Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Чувашский государственный университет имени И.Н. Ульянова»

ФАКУЛЬТЕТ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедравычислительной техники

АГС.

Лабораторная работа №3.

«Вывод полигонов с использованием VBO и VAO.»

Выполнил: студент гр. ИВТ-41-15

Лазарев Дмитрий

Проверил:

Ст. преп. Галибин С.В.

Чебоксары – 2018

В лабораторной работе №3 необходимо:

1. Реализовать класс CMesh для загрузки и вывода меша на экран.Класс должен иметь структуру, описанную выше. Загрузка должна осуществляться из obj-файлов, а вывод осуществляться с использованием VBO-буфера и VAO-объекта.

2. Реализовать класс CResourceManager для централизованного управления ресурсами (мешем). Класс должен иметь структуру, приведенную выше, и должен быть реализован на основе шаблона проектирования Singleton. При загрузке нового меша необходимо проверять, не был ли данный меш загружен ранее. В случае если меш уже загружен, должен быть возвращен идентификатор ранее загруженного меша.

3. Реализовать вывод нескольких (не менее 5) объектов, часть из которых является одинаковыми. Вывести на консоль идентификаторы используемых мешей для каждого объекта.

Текст класса для работы с мешем (CMesh.cpp файл).

#include "CMesh.h"

int CMesh::Load(const char \* filename)

{

std::FILE \*file;

errno\_t error = fopen\_s(&file, filename, "rb");

char f[128] = "";

float x, y, z;

std::vector<glm::vec3> vertexVec;

std::vector<glm::vec2> texCoordVec;

std::vector<glm::vec3> NormVec;

std::vector<Face> FaceVec;

if (file)

{

while (!feof(file))

{

fscanf\_s(file, "%s", f, \_countof(f));

if (strcmp(f, "v") == 0)

{

float x, y, z;

fscanf\_s(file, "%f %f %f", &x, &y, &z);

vertexVec.push\_back(glm::vec3(x, y, z));

continue;

}

if (strcmp(f, "vt") == 0)

{

float s, t;

fscanf\_s(file, "%f %f", &s, &t);

texCoordVec.push\_back(glm::vec2(s, t));

continue;

}

if (strcmp(f, "vn") == 0)

{

float x, y, z;

fscanf\_s(file, "%f %f %f", &x, &y, &z);

NormVec.push\_back(glm::vec3(x, y, z));

continue;

}

if (strcmp(f, "f") == 0)

{

Face face;

fscanf\_s(file, "%i/%i/%i", &face.Indeces[0][0], &face.Indeces[0][1], &face.Indeces[0][2]);

fscanf\_s(file, "%i/%i/%i", &face.Indeces[1][0], &face.Indeces[1][1], &face.Indeces[1][2]);

fscanf\_s(file, "%i/%i/%i", &face.Indeces[2][0], &face.Indeces[2][1], &face.Indeces[2][2]);

FaceVec.push\_back(face);

}

}

fclose(file);

size\_t FaceCount = FaceVec.size();

VertexCount = FaceCount \* 3;

Vertex \*VertexArr = new Vertex[VertexCount];

for (int i = 0; i < FaceCount; ++i)

{

for (int j = 0; j < 3; ++j)

{

VertexArr[i \* 3 + j].vPosition[0] = vertexVec[FaceVec[i].Indeces[j][0] - 1].x;

VertexArr[i \* 3 + j].vPosition[1] = vertexVec[FaceVec[i].Indeces[j][0] - 1].y;

VertexArr[i \* 3 + j].vPosition[2] = vertexVec[FaceVec[i].Indeces[j][0] - 1].z;

VertexArr[i \* 3 + j].vTexCoord[0] = texCoordVec[FaceVec[i].Indeces[j][1] - 1].x;

VertexArr[i \* 3 + j].vTexCoord[1] = texCoordVec[FaceVec[i].Indeces[j][1] - 1].y;

VertexArr[i \* 3 + j].vNormal[0] = NormVec[FaceVec[i].Indeces[j][2] - 1].x;

VertexArr[i \* 3 + j].vNormal[1] = NormVec[FaceVec[i].Indeces[j][2] - 1].y;

VertexArr[i \* 3 + j].vNormal[2] = NormVec[FaceVec[i].Indeces[j][2] - 1].z;

}

}

// генерируем VBO

glGenBuffers(1, &VBO\_Index);

// указываем, что сгенерированный буфер хранит в себе массив данных

glBindBuffer(GL\_ARRAY\_BUFFER, VBO\_Index);

// загружаем вершины

glBufferData(GL\_ARRAY\_BUFFER, sizeof(Vertex) \* VertexCount, VertexArr, GL\_STATIC\_DRAW);

// генерируем VAO

glGenVertexArrays(1, &VAO\_Index);

// начинаем работу с созданным VAO

glBindVertexArray(VAO\_Index);

glBindBuffer(GL\_ARRAY\_BUFFER, VBO\_Index);

// указываем свойства аттрибутов шейдера

// позиция вершины

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_TRUE, sizeof(Vertex), reinterpret\_cast<GLvoid\*>(offsetof(Vertex, vPosition)));

// нормаль вершины

glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_TRUE, sizeof(Vertex), reinterpret\_cast<GLvoid\*>(offsetof(Vertex, vNormal)));

// текстурная координата вершины

glVertexAttribPointer(2, 2, GL\_FLOAT, GL\_TRUE, sizeof(Vertex), reinterpret\_cast<GLvoid\*>(offsetof(Vertex, vTexCoord)));

// включаем аттрибуты

glEnableVertexAttribArray(0);

glEnableVertexAttribArray(1);

glEnableVertexAttribArray(2);

// отвязываем VAO

glBindVertexArray(0);

delete[] VertexArr;

}

return 0;

}

void CMesh::Render(void)

{

glBindVertexArray(VAO\_Index);

glDrawArrays(GL\_TRIANGLES, 0, VertexCount);

glBindVertexArray(0);

}

Текст класса для работы с менеджером ресурсов (CResourceManager.cpp файл).

#include "СResourceManager.h"

int CResourceManager::LoadMesh(std::string filename)

{

int meshID;

auto variable = meshes\_index.find(filename);

if (variable != meshes\_index.end())

{

std::cout <<"Mesh: "<< filename << " exist "<< variable->second<<std::endl;

return variable->second;

}

else

{

CMesh\* Mesh = new CMesh;

Mesh->Load(filename.c\_str());

meshes.push\_back(Mesh);

meshes\_index.insert(std::pair<std::string, int>(filename, (int)meshes.size() - 1));

std::cout << "Mesh: " << filename << " is loaded with ID: " << (int)meshes.size() - 1 << std::endl;

return (int)meshes.size() - 1;

}

}

CMesh \* CResourceManager::GetMesh(int index)

{

CMesh \*Mesh;

try

{

Mesh = meshes[index];

}

catch (std::exception ex)

{

return nullptr;

}

return Mesh;

}

Текст функции main (загрузка мешей) и функции Display (вывод мешей).

void Display(void)

{

// отчищаем буфер цвета

glClearColor(1.0, 1.0, 1.0, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glEnable(GL\_DEPTH\_TEST);

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

glEnable(GL\_CULL\_FACE);

glCullFace(GL\_BACK);

// активируем шейдер

Shader.Activate();

// получаем матрицу проекции

mat4 ProjectionMatrix = Camera.GetProjectionMatrix();

// устанавливаем матрицу проекции

Shader.SetUniform("ProjectionMatrix", ProjectionMatrix);

// получаем матрицу наблюдения

mat4 ViewMatrix = Camera.GetViewMatrix();

// ВЫВОДИМ ПЕРВУЮ МОДЕЛЬ:

// формируем матрицу модели (единичная матрица)

// модель располагается в точке (3,0,0) без поворота

mat4 ModelMatrix1 = mat4(

vec4(1, 0, 0, 0), // 1‐ый столбец: направление оси ox

vec4(0, 1, 0, 0), // 2‐ой столбец: направление оси oy

vec4(0, 0, 1, 0), // 3‐ий столбец: направление оси oz

vec4(0, 0, 0, 1)); // 4‐ый столбец: позиция объекта (начала координат)

// устанавливаем матрицу наблюдения модели

mat4 ModelViewMatrix1 = ViewMatrix \* ModelMatrix1;

Shader.SetUniform("ModelViewMatrix", ModelViewMatrix1);

// устанавливаем uniform‐переменную отвечающую за цвет объекта(фрагментов)

vec4 Color1 = vec4(1.0, 0.0, 0.0, 1.0);

Shader.SetUniform("Color", Color1);

// вывод объекта

CMesh\* mesh = CResourceManager::Instance().GetMesh(MeshId[0]);

if (mesh != nullptr) mesh->Render();

// ВЫВОДИМ ВТОРУЮ МОДЕЛЬ:

// формируем матрицу модели (единичная матрица)

// модель располагается в точке (3,0,0) без поворота

mat4 ModelMatrix2 = mat4(

vec4(1, 0, 0, 0), // 1‐ый столбец: направление оси ox

vec4(0, 1, 0, 0), // 2‐ой столбец: направление оси oy

vec4(0, 0, 1, 0), // 3‐ий столбец: направление оси oz

vec4(10, 0, 0, 1)); // 4‐ый столбец: позиция объекта (начала координат)

// устанавливаем матрицу наблюдения модели

mat4 ModelViewMatrix2 = ViewMatrix \* ModelMatrix2;

Shader.SetUniform("ModelViewMatrix", ModelViewMatrix2);

// устанавливаем uniform‐переменную отвечающую за цвет объекта(фрагментов)

vec4 Color2 = vec4(0.0, 0.0, 1.0, 1.0);

Shader.SetUniform("Color", Color2);

// вывод объекта

mesh = CResourceManager::Instance().GetMesh(MeshId[1]);

if (mesh != nullptr) mesh->Render();

// ВЫВОДИМ ТРЕТЬЮ МОДЕЛЬ:

// формируем матрицу модели (единичная матрица)

// модель располагается в точке (3,0,0) без поворота

mat4 ModelMatrix3 = mat4(

vec4(1, 0, 0, 0), // 1‐ый столбец: направление оси ox

vec4(0, 1, 0, 0), // 2‐ой столбец: направление оси oy

vec4(0, 0, 1, 0), // 3‐ий столбец: направление оси oz

vec4(-10, -0.8, 0, 1)); // 4‐ый столбец: позиция объекта (начала координат)

// устанавливаем матрицу наблюдения модели

mat4 ModelViewMatrix3 = ViewMatrix \* ModelMatrix3;

Shader.SetUniform("ModelViewMatrix", ModelViewMatrix3);

// устанавливаем uniform‐переменную отвечающую за цвет объекта(фрагментов)

vec4 Color3 = vec4(0.0, 0.0, 0.0, 1.0);

Shader.SetUniform("Color", Color3);

// вывод объекта

mesh = CResourceManager::Instance().GetMesh(MeshId[2]);

if (mesh != nullptr) mesh->Render();

mat4 ModelMatrix4 = mat4(

vec4(1, 0, 0, 0), // 1‐ый столбец: направление оси ox

vec4(0, 1, 0, 0), // 2‐ой столбец: направление оси oy

vec4(0, 0, 1, 0), // 3‐ий столбец: направление оси oz

vec4(10, -0.8, 8, 1)); // 4‐ый столбец: позиция объекта (начала координат)

// устанавливаем матрицу наблюдения модели

mat4 ModelViewMatrix4 = ViewMatrix \* ModelMatrix4;

Shader.SetUniform("ModelViewMatrix", ModelViewMatrix4);

// устанавливаем uniform‐переменную отвечающую за цвет объекта(фрагментов)

vec4 Color4 = vec4(0.0, 1.0, 0.0, 1.0);

Shader.SetUniform("Color", Color4);

// вывод объекта

mesh = CResourceManager::Instance().GetMesh(MeshId[4]);

if (mesh != nullptr) mesh->Render();

mat4 ModelMatrix5 = mat4(

vec4(1, 0, 0, 0), // 1‐ый столбец: направление оси ox

vec4(0, 1, 0, 0), // 2‐ой столбец: направление оси oy

vec4(0, 0, 1, 0), // 3‐ий столбец: направление оси oz

vec4(7, -0.8, 8, 1)); // 4‐ый столбец: позиция объекта (начала координат)

// устанавливаем матрицу наблюдения модели

mat4 ModelViewMatrix5 = ViewMatrix \* ModelMatrix5;

Shader.SetUniform("ModelViewMatrix", ModelViewMatrix5);

// устанавливаем uniform‐переменную отвечающую за цвет объекта(фрагментов)

vec4 Color5 = vec4(0.0, 1.0, 0.0, 1.0);

Shader.SetUniform("Color", Color5);

// вывод объекта

mesh = CResourceManager::Instance().GetMesh(MeshId[4]);

if (mesh != nullptr) mesh->Render();

// смена переднего и заднего буферов

glutSwapBuffers();

};

void MeshesInit()

{

MeshId[0] = CResourceManager::Instance().LoadMesh("MESHES\\buildings\\chicken\_shop.obj");

MeshId[1] = CResourceManager::Instance().LoadMesh("MESHES\\buildings\\chicken\_shop.obj");

MeshId[2] = CResourceManager::Instance().LoadMesh("MESHES\\buildings\\drug\_store.obj");

MeshId[3] = CResourceManager::Instance().LoadMesh("MESHES\\natures\\big\_tree.obj");

MeshId[4] = CResourceManager::Instance().LoadMesh("MESHES\\natures\\big\_tree.obj");

}