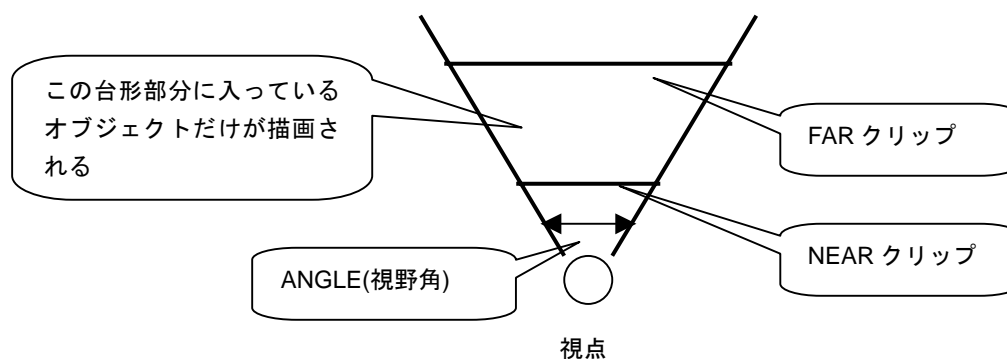
```
IMICRO3D_Render_setViewTrans( pMe->mpMicro3D, &pMe->render, &pMe->view_trans );
```

視点の有効範囲

視点の位置の他、その有効範囲を指定する必要があります。有効範囲とは、視点からどのくらいの奥行きが見えるのか、どのくらいの視野角度とするのか、などです。有効範囲の指定の仕方にはいくつか方法がありますが、今回のサンプルでは NEAR,FAR,ANGLE による指定を行っています。

```
pMe->cam_frame.z_near = NEAR_CLIP;
pMe->cam_frame.z_far = FAR_CLIP;
pMe->cam_frame.angle = ANGLE;
```

[NEAR クリップと FAR クリップのイメージ図]



パラメータのセットは MascotCapsule_TransCamera()内で行っています。

```
IMICRO3D_Render_setPerspectiveFov( pMe->mpMicro3D,
&pMe->render,pMe->cam_frame.z_near,
pMe->cam_frame.z_far, pMe->cam_frame.angle );
```

Step1-3 オブジェクトの描画 MascotCapsule_Draw()

描画に関しては特に難しいことはありません。手順としては、次のようになります。

- 1,テクスチャを Render にセット
- 2,figure を描画登録
- 3,描画内容を VRAM に flush()

1,テクスチャを Render にセット

Texture を Render にセットする作業は毎回必要ですので注意してください。

(これは言い換えれば、1つのモデルに対して、様々なテクスチャを貼り付ける事ができるという事です。)

```
IMICRO3D_Render_setTexture( pMe->mpMicro3D, &pMe->render, &pMe->texture[0] );
```

2,figure を描画登録

figure を IMICRO3D_Render_drawFigure()で描画登録します。

あくまで描画登録であり、この段階ではまだ実際に描画はされません。

```
if( !IMICRO3D_Render_drawFigure( pMe->mpMicro3D, &pMe->render, &pMe->figure ))
    DBGPRINTF("draw failed");
```

3,描画内容を VRAM に flush()

描画登録されているものを実際に VRAM に flush()します。

```
IMICRO3D_Render_flush(pMe->mpMicro3D,&pMe->render);
```

※Render_flush()後の処理としては IDISPLAY_UpdateEx()などプラットフォーム固有のアップデート処理をする必要があります。

Step1-4 オブジェクトの終了処理 MascotCapsule_FreeAppData()

MascotCapsule_InitAppData()で作成した MascotCapsule オブジェクト群を終了処理します。オブジェクトの終了処理は MascotCapsule_FreeAppData()で行っています。

終了処理は、それぞれのオブジェクト用に用意された_finalize()関数を使用します。ただし、MascotCapsule のオブジェクトはそれぞれ参照カウントを内部保持しており、このカウントが0にならないとオブジェクトを終了できません。

詳しくはプログラマーズマニュアルに記載してありますが、原則として Render オブジェクトを先に解放(finalize)すれば問題ないでしょう。

そして、最後に MascotCapsule オブジェクトを IMICRO3D_Release()で終了するようにしてください。

```
IMICRO3D_Render_finalize( pMe->mpMicro3D, &pMe->render );
IMICRO3D_Texture_finalize( pMe->mpMicro3D, &pMe->texture[0] );
IMICRO3D_Texture_finalize( pMe->mpMicro3D, &pMe->texture[1] );
IMICRO3D_Figure_finalize( pMe->mpMicro3D, &pMe->figure );
Allocator_finalize( &pMe->allocator );
...
if (pMe->mpMicro3D != NULL) {
    IMICRO3D_Release(pMe->mpMicro3D);
    pMe->mpMicro3D = NULL;
}
```

Step2 ・ 視点の平行移動

このステップでは視点(カメラ)の平行移動を行いません。ユーザーからのキー入力に応じて視点の座標を変更します。

キー入力のチェックは MascotCapsule_KeyCheck()で行っています。System_KeyProc()で設定された mask 値をチェックして、その mask に応じたパラメータの変更を行います。

ズームイン/ズームアウト

今回の場合、ズームイン/ズームアウトは視点座標の z 値の増減により実現します。

```
if ( (mask & ZIN_MASK) != 0 ) {
    pMe->cam_frame.pos.z -= 256;
}
else if ( (mask & ZOUT_MASK) != 0 ) {
    pMe->cam_frame.pos.z += 256;
}
```

左右移動

今回の場合、左右移動は視点座標の x 値の増減により実現します。

```
if ( (mask & LEFT_MASK) != 0 ) {
    pMe->cam_frame.pos.x -= 256;
}
else if ( (mask & RIGHT_MASK) != 0 ) {
    pMe->cam_frame.pos.x += 256;
}
```

上下移動

今回の場合、上下移動は視点座標の y 値の増減により実現します。

```
if ( (mask & UP_MASK) != 0 ) {
    pMe->cam_frame.pos.y += 256;
}
else if ( (mask & DOWN_MASK) != 0 ) {
    pMe->cam_frame.pos.y -= 256;
}
```

今回は x,y,z 全ての平行移動は 256 という単位で行っていますが、この数値はモデルの大きさや、スケール、移動速度などにより異なりますので、コンテンツに応じて調整する必要があります。

設定したパラメータの反映

設定したパラメータは Step1-2 の MascotCapsule_TransCamera()内で行われます。

```
IMICRO3D_Render_setPerspectiveFov( pMe->mpMicro3D,  
&pMe->render,pMe->cam_frame.z_near,  
pMe->cam_frame.z_far, pMe->cam_frame.angle );
```

今回は視点を動かしていますが、モデルを動かすこともできます。

視点を動かした場合とモデルを動かした場合の考え方は基本的には一緒です。

視点を動かすべきか、それともモデルを動かすべきかは作成するコンテンツに依存します。

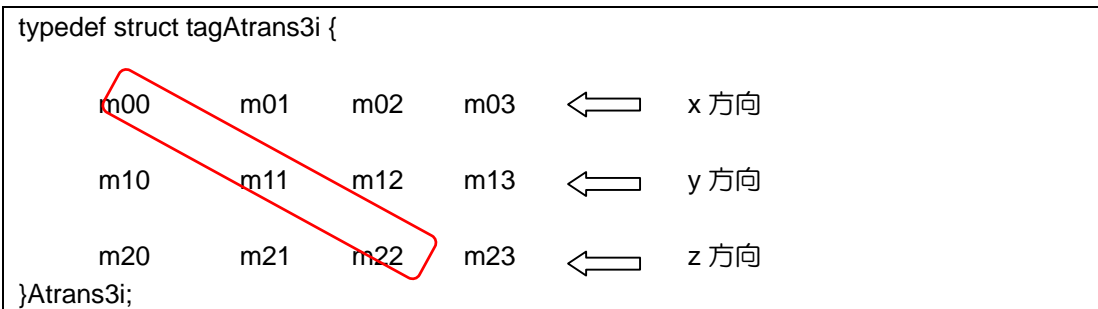
Step3 ・ モデルの拡大、縮小

MascotCapsule においてはモデルを拡大、縮小するための API というのは存在していません。プログラマーが行列演算を行うことで、これを実現します。

行列の扱い

MascotCapsule では行列を扱うためのクラス、Atrans3i が用意されています。Atrans3i は hi_sint32 型が 12 個集まった構造体として定義されています。この 12 個のうち、m00,m11,m22 の要素が拡大、縮小に影響を与えるパラメータとなります。

m00 は z 方向への拡大、縮小、m11 は y 方向への拡大、縮小、m22 は z 方向への拡大、縮小のパラメータとなります。



単位行列

まずは宣言した行列 Atrans3i ex_trans に対して単位行列をセットします。単位行列とは行列における”1”のようなもので、Atrans3i に対し、次の値をセットするということです。

```
Atrans3i ex_trans = { 4096,0,0,0,
                    0,4096,0,0,
                    0,0,4096,0 };
```

単位行列の設定は頻繁に行われるので API が用意されています。

```
IMICRO3D_Atrans3i_setIdentity(pMe->mpMicro3D, &ex_trans );
```

4096 と単位行列について

プログラム中で 4096 という値を指定することが多いのは、MascotCapsule が浮動小数を使わずに、固定小数を使用しているためです。固定小数を使用することで、演算の高速化を図ることができるからです。固定小数は次のような定義になっています。

整数部 20bit	小数部 12bit
-----------	-----------

すなわち、4096 とは次のように MascotCapsule 内では”1”を意味します。

1	0000 0000 0000
---	----------------

拡大

単位行列とはいわば等倍ですので、拡大したい場合は x,y,z 方向へそれぞれ 2 倍します。

```
ex_trans.m00 = 4096 * 2;    //    X 方向へ 2 倍の拡大
ex_trans.m11 = 4096 * 2;    //    Y 方向へ 2 倍の拡大
ex_trans.m22 = 4096 * 2;    //    Z 方向へ 2 倍の拡大
```

作成した行列 ex_trans と視点行列 &pMe->view_trans を演算します。

行列の演算には IMICRO3D_Atrans3i_multiply2() を使用します。

```
IMICRO3D_Atrans3i_multiply2(pMe->mpMicro3D,&pMe->view_trans,
&pMe->view_trans, &ex_trans);
```

演算の結果は pMe->view_trans に格納されているので、これを setViewTrans() します。

```
IMICRO3D_Render_setViewTrans( pMe->mpMicro3D, &pMe->render, &pMe->view_trans );
```

結果として、モデルが拡大されているのがわかります。

縮小

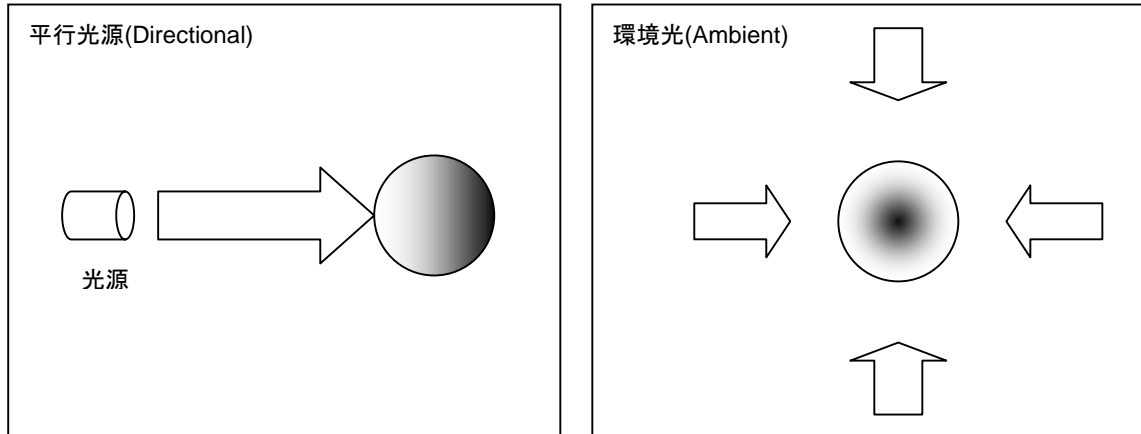
縮小を行ないたい場合は、拡大とは逆に 4096 以下の値を指定します。

例えば半分のサイズに縮小したい場合は 4096 / 2 となります。

```
ex_trans.m00 = 4096 / 2;    //    X 方向へ 2 分の 1
ex_trans.m11 = 4096 / 2;    //    Y 方向へ 2 分の 1
ex_trans.m22 = 4096 / 2;    //    Z 方向へ 2 分の 1
```

Step4 ・ 照明の利用

MascotCapsule では照明として、白色平行光源、環境光の2つが使用可能です(白色のみ)。平行光源は一つの方向からの照明となり、環境光はあらゆる方向から、等しく照らす照明となります。



ライトの設定

ライトについては初期化時などに一度設定すればアプリケーション終了時まで有効となります。サンプルプログラムでは MascotCapsule_InitAppData()内で行っています。

属性の取得と設定

MascotCapsule では Render に設定された「属性」によって描画効果を設定します。(マテリアルという概念はありません)

現在 Render に設定されている属性を取得し、

```
attr = IMICRO3D_Render_getAttribute(pMe->mpMicro3D, &pMe->render);
```

新たに属性を OR を使って付加します。

その他の属性についてはプログラマーズマニュアル等を参照してください。

```
IMICRO3D_Render_setAttribute(pMe->mpMicro3D, &pMe->render, attr | M3D_LIGHTING);
```

環境光の設定

環境光は全方向からの照明となりますので、強度のみを指定します。

数値が大きいほど光の強度が強くなります。

```
IMICRO3D_Render_setAmbientLight(pMe->mpMicro3D, &pMe->render, 4096 * 50 / 100);
```

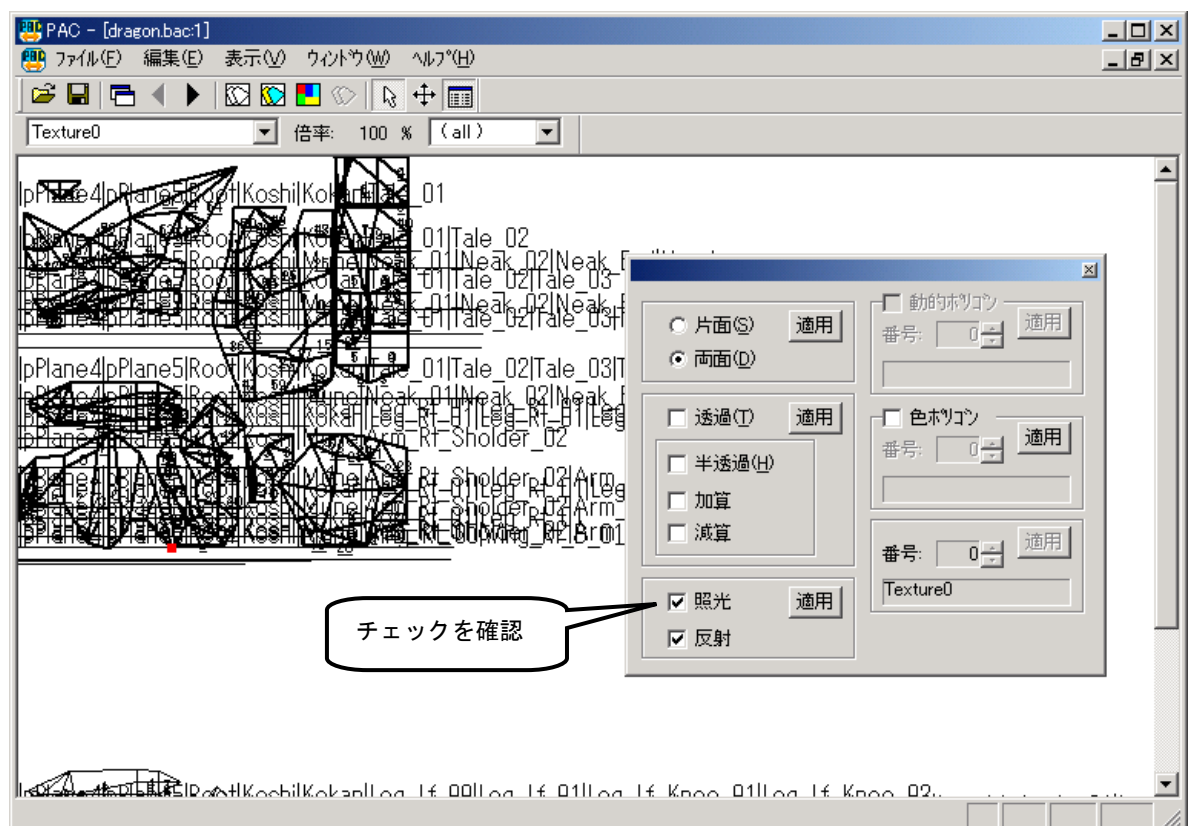
平行光源の設定

平行光についてはその向きが重要となりますので、強度の他に向き(ベクトル)を引数とします。

```
Vec3i light = {2,2,0};
.....
IMICRO3D_Render_setDirectionLight(pMe->mpMicro3D, &pMe->render, &light,
4096 * 50 / 100);
```

モデルデータの属性

上記の手順で通常は照明が有効となりますが、もし有効にならない場合は、PAC を使って、モデルデータの属性を確認してみてください。「照光」にチェックが入っていない場合は、各ポリゴンに対して「照光」にチェックを入れてください。



Step5 ・ 平行投影と透視投影

MascotCapsule では平行投影と透視投影という 2 つの投影形態が選択可能です。透視投影とは描画オブジェクトと視点の位置関係で手前のものほど大きく描画され、奥にあるものは小さく表示される、という投影形式です(遠近法)。対する平行投影は視点の位置に関係なく描画されます。

[透視投影]



[平行投影]



平行投影と透視投影の切替

投影方法は Render に対してセットします。この際に呼び出す API によって投影方法が決定します。サンプルプログラムでは” 7 ” キーを押した際に pMe->isPerspective フラグを TRUE/FALSE することで、二つの投影方法をダイナミックに切り替えるように作成しています。投影方法の切替はサンプル中では MascotCapsule_TransCamera()内で行っています。

透視投影を行う場合は IMICRO3D_Render_setPerspectiveFov()をコールします。

```
if(pMe->isPerspective){
    pMe->view_trans.m23 = 0;
    IMICRO3D_Render_setPerspectiveFov( pMe->mpMicro3D, &pMe->render,
    pMe->cam_frame.z_near,          pMe->cam_frame.z_far,          pMe->cam_frame.angle );
```

平行投影を行う場合は、IMICRO3D_Render_setOrthographicW()をコールします。この際に視点行列の m23 パラメータに値をセットしているのは、投影方法の変化に伴い z の位置を調整するためです。

```
}else {
    pMe->view_trans.m23 = 1024;
    IMICRO3D_Render_setOrthographicW( pMe->mpMicro3D, &pMe->render, 1024 );
}
```

投影方法を指定する API はこの 2 つ以外にも複数用意されています。基本的な機能は変わりませんが、必要とする引数が異なります。詳しくはリファレンスマニュアルを参照してください。

Step6 ・ 視点の回転

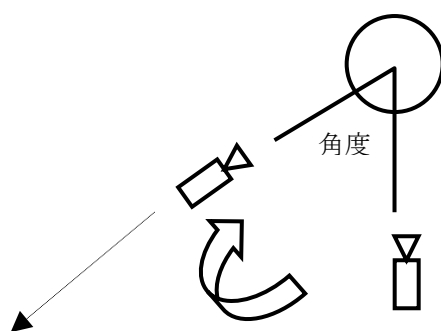
Step2 では視点の平行移動を行いました。このステップでは視点の回転について取り上げます。平行移動については単純に x,y,z の値の増減で行うことができましたが、回転になると、sin,cos を使った回転行列の計算が複雑になりますので、API によるサポートがあります。

回転行列の取得

回転行列とはいわば、回転角度に対する移動力です。この回転行列と回転前の視点行列を掛け合わせることで回転後の視点の位置を求めることができます。

回転行列は角度を元に setRotateX(),setRotateY()などの API で取得できます。

[回転行列のイメージ図]



$$\text{回転後の視点変換行列} = \text{回転行列} * \text{回転前の視点変換行列}$$

今回のプログラムでは2,4,6,8のキーで角度 pMe->x_angle,pMe->y_angleが増減するようになっています。

[X 方向への回転]

```
IMICRO3D_Atrans3i_setIdentity(pMe->mpMicro3D, &cam_xtrans );
...
IMICRO3D_Atrans3i_setRotateX( pMe->mpMicro3D, &cam_xtrans, pMe->x_angle);
```

[Y 方向への回転]

```
IMICRO3D_Atrans3i_setIdentity(pMe->mpMicro3D, &cam_vtrans );
...
IMICRO3D_Atrans3i_setRotateY( pMe->mpMicro3D, &cam_vtrans, pMe->y_angle);
```

回転行列と視点行列の演算

行列同士の演算には IMICRO3D_Atrans3i_multiply2() を利用します。
この multiply2 以外にもいくつかの行列演算用の API が用意されています。
最終的に求められた行列 view_trans を Render にセットします。

```
IMICRO3D_Atrans3i_multiply2(pMe->mpMicro3D,&pMe->view_trans,  
                           &pMe->view_trans, &cam_xtrans);  
IMICRO3D_Atrans3i_multiply2(pMe->mpMicro3D,&pMe->view_trans,  
                           &pMe->view_trans, &cam_vtrans);  
  
//      アフィン変換  
IMICRO3D_Render_setViewTrans( pMe->mpMicro3D, &pMe->render, &pMe->view_trans );
```

Step7 ・ アニメーション

MascotCapsule では簡単にモデルのアニメーションが実現できます。
アニメーションは MTRA 形式としてあらかじめ用意しておく必要があります。
読み込んだ MTRA ファイルは ActTable オブジェクトとして扱われ、プログラム中ではこの ActTable オブジェクトを通じてアニメーション操作が可能になります。

アニメーションの初期化、読み込み

まずはファイルをオープンします。

```
pIFile = IFILEMGR_OpenFile( pIFileMgr, pMe->mbactraFile[3], _OFM_READ );
if ( !pIFile )
    return FALSE;
```

ファイルを読み込みます。

この際に ActTable act_wait と MTRA データの関連付けが行われています。

```
Filestream_initialize( &fis, pIFileMgr, pIFile, pMe );
rval = IMICRO3D_ActTable_loadMtraData( pMe->mpMicro3D,
                                        &pMe->act_wait, Filestream_getIstream( &fis ) );
```

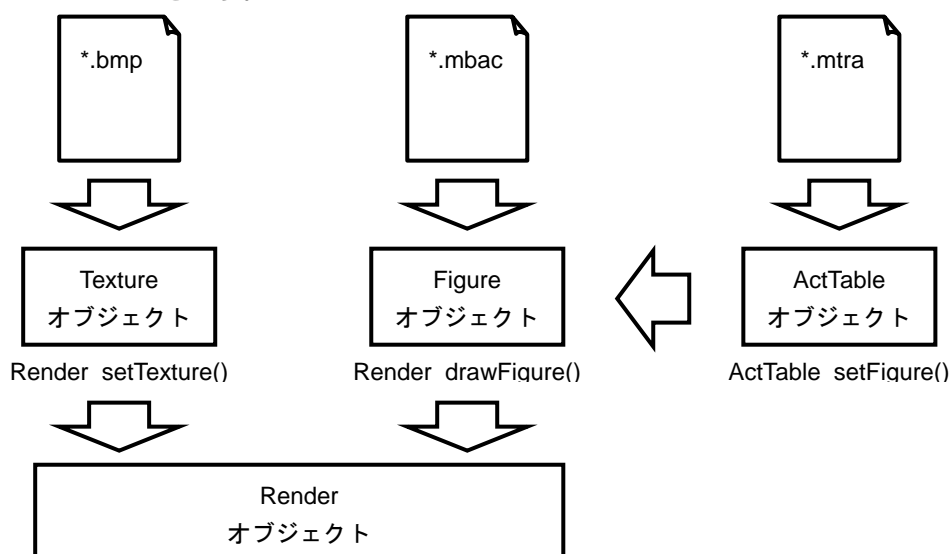
ActTable オブジェクトと Figure オブジェクトの関連付け

MTRA ファイルを元に作成された ActTable オブジェクトをアニメーションさせる Figure(モデル) と関連付けます。

```
IMICRO3D_ActTable_setFigure( pMe->mpMicro3D, &pMe->act_wait, &pMe->figure );
```

MascotCapsule で使用するデータ

ここまでで MascotCapsule で扱うデータオブジェクトが各種揃いました。
ここでまとめておきます。



フレーム毎のアニメーション処理

アニメーションはモデルの動きを変えつつ、毎フレーム更新されなくてはなりません。今回のサンプルプログラムでは System_FrameProc()内でアニメーションを処理するための MascotCapsule_Animate()をコールしています。

まずはアニメーションの長さを取得します。結果を剰余算することで、アニメーションが一周すると、また 0 フレーム目から繰り返し再開するように作っています。

```
pMe->act_frame %= IMICRO3D_ActTable_getNumFrames( pMe->mpMicro3D,
                                                    &pMe->act_wait, 0 );
```

アニメーションするポジションを設定します。

```
IMICRO3D_Figure_setPosture( pMe->mpMicro3D, &pMe->figure, &pMe->act_wait, 0,
                             pMe->act_frame );
```

アニメーションフレームを更新します。この値を大きくするほど、高速にアニメーションすることになります。65536 という数値はフレームの単位です。例えば 3D ツールなどで 30 フレームでアクションを作成した場合、65536 * 30 という値が ActTable_getNumFrames()で取得されることとなります。

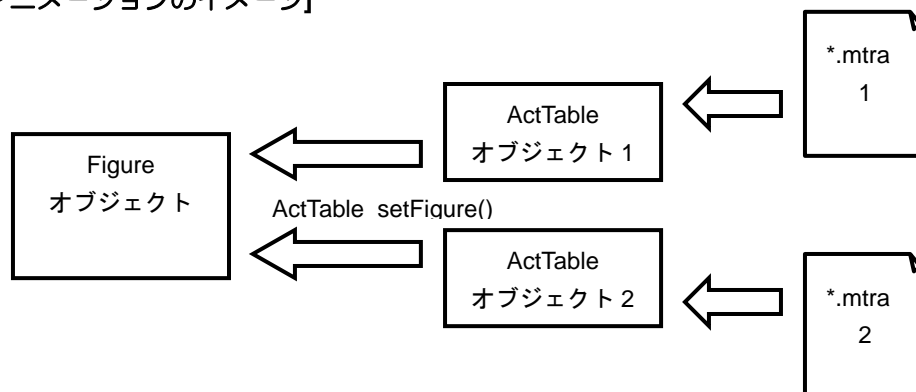
```
pMe->act_frame += 65536 * 2;
```

Step8 ・ 複数アニメーション

Step7 では1つのアニメーションを再生するのみでしたが、ゲームなどを作成する際には、1つの Figure に対し複数のアニメーションを付けたい場合などもあります。例えば、普段は歩いているキャラクターがジャンプしたり、走ったりするなどが考えられます。

そういった場合は、MTRA ファイルを複数用意し、それを1つの Figure オブジェクトに対して関連付けることで実現できます。(1つの MTRA ファイルで複数のアニメーションを実現する方法もあります。詳しくはプログラマーズマニュアルのアクションに関する項目を参照してください。)

[複数アニメーションのイメージ]



アニメーションの初期化、読み込み

まずはファイルをオープンします。

```
plFile = IFILEMGR_OpenFile( plFileMgr, pMe->mbactraFile[4], _OFM_READ );
if ( !plFile )
    return FALSE;
```

ファイルを読み込みます。この際、ActTable act_attack と関連付けが行われています。つまり、既存の ActTable act_wait とあわせて2つの ActTable が存在していることになります。その後、Figure と ActTable の関連付けを行います。

```
Filestream_initialize( &fis, plFileMgr, plFile, pMe );
rval = IMICRO3D_ActTable_loadMtraData( pMe->mpMicro3D,
                                       &pMe->act_attack, Filestream_getIstream( &fis ) );
...
// アクションと figure の関連付け
MICRO3D_ActTable_setFigure( pMe->mpMicro3D, &pMe->act_attack, &pMe->figure );
```

アニメーションの切替

あとはボタンなどに反応する形で、アニメーションを切り替えます。
今回のサンプルでは” SELECT” キー(真中ボタン)で pMe->isAttacking フラグが TRUE/FALSE に切り替わるように作成しています。

以下は MascotCapsule_Animate()での処理です。

```
if( pMe->isAttacking ){
    maxfrm = IMICRO3D_ActTable_getMaxFrame( pMe->mpMicro3D, &pMe->act_attack,
                                             pMe->act_idx );

    if ( pMe->act_frame > maxfrm ) {
        pMe->act_frame = 0;
        pMe->isAttacking = HI_FALSE;
    }else{
        IMICRO3D_Figure_setPosture( pMe->mpMicro3D, &pMe->figure,
                                     &pMe->act_attack, 0, pMe->act_frame );
    }
} else {
    pMe->act_frame %= IMICRO3D_ActTable_getNumFrames( pMe->mpMicro3D,
                                                       &pMe->act_wait, 0 );
    IMICRO3D_Figure_setPosture( pMe->mpMicro3D, &pMe->figure, &pMe->act_wait,
                                0, pMe->act_frame );
}
```

Step9 ・ 複数 3D モデル

ゲームなどでは複数のモデルを同時に描画することがあります。複数モデルを描画する際には、複数の MBAC を用意して描画するだけなのですが、描画の際にはいくつか注意すべき事項があります。

Figure の初期化、MBAC の読み込み

この部分については単体の MBAC を扱う際と変わりはありません。
今回は MascotCapsule_InitAppData()内で行っています。

```
// モデルデータ(*.mbac)を読み込む [ground.mbac]
pIFile = IFILEMGR_OpenFile( pIFileMgr, pMe->mbactraFile[5], _OFM_READ );
if ( !pIFile )
    return FALSE;

Filestream_initialize( &fis, pIFileMgr, pIFile, pMe);
rval = IMICRO3D_Figure_loadMbacData( pMe->mpMicro3D, &pMe->figure_ground,
Filestream_getIstream( &fis ) );
```

Figure の描画

描画を行う際には Render_drawFigure を行なう前に、必ずその Figure に対し設定したい Texture を Render_setTexture()しておきます。また、drawFigure()は Figure の個数分行われますが、Render_flush()は Figure の個数に関係なく 1 度だけコールします。

```
// ドラゴンの描画
IMICRO3D_Render_setTexture( pMe->mpMicro3D, &pMe->render, &pMe->texture[0] );
if(!IMICRO3D_Render_drawFigure( pMe->mpMicro3D, &pMe->render, &pMe->figure ))
    DBGPRINTF("draw failed");

// 地形の描画
IMICRO3D_Render_setTexture( pMe->mpMicro3D, &pMe->render, &pMe->texture[1] );
if(!IMICRO3D_Render_drawFigure( pMe->mpMicro3D, &pMe->render,
                                &pMe->figure_ground ))
    DBGPRINTF("draw failed");

...
IMICRO3D_Render_flush(pMe->mpMicro3D,&pMe->render);
```

※今回のサンプルでは 2 つの Figure をそのまま原点に描画していますが、Figure を個別に移動させて表示したい場合などは、それぞれの Figure 毎に行列を持たせ、個別に視点座標との演算を行う必要があります。複数モデル扱いについてはプログラマーズマニュアルにも項目を設けてありますので、そちらを参照してください。

Step10 ・ プリミティブの描画

ここまでのステップではあらかじめ作成した MBAC データを使うことでモデル表示を行いました。プログラムで動的にモデルを作成することもできます。ただし、描画を行えるのは基本図形(プリミティブ)である、点、線、三角形、四角形などのため、モデルとして描画するには、これらを複数組み合わせる必要があります。

今回のサンプルでは MascotCapsule_DrawPrimitives()内でプリミティブの描画を行っています。

データを用意する

プリミティブで図形を描画するためには次の情報が必要となります。

- 頂点情報 図形の頂点の座標です。図形に応じて数が変わります。
- 法線情報 法線とは光源の計算などに利用されます。面と垂直な線です。
- テクスチャ位置情報 図形に貼り付けるテクスチャの座標です。
テクスチャの座標は 2 次元で、UV 座標(u,v)などと呼ばれます。
- 色情報 図形につける色情報です。

頂点情報

頂点情報は面の数に応じて変わります。今回は立法体なので 6 面必要となっています。

```
hi_sint32 coord_array[]={
    0,0,0, 1000,0,0, 1000,1000,0, 0,1000,0,          /* 1 */
    1000,0,0, 1000,0,-1000, 1000,1000,-1000, 1000,1000,0, /* 2 */
    1000,1000,0, 1000,1000,-1000, 0,1000,-1000, 0,1000,0, /* 3 */
    1000,0,0, 1000,0,-1000, 0,0,-1000, 0,0,0,          /* 4 */
    0,0,-1000, 0,0,0, 0,1000,0, 0,1000,-1000,          /* 5 */
    1000,0,-1000, 0,0,-1000, 0,1000,-1000, 1000,1000,-1000, /* 6 */
};
coord.num = sizeof(coord_array)/sizeof(coord_array[0]);
coord.array = coord_array;
```

法線情報

法線情報も面の数に応じて変わります。立方体の場合、法線はそれぞれ別の方向を向くこととなります。

```
hi_sint32 normal_array[]={
    0,0,1, /* 1 */
    1,0,0, /* 2 */
    0,1,0, /* 3 */
    0,-1,0, /* 4 */
    -1,0,0, /* 5 */
    0,0,-1, /* 6 */
};
normal.num = sizeof(normal_array)/sizeof(normal_array[0]);
normal.array = normal_array;
```

テクスチャ位置情報

プリミティブに対しテクスチャを貼り付ける場合には必要となります。今回はテクスチャではなく、カラーポリゴンを使いますので不要です。カラーポリゴンを使わない場合は、逆にテクスチャが必要となります。

色情報

ポリゴン自体に単色の色をつける場合に指定します。色は RGB 値で指定します。

```
hi_sint32 color_array[] = {
    255<<16 | 0<<8 | 0, /* 1 */
    0<<16 | 255<<8 | 0, /* 2 */
    0<<16 | 0<<8 | 255, /* 3 */
    255<<16 | 255<<8 | 0, /* 4 */
    255<<16 | 0<<8 | 255, /* 5 */
    255<<16 | 255<<8 | 255, /* 6 */
};
color.num = sizeof(color_array)/sizeof(color_array[0]);
color.array = color_array;
```

描画する

各種パラメータを IMICRO3D_Render_drawPrimitive() に渡します。第三引数ではいくつかのフラグを OR していますが、これらは今回の条件に合わせたものです(表参照)。今回使用しているもの以外にもフラグは複数あるためここでは取り上げません。詳しくはプログラマーズマニュアルを参照してください。

```
IMICRO3D_Render_drawPrimitive(pMe->mpMicro3D, &pMe->render,
    M3DPD_QUADS | M3DPD_NORMAL_PER_FACE |
    M3DPD_COLOR_PER_FACE | M3DPD_TEXCOORD_NONE |
    M3D_BLEND_HALF | M3D_COLORKEY |
    (6<<16), &coord, &normal, NULL, &color);
```

フラグ名	解説
M3DPD_QUADS	プリミティブは四角ポリゴンで構成されます。
M3DPD_NORMAL_PER_FACE	プリミティブ単位で法線情報を持ちます。
M3DPD_COLOR_PER_FACE	プリミティブ単位で色情報を持ちます。
M3DPD_TEXCOORD_NONE	テクスチャを使用しません。
M3D_BLEND_HALF	半透明処理を行います。
M3D_COLORKEY	プリミティブはカラーキーによる透過処理の対象となります。

MascotCapsuleV3 Tutorial Document for BREW

日本語版

バージョン 2.0
発行日 2006年7月28日
発行者 株式会社 エイチアイ
〒153-0043 東京都目黒区東山 1-4-4 目黒東山ビル 5F
TEL.03-3710-2843 FAX.03-5773-8660
<http://www.hicorp.co.jp/>

－著作権・商標・免責事項について－

・本書は著作権法上の保護を受けています。本書の一部または全部について(ソフトウェアおよびプログラムを含む)、株式会社エイチアイから書面による承諾を得ずに、いかなる方法においても無断で複写、複製、転載することは禁じられています。

・MascotCapsule(R)は、株式会社エイチアイの日本国における登録商標です。本書に記載されているその他の製品名は、各社の商標、または登録商標です。

・本書に掲載されている情報を利用することで発生するトラブルや損失・損害に対して、当社は一切責任を負いません。

・本書の内容に関しては訂正・改善のため、将来予告なしに変更することがあります。

© 2006 HI CORPORATION. All Rights Reserved.