[OpenGL 3D 2018 第12回]

文字の発明

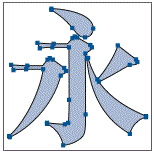
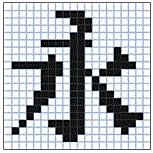
# 文字の表示方法を選ぶ

## アウトラインフォントとビットマップフォント

OpenGLはプリミティブを表示することしかできません。ですから、文字を表示する場合でもプリミティブを組み合わせて作らなければなりません。

現在、プリミティブを使って文字を表示する方法は二種類あります。  
ひとつは「アウトラインフォント」と呼ばれています。この方法では、文字を曲線の集合として定義し、ジオメトリシェーダーなどで曲線をたくさんのプリミティブの集合に変換して描画します。アウトラインフォントは、かなり大きく拡大してもキレイな文字を表示することができますが、あまり縮小すると見づらくなることが多いです。また、曲線からプリミティブへの変換をする計算を行うため、やや複雑なシェーダープログラムを必要とします。そのため、ビットマップフォントと比べるとたくさんの文字を表示するのは不得手です。

もうひとつは「ビットマップフォント」と呼ばれます。これは、文字を小さなピクセルの集合としてテクスチャにし、それをプリミティブに貼り付けて描画する方法です。テクスチャなので、あまり拡大するとギザギザが目立ってしまいますが、等倍で表示するときはアウトラインフォント以上に見やすい表示が可能です。また、大した計算をしなくてもいいので、シェーダープログラムも簡素なものになります。そのため、アウトラインフォントと比べて、たくさんの文字を表示することができます。



　　　　　　　［アウトラインフォント］　　　　　　　　　　　　　［ビットマップフォント］

## 表示方法の選択基準

どちらの方法を使うかは、コンピューターの性能やゲームの内容によります。古い世代のゲーム機には、アウトラインフォントを多用できるほどの性能がありませんでした。そのため、もっぱらビットマップフォントが使われました。  
ゲームのジャンルでいうと、ノベルゲームやアドベンチャーゲームなどのように、画面にあまり動きがなく、さまざまな大きさの文字を表示するゲームでは、アウトラインフォントのほうが見栄えがするでしょう。しかし、シューティングゲームやアクションゲームなどでは、ビットマップフォントの高速性が好まれる傾向にあります。なお、両者を組み合わせたハイブリッドな文字表示システムを作ることも可能です。ただし、メモリに両方の種類の文字データを読み込まなければならない点は注意が必要です。

さて、今回の文字表示には「ビットマップフォント」を使っていきます。  
理由は「作成するのが簡単だから」です。2D画像の表示はすでにタイトル画面でやっていますので、基本的には同じことをするだけです。

# 文字を表示する

## 得点の画像を用意する

まずはプレイヤーの得点を表示してみましょう。そのために、得点を示す「SCORE:」という画像と、０～９の数字の画像を用意してください。これらの画像は以下のURLから取得できます。

**SCORE画像: https://github.com/tn-mai/OpenGL3D2018/blob/master/Res/Score.tga  
数字画像: https://github.com/tn-mai/OpenGL3D2018/blob/master/Res/Number\_0.tga  
 ・  
 ・  
 ・  
 https://github.com/tn-mai/OpenGL3D2018/blob/master/Res/Number\_9.tga**

URLにアクセスしたら、右側にある「download」ボタンを押して画像をダウンロードし、みなさんのプロジェクトのResフォルダにコピーしておいてください。

次に、これらの画像を読み込むためのテクスチャ変数を追加します。  
MainGameSceneクラスに以下のプログラムを追加してください。

private:  
 MeshList meshList;  
  
 Texture::Image2D texId;  
 Texture::Image2D texTree;  
 Texture::Image2D texHouse;  
 Texture::Image2D texRock;  
 Texture::Image2D texHuman;  
 Texture::Image2D texBullet;  
 Texture::Image2D texZombie;  
  
+ // 情報表示用テクスチャ.  
+ Texture::Image2D texScore;  
+ Texture::Image2D texNumber[10];  
  
 Shader::Program progSimple;  
 Shader::Program progLighting;  
 Shader::LightList lights;

続いて、これらの変数にテクスチャを読み込みましょう。  
MainGameScene::Initialize関数に、次のプログラムを追加してください。

texId.Reset(Texture::CreateImage2D(  
 imageWidth, imageHeight, imageData, GL\_RGBA, GL\_UNSIGNED\_BYTE));  
 texTree.Reset(Texture::CreateImage2D(  
 imageWidth, imageHeight, imageTree, GL\_RGBA, GL\_UNSIGNED\_BYTE));  
 texHouse.Reset(Texture::LoadImage2D("Res/House.tga"));  
 texRock.Reset(Texture::LoadImage2D("Res/Rock.tga"));  
 texHuman.Reset(Texture::LoadImage2D("Res/Human.tga"));  
 texBullet.Reset(Texture::LoadImage2D("Res/Bullet.tga"));  
 texZombie.Reset(Texture::LoadImage2D("Res/Zombie.tga"));  
  
**+** // 情報表示用テクスチャを読み込む.  
**+** texScore.Reset(Texture::LoadImage2D("Res/Score.tga"));  
**+** for (int i = 0; i < 10; ++i) {  
**+** const std::string filename =  
**+** std::string("Res/Number\_") + (char)('0' + i) + ".tga";  
**+** texNumber[i].Reset(Texture::LoadImage2D(filename.c\_str()));  
**+** }  
**+**  
 // ライトの設定.  
 lights.ambient.color = glm::vec3(0.05f, 0.1f, 0.1f);  
 lights.directional.direction = glm::normalize(glm::vec3(5, -2, -2));

数値の画像ファイル名は番号が違うだけです。そこで、for文を使って番号の部分だけを変更したファイル名を作成しています。

## 得点を表示する

それでは、読み込んだ画像を使って得点を表示しましょう。  
MainGameScene::Render関数に、次のプログラムを追加してください。

RenderActorList(objectList, progLighting, meshList);  
 RenderActorList(enemyList, progLighting, meshList);  
 RenderActorList(playerBulletList, progLighting, meshList);  
   
 // ポイント・ライトの位置が分かるように適当なモデルを表示.  
 progSimple.Use();  
 progSimple.BindTexture(0, texId.Get());  
 for (int i = 0; i < 8; ++i) {  
 progSimple.Draw(meshList[4], lights.point.position[i],  
 glm::vec3(0, angleY, 0), glm::vec3(1.0f, -0.25f, 1.0f));  
 }  
  
**+** // 情報を表示.  
**+** {  
**+** glDisable(GL\_CULL\_FACE);  
**+** glDisable(GL\_DEPTH\_TEST);  
**+** glEnable(GL\_BLEND);  
**+** glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);  
**+**  
**+** progSimple.Use();  
**+  
+** // 正射影で描画するように行列を設定.  
**+** const glm::mat4x4 matProj = glm::ortho(  
**+** -400.0f, 400.0f, -300.0f, 300.0f, 1.0f, 500.0f);  
**+** const glm::mat4x4 matView = glm::lookAt(  
**+** glm::vec3(0, 0, 100), glm::vec3(0, 0, 0), glm::vec3(0, 1, 0));  
**+** progSimple.SetViewProjectionMatrix(matProj \* matView);  
**+  
+** const GLuint planeMeshId = meshList.Get(4); // この「4」は各自変更のこと.  
**+  
+** // スコアを表示.  
**+** progSimple.BindTexture(0, texScore.Get());  
**+** progSimple.Draw(planeMeshId,  
**+** glm::vec3(-120, 270, 0), glm::vec3(0), glm::vec3(96, 32, 1));  
**+  
+** const int maxScoreDigits = 8; // 表示する桁数. **+** int tmpScore = score;  
**+** for (int i = 0; i < maxScoreDigits; ++i) {  
**+** const int posX = -32 + 32 \* (maxScoreDigits – i);  
**+** const int number = tmpScore % 10;  
**+** tmpScore /= 10;  
**+** progSimple.BindTexture(0, texNumber[number].Get());  
**+** progSimple.Draw(planeMeshId,  
**+** glm::vec3(posX, 270, 0), glm::vec3(0), glm::vec3(32, 32, 1));  
**+** }  
**+** }  
**+** glActiveTexture(GL\_TEXTURE0);  
 glBindTexture(GL\_TEXTURE\_2D, 0);  
 }

基本的には、第10回で作成したタイトル画面を表示するプログラムと同じですが、今回はカリングも深度テストも無効化しています。カリングを無効にすると、上下反転、左右反転した画像を表示できるようになります(今回は特に使いませんが…)。また、深度テストを無効化すると、Zの値を常に0にすることができるようになるので、描画の順番だけを考えればよくなります。タイトル画面のときは既存の設定を流用しましたが、2D画像を表示するときはこの設定のほうが便利なことが多いと思います。

その後は、半透明合成機能を有効にし、正射影行列を設定します。  
そしてテクスチャを設定し、Draw関数でモデルを描画します。このとき、第10回の5.1節で作成したPlane.objを使って描画するようにしています。上記のテキストではモデルのインデックスを4にしていますが、ここはみなさんのプログラムに合わせてインデックスを修正してください。

得点の表示は「点数を10で割り、その余りをインデックスにしてテクスチャを選択する」ということを、表示する桁数の回数だけ繰り返すことで実現しています。例えば123点を10で割ると「12あまり3」ですから、最初の桁に表示するのは「３」のテクスチャになります。次に12を10で割ると「1あまり2」ですから、2桁目には「２」のテクスチャを表示します。そして1を10で割ると「0あまり1」なので、3桁目には「１」のテクスチャを表示します。残りの5桁はずっと「0あまり0」なので、全て「０」のテクスチャが表示される、という仕組みです。

プログラムが書けたら、**ビルドして実行してください。**  
画面上部に得点が表示され、ゾンビに打ち込むたびに得点が増えていけば成功です。

## プレイヤーの体力を表示する

得点の次は、プレイヤーの体力を表示しましょう。  
まずは体力を示すテクスチャが必要ですね。この画像は以下のURLから取得できます。

**HP画像: https://github.com/tn-mai/OpenGL3D2018/blob/master/Res/Score.tga**

URLにアクセスしたら、右側にある「download」ボタンを押して画像をダウンロードし、みなさんのプロジェクトのResフォルダにコピーしておいてください。

次に、画像を読み込むためのテクスチャ変数を追加します。  
MainGameSceneクラスに以下のプログラムを追加してください。

// 情報表示用テクスチャ.  
 Texture::Image2D texScore;  
 Texture::Image2D texNumber[10];  
+ Texture::Image2D texHP;  
  
 Shader::Program progSimple;  
 Shader::Program progLighting;  
 Shader::LightList lights;

続いてテクスチャを読み込みましょう。  
MainGameScene::Initialize関数に、次のプログラムを追加してください。

// 情報表示用テクスチャを読み込む.  
 texScore.Reset(Texture::LoadImage2D("Res/Score.tga"));  
 for (int i = 0; i < 10; ++i) {  
 const std::string filename =  
 std::string("Res/Number\_") + (char)('0' + i) + ".tga";  
 texNumber[i].Reset(Texture::LoadImage2D(filename.c\_str()));  
 }  
**+** texHP.Reset(Texture::LoadImage2D("Res/HP.tga"));  
  
 // ライトの設定.  
 lights.ambient.color = glm::vec3(0.05f, 0.1f, 0.1f);  
 lights.directional.direction = glm::normalize(glm::vec3(5, -2, -2));

これで体力を表示する準備ができました。  
それでは、体力を表示しましょう。MainGameScene::Render関数に、次のプログラムを追加してください。

const int maxScoreDigits = 8; // 表示する桁数. int tmpScore = score;  
 for (int i = 0; i < maxScoreDigits; ++i) {  
 const int posX = -32 + 32 \* (maxScoreDigits – i);  
 const int number = tmpScore % 10;  
 tmpScore /= 10;  
 progSimple.BindTexture(0, texNumber[number].Get());  
 progSimple.Draw(planeMeshId,  
 glm::vec3(posX, 270, 0), glm::vec3(0), glm::vec3(32, 32, 1));  
 }  
**+  
+** // プレイヤーの体力を表示.  
**+** progSimple.BindTexture(0, texHP.Get());  
**+** progSimple.Draw(planeMeshId,  
**+** glm::vec3(-336, -270, 0), glm::vec3(0), glm::vec3(48, 32, 1));  
**+  
+** const int maxHealthDigits = 2; // 表示する桁数. **+** int tmpHealth = player.health;  
**+** for (int i = 0; i < maxHealthDigits; ++i) {  
**+** const int posX = -300 + 32 \* (maxHealthDigits – i);  
**+** const int number = tmpHealth % 10;  
**+** tmpHealth /= 10;  
**+** progSimple.BindTexture(0, texNumber[number].Get());  
**+** progSimple.Draw(planeMeshId,  
**+** glm::vec3(posX, -270, 0), glm::vec3(0), glm::vec3(32, 32, 1));  
**+** }  
 }  
 glActiveTexture(GL\_TEXTURE0);  
 glBindTexture(GL\_TEXTURE\_2D, 0);  
 }

このプログラムは得点の表示とほとんど同じです。表示する座標が左下になり、表示する数値が2桁の体力になっているだけです。

プログラムが書けたら、**ビルドして実行してください。**  
左下に体力が表示されていたら成功です。