# インタラクティブゲーム制作
# ゲームプログラミング講座
# 第 9 回資料

**竹内 亮太**

(2009/7/24)

## 9　やっぱり 2 次元もいいよね！

色々な内容を織り交ぜてお届けしてきたこの講座ですが、今期の最後は 2D の描画について取り扱いたいと思います。今やゲームと言えば 3DCG が当たり前のように使われていますが、2D の表示要素なくしてはゲームは成り立ちません。メッセージ表示、メニュー形式の選択肢、アイコンやゲージなどは 2 次元画像によって表されています。今回は 3DCG のシステム上における、これら 2 次元画像の取り扱いについて述べます。

### 9.1　3DCG のカメラの仕組み

今更なんですが、3DCG のカメラの原理について少し述べます。本来は 3DCG の基礎を語るなら初っ端に話すような内容なんですが、FK ではそこらへんを気にしなくていいようにできているので、お話しする機会がありませんでした。今回は 2 次元画像の話ですが、原理を示す上で必要になるのでついでに教えておきます。

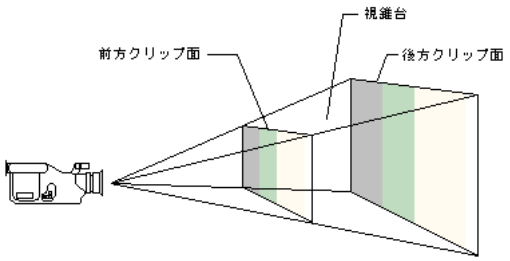


図 1: 視錐台

3DCG において遠近感を持った景観を描写するには、近くのモノは大きく、遠くのモノは小さく描く必要があります。このルールに従うと、3DCG において 1 つのカメラでフォローできる範囲は、図**??**のようなエリアになります。このような台形を貼り合わせたような形状のことを錐台といい、特に 3DCG のカメラを云々する際には視錐台と呼びます。

FK ではあらかじめ、この視錐台が自動的に設定されています。カメラのレンズの目の前を前方クリップ面、カメラの射程限界の面を後方クリップ面と呼びますが、FK のデフォルトでは前方は 1.0、後方は 6000.0 に設定されています。カメラに対して距離 1.0 より近いモノ、距離 6000.0 より遠いモノは描画されない訳です。

もう一つ重要な要素が画角です。カメラが真っ直ぐ向いている方向に対してどのくらいの角度のエリアをフレームに納めるかを設定することで、最終的な視錐台が決まります。FK のデフォルトでは画角は 40 度になっています。以下の図 2 における、$Z_{FRONT}, Z_{BACK}, fov$ がそれぞれ前面距離、後方距離、画角に相当します。

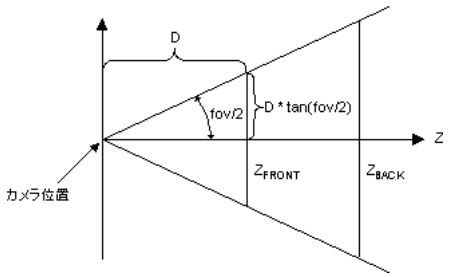


図 2: 視錐台の設定パラメータ

余談ですが、この画角を大きくすることによってズームレンズのような効果を出すことができます。カメラを近づけた場合とは違う表現になりますので、興味があるなら試してみても面白いでしょう。

### 9.2　馬の鼻先のニンジン

さて、2 次元の話をするといったのにバリバリ 3 次元の話をしていますが、これは致し方のないことなのです。実は数世代前から、家庭用ゲーム機などでも 3D の描画機能しか持たない機種の方が多くなっており、その時代から全ての 2D 描画物は今回述べている手法を用いています。なので 3 次元の原理を知らないとお話にならないのです。

3DCG では画像を取り扱う手段として「テクスチャ」という代物を使います。3 次元形状の表面に画像を貼り付けることで、画像が表面の形に応じて変形した状態で表示されるわけですね。アイコンやメッセージテキストを表示したかったら、それらの画像をカメラに対して真正面を向いた板ポリゴンに貼り付ければいいわけです。

とはいえ、ゲームではカメラの位置は頻繁に移動したり回転したりします。そのカメラの移動に合わせて

板ポリゴンの位置や向きを調整するのは面倒極まりないですね。これをどう解決するのかというと、セクション名のようなテクニックを使います。板ポリゴン自体をカメラ自体というか、レンズの表面にぴたっと貼り付ければ、カメラがどこへ移動しようと、どこへ向こうともくっついてきてくれます。

このように、あるモデル A に対してモデル B をくっつけることで、A が動いたり回転したときには常に一緒に移動・回転するような関係のことを、FK システムでは「親子関係」という機能で表現しています。これはカメラの前に板ポリを貼り付ける場合のみならず、複数のモデルから成り立っているキャラクターの位置や向きをコントロールする際にも、この親子関係を利用しています。fk_Model の親子関係機能については、マニュアルの 8 章 16 節を参照してください。親子関係はそれだけで 2 コマはしゃべれてしまう内容なので、後期にぜひ触れたいですね。

### 9.2.1 ニンジンの位置と大きさ

では画像を貼り付ける位置と大きさを考えていきましょう。まずは位置です。先ほどお話しした視錐台の前面距離ギリギリのところへ置きたくなりますが、あまりにギリギリだと表示されるかどうかが微妙になったりします。なので、最短距離の 1.0 から少し余裕を持って 2.0 の地点に配置することにしましょう。

カメラから 2.0 の距離では、どのくらいのサイズがジャストフィットなのでしょうか？視錘台的に考えてみます。図 2 を見返してみてください。これはカメラが Z 軸方向を見据えているのを横から見ている図だと思ってください。注目して欲しいのは $D * \tan(fov/2)$ と書かれている部分です。これはすなわち**カメラから距離 $D$ だけ離れた地点では、画面の中心からてっぺんまでのサイズが $D * \tan(fov/2)$ になる**ということを指します。

ということは、このサイズを実際のウィンドウのピクセルサイズで割ってやれば、1 ピクセルがその距離でどのくらいのサイズなのかが分かります。それさえ分かってしまえば後はこっちのものです。表示したい画像のサイズを X-Y 軸方向にかけてやればどんなサイズの画像だろうとジユウジザイ、画面全体をぴっちり覆うような画像もジャストフィーリングな感じで配置できます。しかもドットバイドットですから、変なギザギザや歪みも起きません。ハラショー！

### 9.2.2 実践してみよう

2D 画像の表示には fk_RectTexture を使うのが便利です。このクラスに直接画像を読み込ませることもできるのですが、それより fk_Image を介した方が後々のことを考えると便利です。fk_Image を fk_RectTexture にセットしたものを fk_Model にセットしたものを fk_Scene にエントリーすることになります。面倒かと思うかもしれませんが、こうやって階層構造になってるのが後々効いてくるんですよ。

```
fk_Image        img;
fk_RectTexture  rectTex;
fk_Model        texModel;
fk_Material     pixelMate;

pixelMate.setEmission(1.0, 1.0, 1.0);
pixelMate.setAlpha(1.0);

img.readPNG("hit.png");
tex.setImage(&img);
tex.setTexRendMode(FK_TEX_REND_SMOOTH);
texModel.setShape(&tex);
texModel.setMaterial(pixelMate);
```

pixelMate というマテリアルによって、光源の向きに関わらず常にギンギンの輝度で画像が表示されるようにしています。Emission は光源角度に関わらずに加える色を表す成分なので、これを最大 (1.0, 1.0, 1.0) にしておけば真っ白な画像がそのまま真っ白に表示されます。逆にこの成分をいじることで、画像に赤みや青みを持たせることもできます。Diffuse、Ambient、Specular は全て 0.0 にしておかないと、光源の状態によって明るさがちらちらしますので、Emission 以外は明示的に 0.0 にしておいた方がいいかもしれません。

setAlpha() は透過率を設定する関数で、これをいじると画像の透過率を操作できます。好きな透過具合の半透明にしたり、表示しながらこの数値を変更することで、フェードイン・アウトも表現できます。

```
double texSizeW = ONE_PIXEL*img.getWidth();
double texSizeH = ONE_PIXEL*img.getHeight();

texModel.setParent(&camera);
texModel.glMoveTo(0.0, 0.0, -2.0);
tex.setTextureSize(texSizeW, texSizeH);
scene.setBlendStatus(true);
scene.entryOverlayModel(&texModel);
```

テクスチャの大きさは、前述した 1 ピクセルの大きさに画像の幅と高さをかけたものを指定します。ONE_PIXEL には 距離 * tan(画角/2)/(ウィンドウ縦幅/2) が格納されているものとします。そしてシーンにエントリーする際には、通常の entryModel() ではなく entryOverlayModel() を使います。FK は entryModel() に登録したモノを先に描画してから entryOverlayModel() に登録したモノを描画するため、後から重ねるように描画したモノはこちらを使ってください。シーンについて setBlendStatus(true) という関数呼び出しがありますが、これをしておかないと半透明状態が描画されません。今日日半透明を使わないゲームも珍しいでしょうから、お決まりの処理として呼び出しておく方が良いでしょう。

## 9.3 応用テクニック色々

まず初っ端からですが、今回のサンプルではこのプリントに載せたような厄介な計算を外部に逃がしています。pixelBase という fk_Model の変数を使って、視錐台の最短距離から+1.0 の地点が基準点になるようにし、サイズの指定もスケールを変更することで「ピクセルサイズをそのまま指定すれば OK」にしてあります。pixelBase が camera の子になっているので、pixelBase を親に指定すれば結果的に camera へ追従します。このようにした方がきっとラクだと思いますので、是非真似してみてください。注意点としては、座標指定は画像の中心を指すことと、画面中心が (0, 0) で Y 軸の向きが通常のピクセル座標系とは反転していることです。

### 9.3.1 フェードイン・アウト

マテリアルを変更することで透明度を変化させていくことができますが、お行儀よくやる場合は毎回マテリアルの値を設定し、そのマテリアルをセットし直すという手順を踏む必要があります。ちょっとかったるいので、サンプルでは「モデルに登録したマテリアルのポインタを取得し、直接 α 値を書き換える」という荒技を使っています。これは「1 度マテリアルを設定したことのあるモデルにのみ通じる手段」ですので、気を付けてくださいね。

画面全体をフェードイン・アウトしたい場合は、真っ白や真っ黒な画像をセットし、それを画面全体のサイズで表示してやれば OK です。とっておきのネタの 1 つだったのに先週に言われたよ。ちくしょう。もうひとつのネタとしては「現在の画面表示から次の画面へクロスフェード」というものがあります。実現のポイントになるのは fk_Window の snapImage() 関数です。こいつを使うと、現在の画面表示を fk_Image に取り込むことができます。つまり、

1. snapImage() で現在の表示内容を fk_Image に取得
2. その Image をテクスチャとして準備して表示
3. 表示したら画面の裏側で次の画面を準備
4. 準備できたらテクスチャを透過させていく

で、クロスフェードになります。ここでポイントになるのが表示順の問題です。この場合のテクスチャは常に最前面に表示して、裏画面を隠す役割を果たさなくてはならないので、毎フレームエントリーさせるようにしています。思い通りの重なり具合にならない場合は、毎フレームエントリーさせ直すのが常套手段ですので、試してみてください。エントリー処理はほとんど負荷がかかりません。

### 9.3.2 組み合わせるとかっこいい

透過率の変化に加えて、スライド、拡大縮小などを組み合わせるとかなり格好良くなります。MacOS チックな表現もできるかもしれません。色々試してみてください。

### 9.3.3 切り取って使う

細かな数字のフォントやアイコンなどを、一つ一つのファイルに小分けにしておくのは読み込みに時間がかかりますし、スマートなやり方ではありません。あらかじめ 1 枚の画像の中に複数の素材を並べておき、切り出して使うのが効率的です。こうしておくと、fk_RectTexture や fk_Model に関しては表示したい個数分必要ですが、それに喰わせる fk_Image は 1 枚で済みます。

```
numTex.setTextureCoord(左下u, 左下v, 右上u, 右上v);
```

同じ fk_Image を喰わせて、setTextureCoord でキリトリ範囲を指定します。これはテクスチャ座標系なので、少し特殊な指定方法になります。画像の左下が (0.0, 0.0) を指し、右上が (1.0, 1.0) を指します。ピクセル単位で指定したかったら「ピクセル座標/幅 or 高さ」とする必要があります。そして、指定する順番は「左下の x, 左下の y, 右上の x, 右上の y」となります。普通の感覚で左上, 右下とやってしまうと、見事に上下反転した表示になります。それはそれで使いであったりするんですけどね。

このように、小さな部品を 1 枚の画像に詰め込むときは、できるだけ無駄のないように、コンピュータにとってキリの良いうれしいサイズになるようにしましょう。うれしいサイズとはズバリ、**2 のべき乗のサイズ**です。$256 \times 256$ や $512 \times 512$ は心底たまらないサイズですね。縦横で数値が揃っていなくても、それぞれがべき乗になっているならキリが良いと見なされます。

## 9.4 終わりに

さて、今期の間に色々なトピックを学んできましたが、いかがでしたでしょうか？私自身も「ゲームプログラミングを教える」というのが実に難しいということを改めて思い知りました。皆さん全員に満足のいく内容をお届けするのは、まず不可能だと思いますし、それは最初から目指してません。

「何をやればゲームプログラマーになれるのか」「どの本を読めばゲームは作れるのか」今の皆さんだったら、もうそんな質問はしてこないと思います。そんな約束された道など存在しないからです。どうしても約束された道が欲しいなら「SIGGRAPH で論文大賞をとったらなれるよ」とか「今出ているプログラミングや数学の本を、全て完全に理解して使いこなせるようになればできるよ」などど、非現実的な解を示すしかなくなります。

大事なのはメディア学の言うところの「問題発見・解決能力」につきるのです。自分がやろうとしていること、やらなければいけないことを分析し、その達成には何が問題となっているか、何が障害となっているか、何を学ぶべきかを把握すること。それせずに闇雲に突っ走っても、大抵徒労に終わることが多いです。

問題が把握できたなら、ステップの半分は通過したも同然です。そこで初めて「○○を学ぶにはどの本を読めばいいのだろう」「どういう単語で検索すればいいのだろう」といった解決策の検討に入れるわけです。もちろん知識や技術のある人に質問するのも有効です。

そして何より、全ての素材を統括するプログラマという存在は、全ての分野について精通とまではいかずとも、見識を広めておく必要があります。3D のモデル、画像、音、シナリオテキストの書式…これらを知らずして、オーサリングはできません。近年のゲーム開発は分業が当たり前になっていますけども、自分の担当じゃない所は知らんぷり、ではお話にならないわけです。まさに皆さん自身が、各分野の素材を媒介する「メディア」にならなくては、ゲームは作れないんじゃないかなーと思います。

## 9.5 課題提出について

ではいい話が済んだところで、課題についてもう一度お知らせします。

- お題：プログラムで何か作れ
- 手段：プログラムで作ってるなら何でもいい
- 仕様書期限：8/7(金) 23:59 まで
- 提出物期限：8/17(月) 15:00 まで

まずはどんなものを作るつもりなのかを仕様書にまとめてください。ここでは厳密な仕様書は要求しません。作成するつもりのプログラムの内容説明、仕様、機能、使い方をまとめるだけで構いません。細かい体裁などは通常のレポートと同様に考えてください。3 年生はアルファ版をそのまま提出物に代えてもらって結構です。2 年生でもチームによる制作を始めている場合は、共同制作物を提出物としてもらって構いません。どちらの場合でも仕様書は個人ごとに提出し、そこに担当箇所を明記するのを忘れないでください。

実際のモノを提出する場合は「実行可能な状態のフォルダ」と「コンパイルが通るプロジェクト」を圧縮して提出してください。別途インストールが必要なライブラリ (XNA など) を用いている場合は、何のどのバージョンを使っているかを仕様書に明記しておいてください。

提出方法については、Assit を使います。VPN も Assit も現在トラブルが起きているみたいですが、夏休み前にはなんとかなってると思います。

それではみなさん、夏休みを有意義に活用してください。大学生活の質は夏休みの過ごし方でで決まります。