Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Чувашский государственный университет имени И.Н. Ульянова»

ФАКУЛЬТЕТ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедравычислительной техники

АГС.

Лабораторная работа №6.

«Наложение текстур

с использованием GLSL-шейдеров.»

Выполнил: студент гр. ИВТ-41-15

Лазарев Дмитрий

Проверил:

Ст. преп. Галибин С.В.

Чебоксары – 2018

В лабораторной работе №6 необходимо:

1. Вывести на экран объекты, с наложенной на них текстурой. Для работы с текстурой необходимо реализовать класс CTexture. Загрузка текстур осуществляется через класс CResourceManager так, что класс для работы с объектом CGraphicObject хранит только идентификатор используемой текстуры, а не саму текстуру. Каждая текстура должна храниться в памяти в единственном экземпляре.
2. В методе init() класса CScene необходимо инициализировать библиотеку winsock, считать приветственное сообщение с сервера и вывести его на экран.
3. В методе simulate() класса CScene необходимо непрерывно считывать информацию о демо-сцене, и обновлять необходимые данные графических объектов.
4. Для хранения всех графических объектов сцены необходимо использовать контейнер map, ключом которого является идентификатор объекта, а не контейнер vector.

Текст класса CScene (\*.cpp файл).

#include "CScene.h"

void CScene::init(void)

{

pugi::xml\_parse\_result result = resources\_description.load\_file("Resources.xml");

Light = CreateLight();

// временная локальная переменная

CGraphicObject TempGraphicObject;

// первый объект

WSADATA wsa;

printf("\nInitialising Winsock...");

if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {

printf("Failed. Error Code : %d\n", WSAGetLastError());

}

else {

printf("OK\n");

}

s = socket(AF\_INET, SOCK\_STREAM, 0);

if (s == INVALID\_SOCKET) {

printf("Could not create socket : %d\n", WSAGetLastError());

}

else {

printf("Socket created.\n");

}

sockaddr\_in server;

server.sin\_family = AF\_INET;

server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server.sin\_port = htons(27000);

// подключение к серверу

int connect\_error = connect(s, (struct sockaddr \*)&server, sizeof(server));

if (connect\_error == SOCKET\_ERROR) {

printf("Connection failed : %d\n", WSAGetLastError());

}

else {

printf("Connection is established.\n");

};

// запрос приветственного сообщения (код запроса ‐ 0)

SNetworkHader header;

header.transactionID = transID; // номер транзакции

header.frameNumber = 1; // первый пакет ...

header.frameCount = 1; // ... из одного

header.dataLen = 1; // отправляется один байт данных (код запроса)

header.funcID = 0; // код запроса = 0 (запрос приветственного сообщения)

int send\_bytes\_count = send(s, (const char\*)&header, sizeof(header), 0);

if (send\_bytes\_count < sizeof(header)) {

printf("Data was not send: %d.\n", WSAGetLastError());

}

// считывание приветственного сообщения с сервера (код запроса ‐ 0)

int receive\_data\_count = recv(s, (char\*)&header, sizeof(header), 0);

if (receive\_data\_count <= 0) {

printf("Data was not received: %d.\n", WSAGetLastError());

}

else {

char \*str = new char[header.dataLen - 1];

recv(s, str, header.dataLen - 1, 0);

printf("%s\n", str);

delete[] str;

};

}

void CScene::simulate(float sec)

{

SNetworkHader header;

header.transactionID = ++transID;

header.frameNumber = 1;

header.frameCount = 1;

header.dataLen = 1;

header.funcID = 1;

int send\_bytes\_count = send(s, (const char\*)&header, sizeof(header), 0);

if (send\_bytes\_count < sizeof(header)) {

printf("Data was not send: %d.\n", WSAGetLastError());

}

int receive\_data\_count = recv(s, (char\*)&header, sizeof(header), 0);

if (receive\_data\_count <= 0) {

printf("Data was not received: %d.\n", WSAGetLastError());

}

else {

int count;

recv(s, reinterpret\_cast<char\*>(&count), sizeof(int), 0);

SGameObjectDescription Description;

CGraphicObject tempGraphicObject;

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

{

recv(s, reinterpret\_cast<char\*>(&Description), sizeof(Description), 0);

tempGraphicObject = CreateGraphicObject(Description.Model);

tempGraphicObject.setID(Description.ObjectID);

tempGraphicObject.setPosition(vec3(Description.x, Description.y, Description.z));

tempGraphicObject.setRotation(-Description.Yaw);

GraphicObjects[Description.ObjectID] = tempGraphicObject;

}

};

bool Forward = GetAsyncKeyState(VK\_UP);

bool Back = GetAsyncKeyState(VK\_DOWN);

bool Left = GetAsyncKeyState(VK\_LEFT);

bool Right = GetAsyncKeyState(VK\_RIGHT);

float dForward = (int(Forward) - int(Back))\*sec;

float dRigth = (int(Right) - int(Left))\*sec;

GetCursorPos(&newPossition);

float dHorizAngle = oldPossition.x - newPossition.x;

float dVertAngle = oldPossition.y - newPossition.y;

oldPossition = newPossition;

Camera.MoveOXZ(dForward, dRigth);

Camera.Rotate(dHorizAngle, dVertAngle);

}

CCamera \* CScene::getCamera(void)

{

return &Camera;

}

void CScene::draw()

{

// передаем в рендер‐менеджер используемую камеру

CRenderManager::Instance().setCamera(Camera);

// передаем в рендер‐менеджер используемый источник света

CRenderManager::Instance().setLight(Light);

// передаем все модели

for (auto it = GraphicObjects.begin(); it != GraphicObjects.end(); it++) {

CRenderManager::Instance().addToRenderQueue(it->second);

};

}

CGraphicObject CScene::CreateGraphicObject(std::string name)

{

CGraphicObject tempGraphicObject;

CMaterial tempMaterial;

float temp[3];

pugi::xml\_node resources = resources\_description.child("Resources");

pugi::xml\_node models = resources.child("Models");

pugi::xml\_node model = models.find\_child\_by\_attribute("id", name.c\_str());

pugi::xml\_attribute mesh = model.child("Mesh").attribute("path");

char mesh\_name[128];

sprintf\_s(mesh\_name, "%s", mesh.value());

tempGraphicObject.setMesh(CResourceManager::Instance().LoadMesh(mesh\_name));

pugi::xml\_node material = model.child("Material");

pugi::xml\_attribute texture = material.child("Texture").attribute("path");

char texture\_name[128];

sprintf\_s(texture\_name, "%s", texture.value());

tempGraphicObject.setTexture(CResourceManager::Instance().LoadTexture(texture\_name));

pugi::xml\_node phong = material.child("PhongParameters");

pugi::xml\_attribute diffuse = phong.attribute("diffuse");

sscanf\_s(diffuse.value(), "%f %f %f", &temp[0], &temp[1], &temp[2]);

vec4 vec = vec4(temp[0], temp[1], temp[2], 0);

tempMaterial.setDiffuse(vec);

pugi::xml\_attribute ambient = phong.attribute("ambient");

sscanf\_s(ambient.value(), "%f %f %f", &temp[0], &temp[1], &temp[2]);

vec = vec4(temp[0], temp[1], temp[2], 0);

tempMaterial.setAmbient(vec);

pugi::xml\_attribute specular = phong.attribute("specular");

sscanf\_s(specular.value(), "%f %f %f", &temp[0], &temp[1], &temp[2]);

vec = vec4(temp[0], temp[1], temp[2], 0);

tempMaterial.setSpecular(vec);

pugi::xml\_attribute shininess = phong.attribute("shininess");

sscanf\_s(shininess.value(), "%f", &temp[0]);

tempMaterial.setShininess(temp[0]);

tempGraphicObject.setMaterial(tempMaterial);

return tempGraphicObject;

}

CLight CScene::CreateLight()

{

CLight tempLight;

float temp[3];

pugi::xml\_node resources = resources\_description.child("Resources");

pugi::xml\_node light = resources.child("Light");

pugi::xml\_attribute direction = light.child("Direction").attribute("vector");

sscanf\_s(direction.value(), "%f %f %f", &temp[0], &temp[1], &temp[2]);

vec4 vec = vec4(temp[0], temp[1], temp[2], 0);

tempLight.setPosition(vec);

pugi::xml\_node phong = light.child("PhongParameters");

pugi::xml\_attribute diffuse = phong.attribute("diffuse");

sscanf\_s(diffuse.value(), "%f %f %f", &temp[0], &temp[1], &temp[2]);

vec = vec4(temp[0], temp[1], temp[2], 0);

tempLight.setDiffuse(vec);

pugi::xml\_attribute ambient = phong.attribute("ambient");

sscanf\_s(ambient.value(), "%f %f %f", &temp[0], &temp[1], &temp[2]);

vec = vec4(temp[0], temp[1], temp[2], 0);

tempLight.setAmbient(vec);

pugi::xml\_attribute specular = phong.attribute("specular");

sscanf\_s(specular.value(), "%f %f %f", &temp[0], &temp[1], &temp[2]);

vec = vec4(temp[0], temp[1], temp[2], 0);

tempLight.setSpecular(vec);

return tempLight;

}

Текст класса CTexture (\*.cpp файл).

#include "CTexture.h"

#include <iostream>

void CTexture::Load(const char \* filename)

{

ILuint TextureID = ilGenImage();

ilBindImage(TextureID);

wchar\_t Unicode\_String[256];

wsprintf(Unicode\_String, L"%S", filename);

bool rezult = ilLoadImage(Unicode\_String);

int width = ilGetInteger(IL\_IMAGE\_WIDTH);

int height = ilGetInteger(IL\_IMAGE\_HEIGHT);

int size = width \* height \* 4;

float \*Data = new float[size];

ilCopyPixels(0, 0, 0, width, height, 1, IL\_RGBA, IL\_FLOAT, Data);

glGenTextures(1, &TexIndex);

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, TexIndex);

glPixelStoref(GL\_UNPACK\_ALIGNMENT, 1);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, width, height, 0, GL\_RGBA, GL\_FLOAT, Data);

glGenerateMipmap(GL\_TEXTURE\_2D);

delete[] Data;

ilDeleteImage(TextureID);

}

void CTexture::Apply(GLenum texUnit)

{

glActiveTexture(texUnit);

glBindTexture(GL\_TEXTURE\_2D, TexIndex);

}

Текст класса 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;

}

}

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

{

int textureID;

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

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

{

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

return variable->second;

}

else

{

CTexture\* Texture = new CTexture;

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

textures.push\_back(Texture);

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

//std::cout << "Texture: " << filename << " is loaded with ID: " << (int)textures.size() - 1 << std::endl;

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

}

}

CMesh \* CResourceManager::GetMesh(int index)

{

CMesh \*Mesh;

try

{

Mesh = meshes[index];

}

catch (std::exception ex)

{

return nullptr;

}

return Mesh;

}

CTexture \* CResourceManager::GetTexture(int index)

{

CTexture \*Texture;

try

{

Texture = textures[index];

}

catch (std::exception ex)

{

return nullptr;

}

return Texture;

}