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 2 回資料

竹内 亮太

(2009/5/22)

2　触って遊べる 3DCG プログラム

2.1　前回までのあらすじ

前回の宿題では「キー操作で表示物を動かす」というプログラムを目標にしてみました。これを自力で実現できた人は、結構いい線行ってます。ドキュメントから命令の使い方を調べて実際にやってみる、という流れは今後の実践的なプログラミングにおいて必要不可欠だからです。とはいえ、いきなりハードルが高い課題だったのも事実でしょう。今回はこれの答え合わせをしつつ、新たな要素を付け加えていきましょう。

2.2　座標系の話

「表示物を動かす」とさらっと言ってしまいましたが、皆さんは 3 次元座標空間をしっかり把握できていますか？2 次元の画面 (画像) は横方向 (x 軸) と縦方向 (y 軸) で話が済みますが、3 次元はこれに奥方向 (z 軸) の要素が加わります。(x, y, z) の 3 つの数値で位置を表すわけですね。これに加えて 3DCG では「カメラ」の概念が重要になります。**どの位置からどの方向を見ているかによって、画面の横方向・縦方向がどっちの向きになるのかが変わる**わけです。詳しくはメディア基礎演習 (3D コンテンツの基礎) のページを参照してみてください。

ではサンプルプログラムでのカメラの設定を確認しましょう。以下の部分に注目してください。

```
// 視点の位置と姿勢を設定
camera.glMoveTo(0.0, 0.0, 1000.0);
camera.glFocus(0.0, 0.0, 0.0);
camera.glUpvec(0.0, 1.0, 0.0);
```

FK では、視点や物体の位置は全て fk_Model の変数を使って表します。サンプルプログラムでは camera というそのままの名前の変数で視点を表しているため、この変数に対して位置や向きを指定することで視点が操作できることになります。視点を動かすプログラムも面白いのですが、物体と視点を同時に動かし始めると訳が分からなくなりますので、まずは設定を確認するだけにしておきます。glMoveTo() で座標 $(0, 0, 1000)$ に視点位置を持って行きます。かなり奥に持って行ってますね。そして glFocus() で「座標 $(0, 0, 0)$ を向け」と指示しています。glUpvec() は「自分が今いる位置から、頭のてっぺんが向いている方向を指示する」命令ですから、$(0, 1, 0)$ で地面に垂直に、まっすぐ立っている状態に設定されています。ということは、サンプルプログラムの状態だと

- 画面の横方向が x 軸、中心が 0 で右方向がプラス
- 画面の縦方向が y 軸、中心が 0 で上方向がプラス
- 画面の奥方向が z 軸、今居る位置が 1000 で奥方向がプラス

になっています。

ということは、キー操作に合わせて物を動かすには上記のルールに従って座標値をいじればいいことになります。サンプルプログラムで物体を表している変数は blockModel ですから、こいつに対して移動させる命令を呼びます。while ループ内に以下のコードを追記してみましょう。fk_Model に対して使える命令はマニュアルの第 8 章にまとまっていますので、色々試してみてください。

```
    // 直方体を Y 軸中心に回転
    blockModel.glRotateWithVec(0.0, 0.0, 0.0,
fk_Y, FK_PI/1000.0);

    // [これを追加]
    blockModel.glTranslate(1.0, 0.0, 0.0);
}
```

こうすると、物体は回転しながら右に流れて消えていきます。しかし当然、勝手に動いていってしまうので、このままでは「動かしている」感はありません。さて、どうしたものでしょう？

2.3　ゲームは繰り返しと条件分岐で作れる!?

私の知り合いのゲームプログラマーが生んだ名言です。ある意味真理です。ゲームプログラムの構造上、ぐ

るぐる回り続けるのが日常になっているということは前回お話ししました。その構造の中で

- あるキーが押されてる時はこうする
- キャラクターが空中にいたらこうする
- キャラクターとものがぶつかったらこうする

という条件分岐をひたすら書き連ねていくのがゲームプログラミングである、と言い切ってしまっても間違いではありません。もちろんこれらを正直に書いていたら、プログラムは間違いなくパンクします。プログラムよりも先にプログラマがパンクしかねません。それをどうにかするために、関数だのオブジェクト指向だのを持ち出して、効率的に書けるように工夫する必要があるのです。

その工夫の話は後回しにして、今は動きに反応して動いてくれることを目標にしましょう。カーソルキーの右キーに反応して横に移動するようにするには、以下のようにします。

```
    // キーが押されている時だけ右に移動
    if(window.getSpecialKetStatus(FK_RIGHT,
false)) {
        blockModel.glTranslate(1.0, 0.0, 0.0);
    }
}
```

fk_Window の変数がウィンドウを表すわけですが、この変数はキーが押されているかどうか、などといった命令も持っています。詳細は FK ユーザーマニュアルの第 10 章を見てください。これがうまく行ったら、次は上下左右に動かしたくなるのが人情でしょう。これはサンプルコードを載せませんので、自力でコードを追加して実現してみてください。

2.4 もうちょっとゲームらしく

こうして「操作」が曲がりなりにもできるようになってきたわけですが、このままでは見た目があまりにも殺風景です。ゲームの面白さは見た目だけではありませんが、あまりに無機質な表示内容だと気が滅入ってきます。そこで、このサンプルに以下の改良を加えましょう。

- 背景色をもっと明るい色にしてみる
- 床を作る
- もっとゲームっぽい視点にする

それぞれ実現方法を述べていきます。

2.4.1 背景色の変更

背景色は fk_Scene 型の変数でコントロールします。サンプルだと scene という名前の変数がありますね。こいつは背景色を変えるだけではなく、画面に表示するものを登録したり、取り消したりするという大役を任されている変数です。サンプルでも表示物を登録する entryModel()、視点を登録する entryCamera() という関数が使われているのが分かると思います。

色々ついでに説明したくなりますが、今は背景色に集中しましょう。マニュアルの第 9 章に一通りの機能が載っていますが、背景色の設定は次のようになります。while ループ突入前に以下を追記してください。

```
// 背景色を変更する
scene.setBGColor(0.0, 1.0, 1.0);
```

これで背景が爽やかな (目が痛い) 水色になります。色が気に入らなかったら、この引数は (R, G, B) を表している数値ですので、自由に変えてみてください。

2.4.2 床を作る

発想としては「幅と奥行きがでっかく、高さがちっちゃい直方体を作る」ことで実現できると考えられます。物を増やす際には、FK では二通りの考え方があります。

- 形が同じものを増やしたい→ fk_Model の変数を増やせばよい
- 形が違うものを増やしたい→ fk_Model の変数と、形を表す変数も追加

この場合の形を表す変数とは fk_Block のことになりますが、他にも様々な形表現変数の種類が存在します。どのようなものがあるかは第 4,5 章あたりを見てください。

今回は床用に別の形を用意したいので、fk_Model と fk_Block の両方を増やすことにしましょう。冒頭の変

数宣言部分に追記します。

```
    Fl_Window   mainWindow(512, 512, "FK TEST");
    fk_Model    camera, blockModel, lightModel,
floorModel; // 実際は改行せずに追加
    fk_Light    light;
    fk_Block    block(50.0, 70.0, 40.0),
floor(2000.0, 1.0, 2000.0); // 実際は改行せずに追
加
    fk_Scene    scene;
    fk_Window   window(0, 0, 512, 512);
```

fk_Block の変数は、作る時にブロックの大きさを決めることができます。最初に表示されていたのは幅 (x)50・高さ (y)70・奥行き (z)40 のブロックでしたが、floor は床用ということでアンバランスなサイズになっています。こうして作った floor を、fk_Model と結びつけてシーンに登録すれば、見事に表示されます。続けて変数への設定部も書いてみましょう。

```
    // 直方体の設定
    blockModel.setShape(&block);
    blockModel.setMaterial(Yellow);
    // 床の設定 (追加)
    floorModel.setShape(&floor); // 形とモデルを
結びつける
    floorModel.setMaterial(Gray1); // 色は他のも
のでも可

    // 各モデルをディスプレイリストに登録
    scene.entryCamera(&camera);
    scene.entryModel(&blockModel);
    scene.entryModel(&lightModel);
    scene.entryModel(&floorModel); // scene に表
示物として登録
```

「形状」→「モデル」→「シーン」という構造は、一見煩雑に見えるかも知れませんが実に便利な階層表現になっています。そのうちありがたみが分かってくるでしょう。

2.4.3 視点の変更

今の設定では「床と同じ高さ $(y = 0)$ で平行な視点」になっていて、何がなんだかよくわからないことになっていると思います。ここでは xz 軸方向に床が広がっているので、それを見下ろすような視点を考えたいと思います。見下ろすということは、視点の位置を高くするということ。高さを表す座標値は……もうお分かりですね？これは自力でやってみましょう。

視点の変更に合わせて、キー操作で物体を動かす方向も修正しましょう。先ほどは上下方向で y 軸方向に移動しましたが、見下ろしの画面になると少し不自然になります。z 軸方向の移動にした方が自然でしょう。

2.5 更に発展

ただ直方体が回るだけの状態から、だいぶゲームっぽい画面になってきたと思います。ここからは皆さんが思い描くゲームのイメージに近づくようなギミックを足していってみてください。

- 物体の動きに合わせて視点も動くようにする
- ジャンプできるようにする
- 障害物を置いてぶつかれるようにする

などが考えられます。どんどんとアクションゲームやパズルゲームの様相を呈してきましたが、どれもこれも 3DCG を使ったゲームなら定番の内容ばかりです。まずは自分なりのやり方で実現しようとしてみてください。

2.6 次回以降の内容

今回で色々な要素を足してきましたが、結構この while ループの中がごちゃごちゃしてきたと思います。まだ練習用のプログラムでこの有様ですから、本番のプログラムがどうなってしまうのか、ちょっと心配ですよね。

そこで、次からはオブジェクト指向のプログラミングを取り入れていきます。今までも「超変数」は利用してきたわけですが、これからは自分自身が書く部分についても「超変数化」していくことで、プログラムの修正や拡張、他のプログラマとの作業分担をしやすくしていきます。

更に、いつまでたってもブロックだけでは味気ないですから、基礎演習で登場しているキャラクターたちを自分のプログラムに登場させてみたいと思います。もちろんアニメーションさせられますから、一気に「ゲーム作ってる！」感が増すはずです。

2.7　今日の課題

1. カーソルキーの上下左右で物体が動くようにしてみよう
2. 視点を自分なりに工夫した位置、動きにしてみよう
3. (ちょっとチャレンジ)FK Performer のアニメーションキャラを取り込んでみよう
4. (かなりチャレンジ) ジャンプや障害物への接触を実現してみよう

ジャンプや障害物に関しては結構ハードルが高いですが、今後の講座でも取り上げるつもりですので「その前に自力で」頑張ってみてください。一度自分で挑戦しておいた方が「難しいポイント」が分かるのでいい予習になると思います。

FK Performer のアニメーション取り込みは、配布ページにサンプルが載っていますので、その通りにやれば案外簡単です。これは気軽に挑戦できると思います。予習の意味をこめて挑戦してみてください。