[OpenGL 3D 2018 第10回]

このシーンがすごい

# テクスチャIDを管理するクラスを作成する

## Image2Dクラスを定義する

Shader::Programクラスを作成したことで、いつglDeleteProgram関数を実行するべきかを考えなくてもいいようになりました。同様に、テクスチャを管理するクラスを作成して、glDeleteTexture関数を実行するタイミングを気にかけずに済むようにしていきます。

Texture.hを開き、次のプログラムを追加してください。

namespace Texture {  
  
 GLuint CreateImage2D(GLsizei width, GLsizei height, const GLvoid\* data,  
 GLenum format, GLenum type);  
 GLuint LoadImage2D(const char\* path);  
  
**+**/\*\*  
**+**\* 2Dテクスチャ.  
**+**\*/  
**+**class Image2D  
**+**{  
**+**public:  
**+** Image2D() = default;  
**+** explicit Image2D(GLuint texId);  
**+** ~Image2D();  
**+**  
**+** void Reset(GLuint texId);  
**+** bool IsNull() const;  
**+** GLuint Get() const;  
**+**  
**+**private:  
**+** GLuint id = 0;  
**+**};  
**+**  
} // namespace Texture

2D画像を管理するクラスなので、「Image2D(いめーじ・つー・でぃー)」という名前にしました。

## メンバ関数を定義する

それではメンバ関数を定義していきましょう。Texture.cppを開き、LoadImage2D関数の定義の下に、次のプログラムを追加してください。

// 読み込んだ画像データからテクスチャを作成する.  
 return CreateImage2D(width, height, buf.data(), format, type);  
 }  
  
**+**/\*\*  
**+**\* コンストラクタ.  
**+**\*  
**+**\* @param texId テクスチャ・オブジェクトのID.  
**+**\*/  
**+**Image2D::Image2D(GLuint texId)  
**+**{  
**+** Reset(texId);  
**+**}  
**+**  
**+**/\*\*  
**+**\* デストラクタ.  
**+**\*/  
**+**Image2D::~Image2D()  
**+**{  
**+** glDeleteTextures(1, &id);  
**+**}  
**+**  
**+**/\*\*  
**+**\* テクスチャ・オブジェクトを設定する.  
**+**\*  
**+**\* @param texId テクスチャ・オブジェクトのID.  
**+**\*/  
**+**void Image2D::Reset(GLuint texId)  
**+**{  
**+** glDeleteTextures(1, &id);  
**+** id = texId;  
**+**}  
**+**  
**+**/\*\*  
**+**\* テクスチャ・オブジェクトが設定されているか調べる.  
**+**\*  
**+**\* @retval true 設定されている.  
**+**\* @retval false 設定されていない.  
**+**\*/  
**+**bool Image2D::IsNull() const  
**+**{  
**+** return id;  
**+**}  
**+**  
**+**/\*\*  
**+**\* テクスチャ・オブジェクトを取得する.  
**+**\*  
**+**\* @return テクスチャ・オブジェクトのID.  
**+**\*/  
**+**GLuint Image2D::Get() const  
**+**{  
**+** return id;  
**+**}  
**+**  
 } // namespace Texture

Reset(りせっと)は、あとからテクスチャIDを設定したり変更したりするための関数です。

## Image2Dクラスを使う

それでは、作成したImage2Dクラスを使って、glDeleteTextures関数を削除していきましょう。

Main.cppを開き、テクスチャを作成するプログラムを次のように変更してください。

-GLuint texId = Texture::CreateImage2D(imageWidth, imageHeight, imageData,  
- GL\_RGBA, GL\_UNSIGNED\_BYTE);  
-GLuint texTree = Texture::CreateImage2D(imageWidth, imageHeight, imageTree,  
- GL\_RGBA, GL\_UNSIGNED\_BYTE);  
-GLuint texHouse = Texture::LoadImage2D("Res/House.tga");  
-GLuint texRock = Texture::LoadImage2D("Res/Rock.tga");  
-GLuint texHuman = Texture::LoadImage2D("Res/human.tga");  
**+**Texture::Image2D texId(Texture::CreateImage2D(imageWidth, imageHeight,  
**+** imageData, GL\_RGBA, GL\_UNSIGNED\_BYTE));  
**+**Texture::Image2D texTree(Texture::CreateImage2D(imageWidth, imageHeight,  
**+** imageTree, GL\_RGBA, GL\_UNSIGNED\_BYTE));  
**+**Texture::Image2D texHouse(Texture::LoadImage2D("Res/House.tga"));  
**+**Texture::Image2D texRock(Texture::LoadImage2D("Res/Rock.tga"));  
**+**Texture::Image2D texHuman(Texture::LoadImage2D("Res/human.tga"));

次に、glDeleteTextures関数呼び出しを削除します。main関数の末尾を次のように変更してください。

window.SwapBuffers();  
 }  
  
- glDeleteTextures(1, &texHuman);  
- glDeleteTextures(1, &texRock);  
- glDeleteTextures(1, &texHouse);  
- glDeleteTextures(1, &texTree);  
- glDeleteTextures(1, &texId);  
-  
 return 0;  
 }

**[課題01]** 全てのProgram::BindTexture関数について、テクスチャIDの指定を「texId.Get()」のようにImage2D::Get関数を使うように置き換えてください。

課題01を終えたらビルドして実行してください。Image2Dクラスを使う前と全く同じように表示されていたら成功です。

# LightList構造体に初期化機能を追加する

LightList構造体のメンバを都度初期化するのは面倒です。そこで、初期化を行う関数を追加しましょう。Shader.hを開き、LightList構造体に次のプログラムを追加してください。

/\*\*  
 \* ライトをまとめた構造体.  
 \*/  
 struct LightList  
 {  
 AmbientLight ambient;  
 DirectionalLight directional;  
 PointLight point;  
 SpotLight spot;  
  
**+** void Init();  
 };

次にShader.cppを開き、BuildFromFile関数定義の下に、Initメンバ関数の定義を追加してください。

const std::vector<GLchar> vsCode = ReadFile(vsPath);  
 const std::vector<GLchar> fsCode = ReadFile(fsPath);  
 return Build(vsCode.data(), fsCode.data());  
 }  
  
**+**/\*\*  
**+**\* ライトリストを初期化する.  
**+**\*  
**+**\* 全ての光源の明るさを0にする.  
**+**\*/  
**+**void LightList::Init()  
**+**{  
**+** ambient.color = glm::vec3(0);  
**+** directional.color = glm::vec3(0);  
**+** for (int i = 0; i < 8; ++i) {  
**+** point.color[i] = glm::vec3(0);  
**+** }  
**+** for (int i = 0; i < 4; ++i) {  
**+** spot.color[i] = glm::vec3(0);  
**+** }  
**+**}  
**+** /\*\*  
 \* コンストラクタ.  
 \*  
 \* @param id プログラム・オブジェクトのID.  
 \*/

これで初期化機能が追加されました。

# Shader::Programクラスの変更

## メンバ関数を追加する

現在のShader::Programクラスを作成するには、引数付きのコンストラクタを使うしかありません。引数がなくても作成できないと、クラスのメンバ変数などで使うには不便です。そこで、いくつかのメンバ関数を追加します。Shader.hを開き、Programクラスを次のように変更してください。

/\*\*  
 \* シェーダー・プログラム.  
 \*/  
 class Program  
 {  
 public:  
**+** Program();  
 explicit Program(GLuint programId);  
 ~Program();  
  
**+** void Reset(GLuint programId);  
**+** bool IsNull() const;  
 void Use();  
 void BindVertexArray(GLuint);  
 void BindTexture(GLuint, GLuint);  
 void SetLightList(const LightList&);  
 void SetViewProjectionMatrix(const glm::mat4&);  
 void Draw(const Mesh&,  
 const glm::vec3& t, const glm::vec3& r, const glm::vec3& s);  
  
 private:  
**-** GLuint id; // プログラムID.  
**+** GLuint id = 0; // プログラムID.  
  
 // uniform変数の位置.  
- GLint locMatMV;  
- GLint locPointLightPos;  
- GLint locPointLightCol;  
- GLint locDirLightDir;  
- GLint locDirLightCol;  
- GLint locAmbLightCol;  
- GLint locSpotLightPos;  
- GLint locSpotLightDir;  
- GLint locSpotLightCol;  
**+** GLint locMatMVP = -1;  
**+** GLint locPointLightPos = -1;  
**+** GLint locPointLightCol = -1;  
**+** GLint locDirLightDir = -1;  
**+** GLint locDirLightCol = -1;  
**+** GLint locAmbLightCol = -1;  
**+** GLint locSpotLightPos = -1;  
**+** GLint locSpotLightDir = -1;  
**+** GLint locSpotLightCol = -1;  
  
- glm::mat4 matVP;  
**+** glm::mat4 matVP = glm::mat4(1);  
 LightList lights;  
 };

いくつかのメンバ関数の宣言を追加し、メンバ変数には初期値を設定するようにしました。

次に、メンバ関数を定義しましょう。Shader.cppを開き、LightList::Init関数定義の下に、次のプログラムを追加してください。

for (int i = 0; i < 4; ++i) {  
 spot.color[i] = glm::vec3(0);  
 }  
 }  
  
**+**/\*\*  
**+**\* コンストラクタ.  
**+**\*/  
**+**Program::Program()  
**+**{  
**+** lights.Init();  
**+**}  
**+**  
 /\*\*  
 \* コンストラクタ.  
 \*

コンストラクタの次は、Resetメンバ関数とIsNullメンバ関数を定義します。デストラクタ定義の下に、次のプログラムを追加してください。

Program::~Program()  
 {  
 glDeleteProgram(id);  
 }  
  
**+**/\*\*  
**+**\* プログラム・オブジェクトを設定する.  
**+**\*  
**+**\* @param id プログラム・オブジェクトのID.  
**+**\*/  
**+**void Program::Reset(GLuint programId)  
**+**{  
**+** glDeleteProgram(id);  
**+** id = programId;  
**+** if (id == 0) {  
**+** locMatMVP = -1;  
**+** locPointLightPos = -1;  
**+** locPointLightCol = -1;  
**+** locDirLightDir = -1;  
**+** locDirLightCol = -1;  
**+** locAmbLightCol = -1;  
**+** locSpotLightPos = -1;  
**+** locSpotLightDir = -1;  
**+** locSpotLightCol = -1;  
**+** return;  
**+** }  
**+**  
**+** locMatMVP = glGetUniformLocation(id, "matMVP");  
**+** locPointLightPos = glGetUniformLocation(id, "pointLight.position");  
**+** locPointLightCol = glGetUniformLocation(id, "pointLight.color");  
**+** locDirLightDir = glGetUniformLocation(id, "directionalLight.direction");  
**+** locDirLightCol = glGetUniformLocation(id, "directionalLight.color");  
**+** locAmbLightCol = glGetUniformLocation(id, "ambientLight.color");  
**+** locSpotLightPos = glGetUniformLocation(id, "spotLight.posAndInnerCutOff");  
**+** locSpotLightDir = glGetUniformLocation(id, "spotLight.dirAndCutOff");  
**+** locSpotLightCol = glGetUniformLocation(id, "spotLight.color");  
**+**  
**+** const GLint texColorLoc = glGetUniformLocation(id, "texColor");  
**+** if (texColorLoc >= 0) {  
**+** glUseProgram(id);  
**+** glUniform1i(texColorLoc, 0);  
**+** glUseProgram(0);  
**+** }  
**+**}  
**+**  
**+**/\*\*  
**+**\* プログラム・オブジェクトが設定されているか調べる.  
**+**\*  
**+**\* @retval true 設定されている.  
**+**\* @retval false 設定されていない.  
**+**\*/  
**+**bool Program::IsNull() const  
**+**{  
**+** return id;  
**+**}  
**+**  
 /\*\*  
 \* プログラム・オブジェクトをグラフィックス・パイプラインに割り当てる.  
 \*  
 \* プログラム・オブジェクトを使い終わったらglUseProgram(0)を実行して解除すること.  
 \*/  
 void Program::Use()

Reset関数のうち、glGetUniformLocation関数を使う部分は引数付きコンストラクタと全く同じです。コピー&貼り付けを使ってください。

Reset関数が初期化の機能を備えているので、引数月コンストラクタはReset関数を呼び出すように変更します。引数付きコンストラクタを次のように変更してください。

/\*\*  
 \* コンストラクタ.  
 \*  
 \* @param id プログラム・オブジェクトのID.  
 \*/  
 Program::Program(GLuint id)  
 {  
- locMatMVP = glGetUniformLocation(id, "matMVP");  
- locPointLightPos = glGetUniformLocation(id, "pointLight.position");  
- locPointLightCol = glGetUniformLocation(id, "pointLight.color");  
- locDirLightDir = glGetUniformLocation(id, "directionalLight.direction");  
- locDirLightCol = glGetUniformLocation(id, "directionalLight.color");  
- locAmbLightCol = glGetUniformLocation(id, "ambientLight.color");  
- locSpotLightPos = glGetUniformLocation(id, "spotLight.posAndInnerCutOff");  
- locSpotLightDir = glGetUniformLocation(id, "spotLight.dirAndCutOff");  
- locSpotLightCol = glGetUniformLocation(id, "spotLight.color");  
-  
- const GLint texColorLoc = glGetUniformLocation(id, "texColor");  
- if (texColorLoc >= 0) {  
- glUseProgram(id);  
- glUniform1i(texColorLoc, 0);  
- glUseProgram(0);  
- }  
+ lights.Init();  
+ Reset(id);  
 }

デフォルトコンストラクタの追加によって、プログラムIDが設定されない状態があり得るようになってしまいました。そのため、プログラムIDが設定されていないのに描画が実行され得ます。対策として、プログラムIDが0のときは何もしないようにします。Program::Draw関数に次のプログラムを追加してください。

void Program::Draw(const Mesh& mesh, const glm::vec3& t, const glm::vec3& r,  
 const glm::vec3& s)  
 {  
+ if (id == 0) {  
+ return;  
+ }  
+  
 // モデル行列を計算する.  
 const glm::mat4 matScale = glm::scale(glm::mat4(1), s);  
 const glm::mat4 matRotateX = glm::rotate(glm::mat4(1), r.x,  
 glm::vec3(1, 0, 0));

これでプログラムIDが設定されていなくても安心ですね。

ここでビルドを行い、エラーが出ないことを確認してください。エラーが出ていたらそれを修正し、エラーが出なくなるまでビルドしてください。実行はしなくて構いません。

# 機能を関数に分ける

これまで、表示内容に関わるプログラムのほとんどはmain関数に書いてきました。いろいろな機能を追加して規模も大きくなってきたことですし、そろそろ機能ごとに関数に分けようと思います。  
各機能は次の5つの関数に分配していきます。

Initialize 画面の表示に必要なデータを準備する。

ProcessInput 入力を受けて内部状態を設定する。

Update プログラムの状態を更新する。

Render モデルの表示などを行う。

Finalize Initialize関数で準備したデータを破棄する。

これらの関数は、2Dシューティングのテキストでも同様の名前で作成していました。ですから、見覚えがあると思います。今回は「MainGameScene」(メインゲーム・シーン)という名前のクラスを作成し、上記の関数はそのメンバ関数として作って行こうと思います。

まずはクラスを作成します。  
Srcフォルダに「MainGameScene.h」という名前のヘッダーファイルを追加してください。  
追加したファイルを開き、次のプログラムを追加してください。

/\*\*  
\* @file MainGameScene.h  
\*/  
#ifndef MAINGAMESCENE\_H\_INCLUDED  
#define MAINGAMESCENE\_H\_INCLUDED  
#include "GLFWEW.h"  
#include "Texture.h"  
#include "Shader.h"  
#include "MeshList.h"  
#include "Scene.h"  
  
/\*\*  
\* メインゲーム画面.  
\*/  
class MainGameScene : public Scene  
{  
public:  
 MainGameScene() = default;  
 ~MainGameScene() = default;  
  
 bool Initialize();  
 void ProcessInput();  
 void Update();  
 void Render();  
 void Finalize();  
  
private:  
 MeshList meshList;  
  
 Texture::Image2D texId;  
 Texture::Image2D texTree;  
 Texture::Image2D texHouse;  
 Texture::Image2D texRock;  
 Texture::Image2D texHuman;  
  
 Shader::Program progSimple;  
 Shader::Program progLighting;  
 Shader::LightList lights;  
  
 glm::vec3 viewPos;  
 float pointLightAngle;  
};  
  
#endif // MAINGAMESCENE\_H\_INCLUDED