Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное

Образовательное учреждение высшего образования

«Уфимский университет науки и технологий»

Факультет информатики и робототехники

Кафедра Информатики

Отчет по лабораторной работе № 4

на тему: «Работа с тенями, скайбокс и карта нормалей в OpenGL»

Выполнил:

Студент группы ПРО-233Б Ермолов И О

Проверил:

канд. техн. наук В.А. Котельников

Уфа - 2023

**Цель работы**

Изучение работы с тенями, скайбокс и карта нормалей в OpenGL.

**Задание**

Выполнить 23-26 уроки по OpenGL <https://triplepointfive.github.io/ogltutor/>

**Ход работы**

1. В уроках 23-24 я научился загружать модели через Assimp и работать с тенями загруженных моделей.

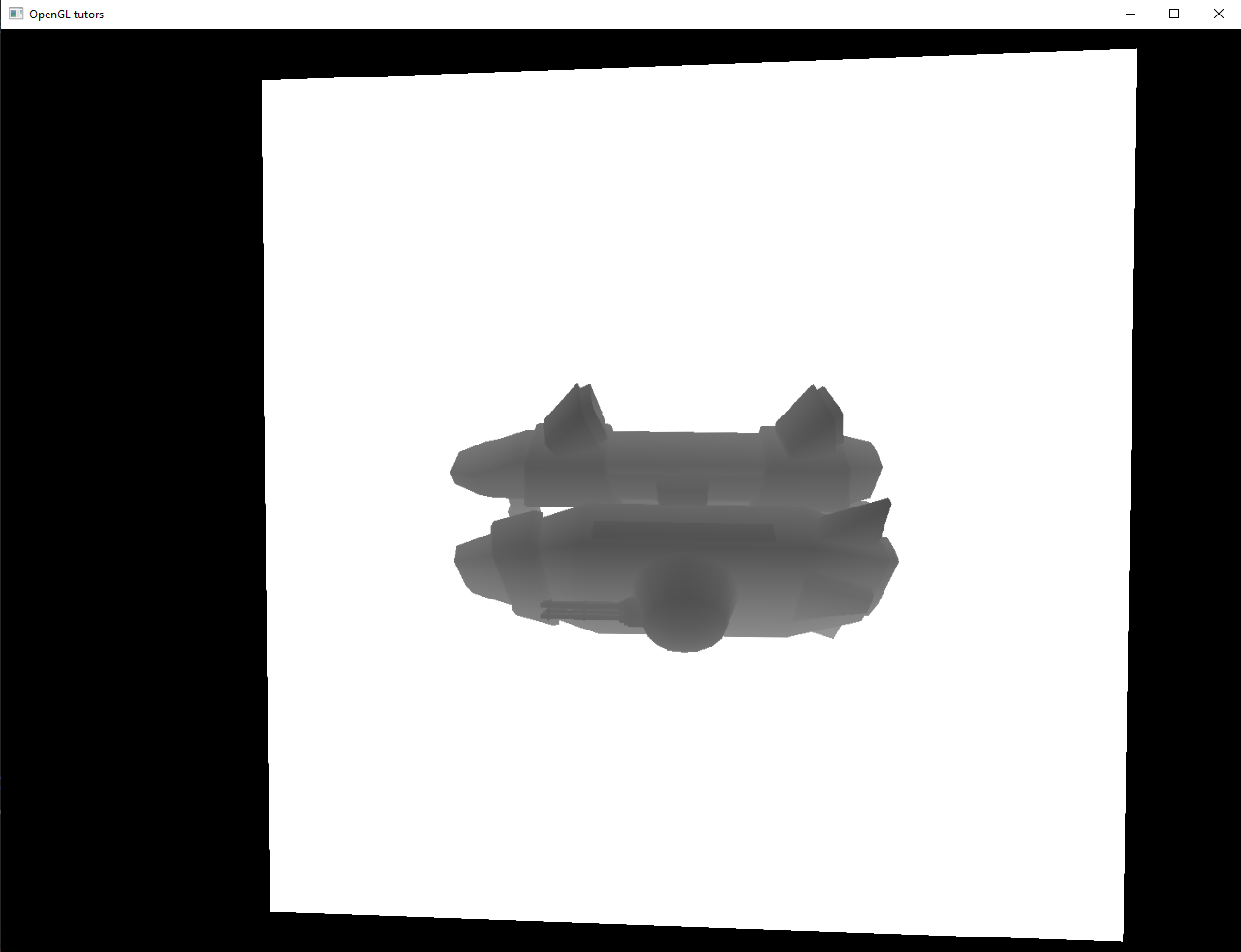


Рисунок 1. Загруженная модель с картой теней.

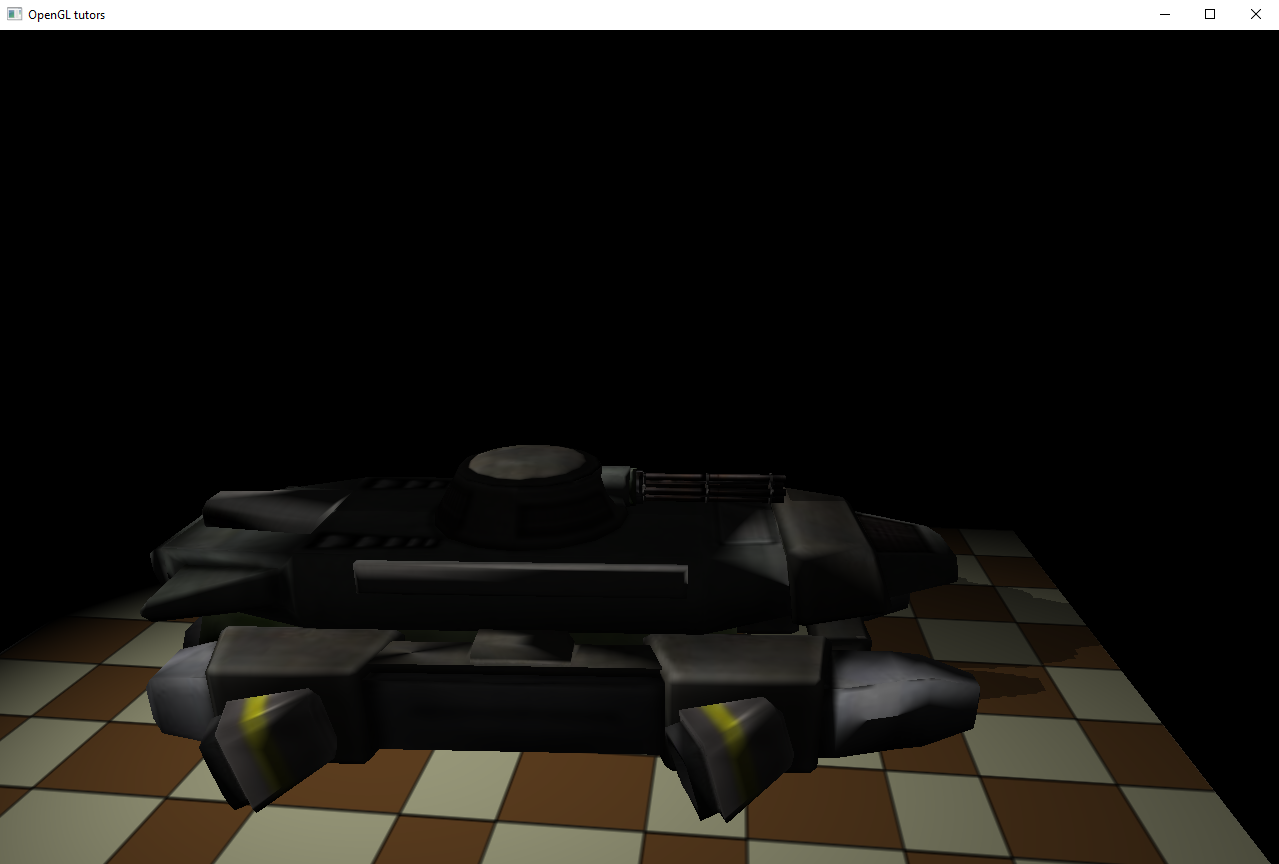


Рисунок 2. Другая загруженная модель с картой теней.

1. В 25 уроке я изучил назначение, определение и способ создания скайбокса.



Рисунок 3. Модель в окружении скайбокса.

1. В урокe 26 я изучил назначение и способ создание карты нормалей.

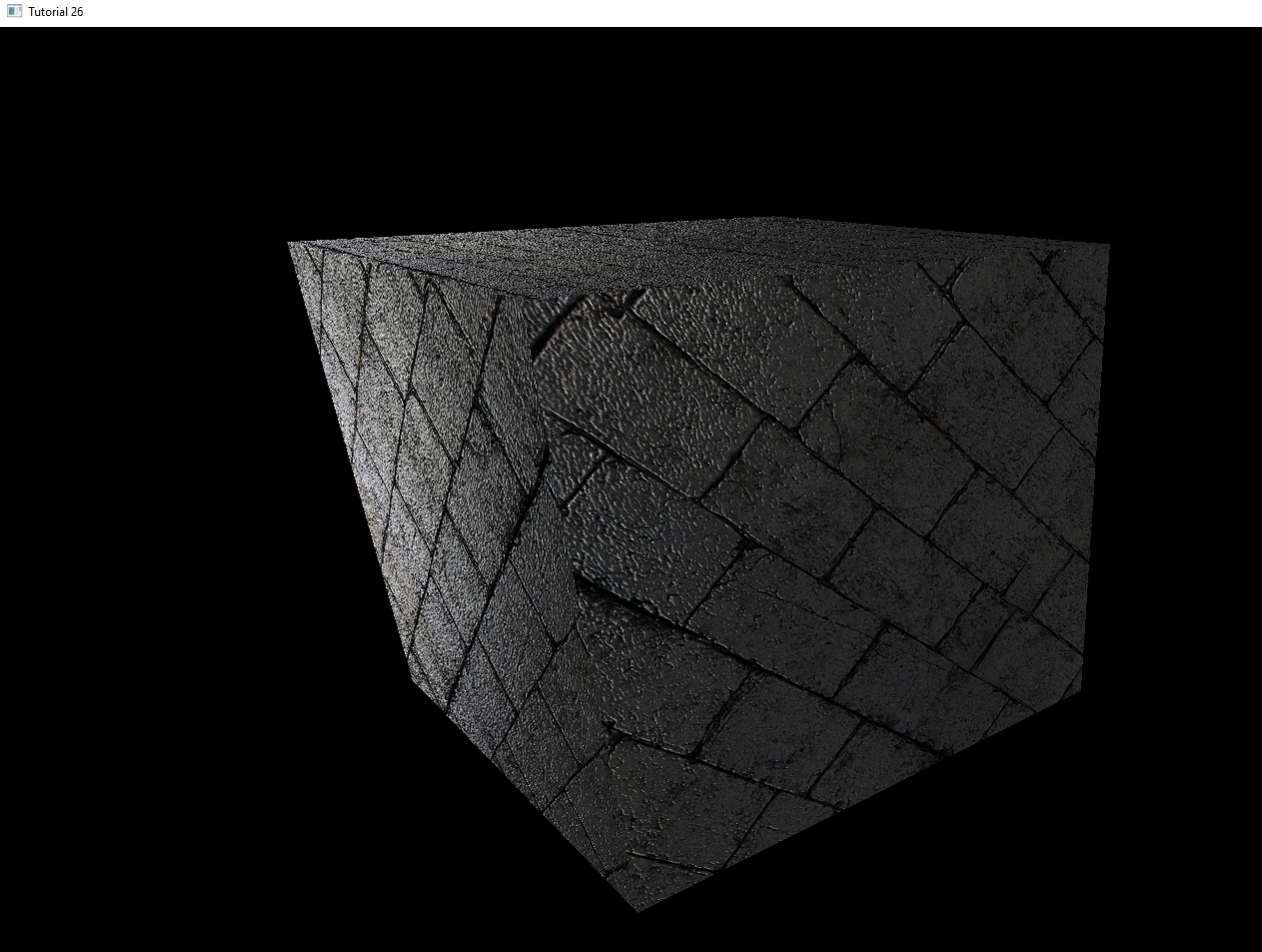


Рисунок 4. Модель с картой нормалей.

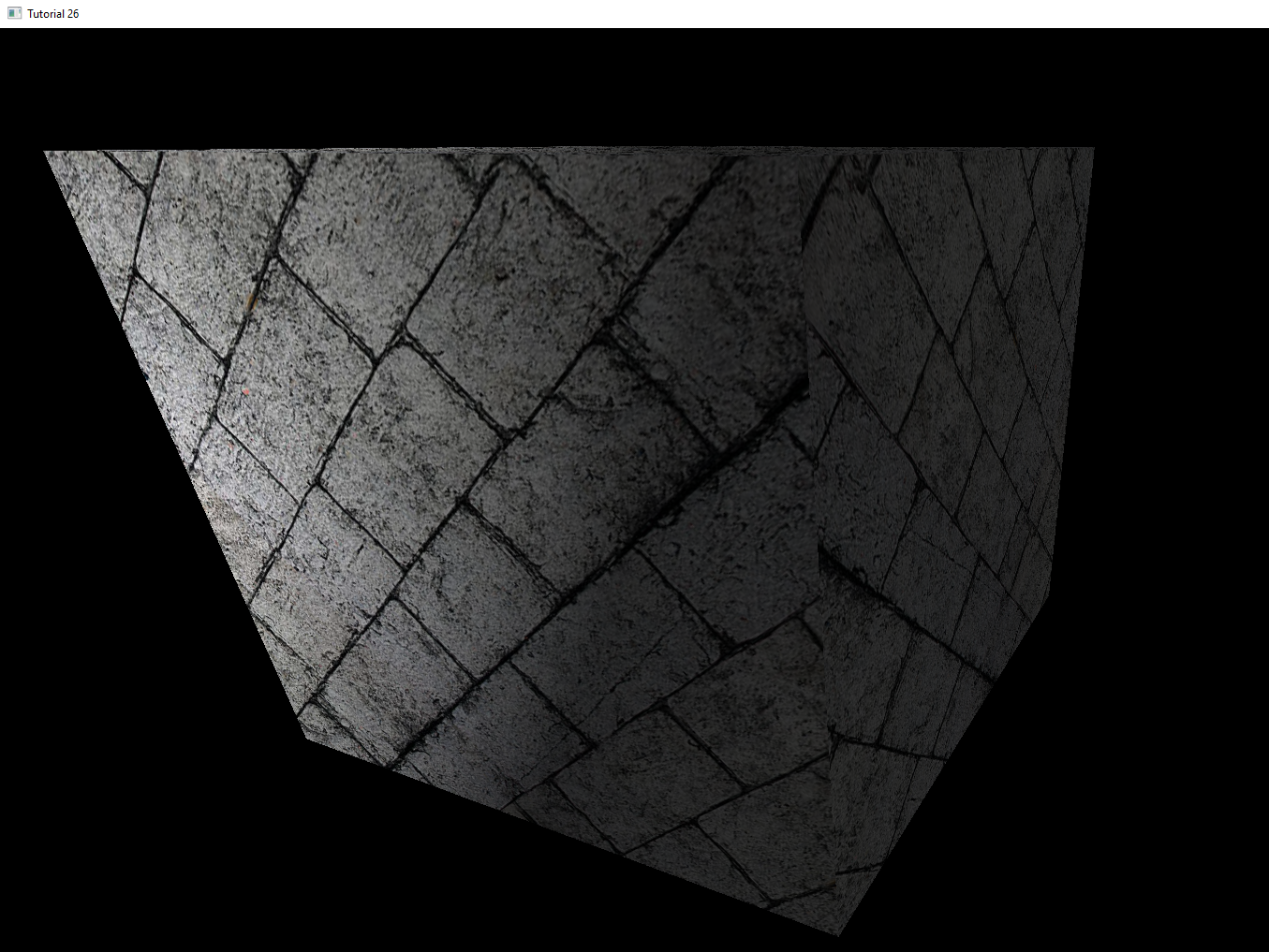


Рисунок 5. Модель без карты нормалей.

**Выводы к работе**

При выполнении данной лабораторной работы я познакомился с загрузкой моделей в OpenGL, способами создания теней, а так же со скайбоксом. Полный код программ по ссылке: [https://github.com/Ivan122727/EKG/tree/master/LW4](https://github.com/VerVyle/LW_ECG/tree/master/LW4)

**Приложение 1**

**main.cpp (task26)**

#include <math.h>

#include <GL/glew.h>

#include <GL/freeglut.h>

#include "engine\_common.h"

#include "util.h"

#include "pipeline.h"

#include "camera.h"

#include "texture.h"

#include "lighting\_technique.h"

#include "glut\_backend.h"

#include "mesh.h"

#define WINDOW\_WIDTH 1920

#define WINDOW\_HEIGHT 1200

class Tutorial26 : public ICallbacks

{

public:

Tutorial26()

{

m\_pLightingTechnique = NULL;

m\_pGameCamera = NULL;

m\_pSphereMesh = NULL;

m\_scale = 0.0f;

m\_pTexture = NULL;

m\_pNormalMap = NULL;

m\_pTrivialNormalMap = NULL;

m\_dirLight.AmbientIntensity = 0.2f;

m\_dirLight.DiffuseIntensity = 0.8f;

m\_dirLight.Color = Vector3f(1.0f, 1.0f, 1.0f);

m\_dirLight.Direction = Vector3f(1.0f, 0.0f, 0.0f);

m\_persProjInfo.FOV = 60.0f;

m\_persProjInfo.Height = WINDOW\_HEIGHT;

m\_persProjInfo.Width = WINDOW\_WIDTH;

m\_persProjInfo.zNear = 0.01f;

m\_persProjInfo.zFar = 100.0f;

m\_bumpMapEnabled = true;

}

~Tutorial26()

{

SAFE\_DELETE(m\_pLightingTechnique);

SAFE\_DELETE(m\_pGameCamera);

SAFE\_DELETE(m\_pSphereMesh);

SAFE\_DELETE(m\_pTexture);

SAFE\_DELETE(m\_pNormalMap);

SAFE\_DELETE(m\_pTrivialNormalMap);

}

bool Init()

{

Vector3f Pos(0.5f, 1.025f, 0.25f);

Vector3f Target(0.0f, -0.5f, 1.0f);

Vector3f Up(0.0, 1.0f, 0.0f);

m\_pGameCamera = new Camera(WINDOW\_WIDTH, WINDOW\_HEIGHT, Pos, Target, Up);

m\_pLightingTechnique = new LightingTechnique();

if (!m\_pLightingTechnique->Init()) {

printf("Error initializing the lighting technique\n");

return false;

}

m\_pLightingTechnique->Enable();

m\_pLightingTechnique->SetDirectionalLight(m\_dirLight);

m\_pLightingTechnique->SetColorTextureUnit(0);

m\_pLightingTechnique->SetNormalMapTextureUnit(2);

m\_pSphereMesh = new Mesh();

if (!m\_pSphereMesh->LoadMesh("../Content/box.obj")) {

return false;

}

m\_pTexture = new Texture(GL\_TEXTURE\_2D, "../Content/bricks.jpg");

if (!m\_pTexture->Load()) {

return false;

}

m\_pTexture->Bind(COLOR\_TEXTURE\_UNIT);

m\_pNormalMap = new Texture(GL\_TEXTURE\_2D, "../Content/normal\_map.jpg");

if (!m\_pNormalMap->Load()) {

return false;

}

m\_pTrivialNormalMap = new Texture(GL\_TEXTURE\_2D, "../Content/normal\_up.jpg");

if (!m\_pTrivialNormalMap->Load()) {

return false;

}

return true;

}

void Run()

{

GLUTBackendRun(this);

}

virtual void RenderSceneCB()

{

m\_pGameCamera->OnRender();

m\_scale += 0.01f;

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

m\_pLightingTechnique->Enable();

Pipeline p;

p.Rotate(0.0f, m\_scale, 0.0f);

p.WorldPos(sinf(m\_scale), 0.0f, 3.0f);

p.SetCamera(m\_pGameCamera->GetPos(), m\_pGameCamera->GetTarget(), m\_pGameCamera->GetUp());

p.SetPerspectiveProj(m\_persProjInfo);

m\_pTexture->Bind(COLOR\_TEXTURE\_UNIT);

if (m\_bumpMapEnabled)

{

m\_pNormalMap->Bind(NORMAL\_TEXTURE\_UNIT);

}

else

{

m\_pTrivialNormalMap->Bind(NORMAL\_TEXTURE\_UNIT);

}

m\_pLightingTechnique->SetWVP(p.GetWVPTrans());

m\_pLightingTechnique->SetWorldMatrix(p.GetWorldTrans());

m\_pSphereMesh->Render();

glutSwapBuffers();

}

virtual void IdleCB()

{

RenderSceneCB();

}

virtual void SpecialKeyboardCB(int Key, int x, int y)

{

m\_pGameCamera->OnKeyboard(Key);

}

virtual void KeyboardCB(unsigned char Key, int x, int y)

{

switch (Key) {

case 'q':

glutLeaveMainLoop();

break;

case 'b':

m\_bumpMapEnabled = !m\_bumpMapEnabled;

break;

}

}

virtual void PassiveMouseCB(int x, int y)

{

m\_pGameCamera->OnMouse(x, y);

}

private:

LightingTechnique\* m\_pLightingTechnique;

Camera\* m\_pGameCamera;

float m\_scale;

DirectionalLight m\_dirLight;

Mesh\* m\_pSphereMesh;

Texture\* m\_pTexture;

Texture\* m\_pNormalMap;

Texture\* m\_pTrivialNormalMap;

PersProjInfo m\_persProjInfo;

bool m\_bumpMapEnabled;

};

int main(int argc, char\*\* argv)

{

Magick::InitializeMagick(\*argv);

GLUTBackendInit(argc, argv);

if (!GLUTBackendCreateWindow(WINDOW\_WIDTH, WINDOW\_HEIGHT, 32, false, "Tutorial 26")) {

return 1;

}

Tutorial26\* pApp = new Tutorial26();

if (!pApp->Init()) {

return 1;

}

pApp->Run();

delete pApp;

return 0;

}

**main.cpp (task25)**

#include <math.h>

#include <GL/glew.h>

#include <GL/freeglut.h>

#include "util.h"

#include "pipeline.h"

#include "camera.h"

#include "texture.h"

#include "lighting\_technique.h"

#include "glut\_backend.h"

#include "mesh.h"

#include "skybox.h"

#define WINDOW\_WIDTH 1920

#define WINDOW\_HEIGHT 1200

class Main : public ICallbacks

{

public:

Main()

{

m\_pLightingTechnique = NULL;

m\_pGameCamera = NULL;

m\_pTankMesh = NULL;

m\_scale = 0.0f;

m\_pSkyBox = NULL;

m\_dirLight.AmbientIntensity = 0.2f;

m\_dirLight.DiffuseIntensity = 0.8f;

m\_dirLight.Color = Vector3f(1.0f, 1.0f, 1.0f);

m\_dirLight.Direction = Vector3f(1.0f, -1.0f, 0.0f);

m\_persProjInfo.FOV = 60.0f;

m\_persProjInfo.Height = WINDOW\_HEIGHT;

m\_persProjInfo.Width = WINDOW\_WIDTH;

m\_persProjInfo.zNear = 1.0f;

m\_persProjInfo.zFar = 100.0f;

}

virtual ~Main()

{

SAFE\_DELETE(m\_pLightingTechnique);

SAFE\_DELETE(m\_pGameCamera);

SAFE\_DELETE(m\_pTankMesh);

SAFE\_DELETE(m\_pSkyBox);

}

bool Init()

{

Vector3f Pos(0.0f, 1.0f, -20.0f);

Vector3f Target(0.0f, 0.0f, 1.0f);

Vector3f Up(0.0, 1.0f, 0.0f);

m\_pGameCamera = new Camera(WINDOW\_WIDTH, WINDOW\_HEIGHT, Pos, Target, Up);

m\_pLightingTechnique = new LightingTechnique();

if (!m\_pLightingTechnique->Init()) {

printf("Error initializing the lighting technique\n");

return false;

}

m\_pLightingTechnique->Enable();

m\_pLightingTechnique->SetDirectionalLight(m\_dirLight);

m\_pLightingTechnique->SetTextureUnit(0);

m\_pTankMesh = new Mesh();

if (!m\_pTankMesh->LoadMesh("../Content/phoenix\_ugv.md2")) {

return false;

}

m\_pSkyBox = new SkyBox(m\_pGameCamera, m\_persProjInfo);

if (!m\_pSkyBox->Init(".",

"../Content/sp3right.jpg",

"../Content/sp3left.jpg",

"../Content/sp3top.jpg",

"../Content/sp3bot.jpg",

"../Content/sp3front.jpg",

"../Content/sp3back.jpg")) {

return false;

}

return true;

}

void Run()

{

GLUTBackendRun(this);

}

virtual void RenderSceneCB()

{

m\_pGameCamera->OnRender();

m\_scale += 0.05f;

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

m\_pLightingTechnique->Enable();

Pipeline p;

p.Scale(0.1f, 0.1f, 0.1f);

p.Rotate(0.0f, m\_scale, 0.0f);

p.WorldPos(0.0f, -5.0f, 3.0f);

p.SetCamera(m\_pGameCamera->GetPos(), m\_pGameCamera->GetTarget(), m\_pGameCamera->GetUp());

p.SetPerspectiveProj(m\_persProjInfo);

m\_pLightingTechnique->SetWVP(p.GetWVPTrans());

m\_pLightingTechnique->SetWorldMatrix(p.GetWorldTrans());

m\_pTankMesh->Render();

m\_pSkyBox->Render();

glutSwapBuffers();

}

virtual void IdleCB()

{

RenderSceneCB();

}

virtual void SpecialKeyboardCB(int Key, int x, int y)

{

m\_pGameCamera->OnKeyboard(Key);

}

virtual void KeyboardCB(unsigned char Key, int x, int y)

{

switch (Key) {

case 'q':

glutLeaveMainLoop();

break;

}

}

virtual void PassiveMouseCB(int x, int y)

{

m\_pGameCamera->OnMouse(x, y);

}

private:

LightingTechnique\* m\_pLightingTechnique;

Camera\* m\_pGameCamera;

float m\_scale;

DirectionalLight m\_dirLight;

Mesh\* m\_pTankMesh;

SkyBox\* m\_pSkyBox;

PersProjInfo m\_persProjInfo;

};

int main(int argc, char\*\* argv)

{

Magick::InitializeMagick(\*argv);

GLUTBackendInit(argc, argv);

if (!GLUTBackendCreateWindow(WINDOW\_WIDTH, WINDOW\_HEIGHT, 32, false, "Tutorial 25")) {

return 1;

}

Main\* pApp = new Main();

if (!pApp->Init()) {

return 1;

}

pApp->Run();

delete pApp;

return 0;

}

**main.cpp (task24)**

#include <math.h>

#include <GL/glew.h>

#include <GL/freeglut.h>

#include "pipeline.h"

#include "camera.h"

#include "texture.h"

#include "lighting\_technique.h"

#include "glut\_backend.h"

#include "util.h"

#include "mesh.h"

#include "shadow\_map\_fbo.h"

#include "shadow\_map\_technique.h"

#define WINDOW\_WIDTH 1280

#define WINDOW\_HEIGHT 1024

class Main : public ICallbacks

{

public:

Main()

{

m\_pLightingEffect = NULL;

m\_pShadowMapEffect = NULL;

m\_pGameCamera = NULL;

m\_pMesh = NULL;

m\_pQuad = NULL;

m\_scale = 0.0f;

m\_pGroundTex = NULL;

m\_spotLight.AmbientIntensity = 0.1f;

m\_spotLight.DiffuseIntensity = 0.9f;

m\_spotLight.Color = Vector3f(1.0f, 1.0f, 1.0f);

m\_spotLight.Attenuation.Linear = 0.01f;

m\_spotLight.Position = Vector3f(-20.0, 20.0, 1.0f);

m\_spotLight.Direction = Vector3f(1.0f, -1.0f, 0.0f);

m\_spotLight.Cutoff = 20.0f;

}

virtual ~Main()

{

SAFE\_DELETE(m\_pLightingEffect);

SAFE\_DELETE(m\_pShadowMapEffect);

SAFE\_DELETE(m\_pGameCamera);

SAFE\_DELETE(m\_pMesh);

SAFE\_DELETE(m\_pQuad);

SAFE\_DELETE(m\_pGroundTex);

}

bool Init()

{

Vector3f Pos(3.0f, 8.0f, -10.0f);

Vector3f Target(0.0f, -0.2f, 1.0f);

Vector3f Up(0.0, 1.0f, 0.0f);

if (!m\_shadowMapFBO.Init(WINDOW\_WIDTH, WINDOW\_HEIGHT)) {

return false;

}

m\_pGameCamera = new Camera(WINDOW\_WIDTH, WINDOW\_HEIGHT, Pos, Target, Up);

m\_pLightingEffect = new LightingTechnique();

if (!m\_pLightingEffect->Init()) {

printf("Error initializing the lighting technique\n");

return false;

}

m\_pLightingEffect->Enable();

m\_pLightingEffect->SetSpotLights(1, &m\_spotLight);

m\_pLightingEffect->SetTextureUnit(0);

m\_pLightingEffect->SetShadowMapTextureUnit(1);

m\_pShadowMapEffect = new ShadowMapTechnique();

if (!m\_pShadowMapEffect->Init()) {

printf("Error initializing the shadow map technique\n");

return false;

}

m\_pQuad = new Mesh();

if (!m\_pQuad->LoadMesh("../Content/quad.obj")) {

return false;

}

m\_pGroundTex = new Texture(GL\_TEXTURE\_2D, "../Content/test.png");

if (!m\_pGroundTex->Load()) {

return false;

}

m\_pMesh = new Mesh();

return m\_pMesh->LoadMesh("../Content/phoenix\_ugv.md2");

}

void Run()

{

GLUTBackendRun(this);

}

virtual void RenderSceneCB()

{

m\_pGameCamera->OnRender();

m\_scale += 0.5f;

ShadowMapPass();

RenderPass();

glutSwapBuffers();

}

virtual void ShadowMapPass()

{

m\_shadowMapFBO.BindForWriting();

glClear(GL\_DEPTH\_BUFFER\_BIT);

m\_pShadowMapEffect->Enable();

Pipeline p;

p.Scale(0.1f, 0.1f, 0.1f);

p.Rotate(0.0f, m\_scale, 0.0f);

p.WorldPos(0.0f, 0.0f, 5.0f);

p.SetCamera(m\_spotLight.Position, m\_spotLight.Direction, Vector3f(0.0f, 1.0f, 0.0f));

p.SetPerspectiveProj(60.0f, WINDOW\_WIDTH, WINDOW\_HEIGHT, 1.0f, 50.0f);

m\_pShadowMapEffect->SetWVP(p.GetWVPTrans());

m\_pMesh->Render();

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

}

virtual void RenderPass()

{

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

m\_pLightingEffect->Enable();

m\_shadowMapFBO.BindForReading(GL\_TEXTURE1);

Pipeline p;

p.SetPerspectiveProj(60.0f, WINDOW\_WIDTH, WINDOW\_HEIGHT, 1.0f, 50.0f);

p.Scale(10.0f, 10.0f, 10.0f);

p.WorldPos(0.0f, 0.0f, 1.0f);

p.Rotate(90.0f, 0.0f, 0.0f);

p.SetCamera(m\_pGameCamera->GetPos(), m\_pGameCamera->GetTarget(), m\_pGameCamera->GetUp());

m\_pLightingEffect->SetWVP(p.GetWVPTrans());

m\_pLightingEffect->SetWorldMatrix(p.GetWorldTrans());

p.SetCamera(m\_spotLight.Position, m\_spotLight.Direction, Vector3f(0.0f, 1.0f, 0.0f));

m\_pLightingEffect->SetLightWVP(p.GetWVPTrans());

m\_pLightingEffect->SetEyeWorldPos(m\_pGameCamera->GetPos());

m\_pGroundTex->Bind(GL\_TEXTURE0);

m\_pQuad->Render();

p.Scale(0.1f, 0.1f, 0.1f);

p.Rotate(0.0f, m\_scale, 0.0f);

p.WorldPos(0.0f, 0.0f, 3.0f);

p.SetCamera(m\_pGameCamera->GetPos(), m\_pGameCamera->GetTarget(), m\_pGameCamera->GetUp());

m\_pLightingEffect->SetWVP(p.GetWVPTrans());

m\_pLightingEffect->SetWorldMatrix(p.GetWorldTrans());

p.SetCamera(m\_spotLight.Position, m\_spotLight.Direction, Vector3f(0.0f, 1.0f, 0.0f));

m\_pLightingEffect->SetLightWVP(p.GetWVPTrans());

m\_pMesh->Render();

}

virtual void IdleCB()

{

RenderSceneCB();

}

virtual void SpecialKeyboardCB(int Key, int x, int y)

{

m\_pGameCamera->OnKeyboard(Key);

}

virtual void KeyboardCB(unsigned char Key, int x, int y)

{

switch (Key) {

case 'q':

glutLeaveMainLoop();

break;

}

}

virtual void PassiveMouseCB(int x, int y)

{

m\_pGameCamera->OnMouse(x, y);

}

private:

LightingTechnique\* m\_pLightingEffect;

ShadowMapTechnique\* m\_pShadowMapEffect;

Camera\* m\_pGameCamera;

float m\_scale;

SpotLight m\_spotLight;

Mesh\* m\_pMesh;

Mesh\* m\_pQuad;

ShadowMapFBO m\_shadowMapFBO;

Texture\* m\_pGroundTex;

};

int main(int argc, char\*\* argv)

{

Magick::InitializeMagick(\*argv);

GLUTBackendInit(argc, argv);

if (!GLUTBackendCreateWindow(WINDOW\_WIDTH, WINDOW\_HEIGHT, 32, false, "OpenGL tutors")) {

return 1;

}

Main\* pApp = new Main();

if (!pApp->Init()) {

return 1;

}

pApp->Run();

delete pApp;

return 0;

}

**main.cpp (task23)**

#include <math.h>

#include <GL/glew.h>

#include <GL/freeglut.h>

#include "pipeline.h"

#include "camera.h"

#include "texture.h"

#include "lighting\_technique.h"

#include "glut\_backend.h"

#include "util.h"

#include "mesh.h"

#include "shadow\_map\_fbo.h"

#include "shadow\_map\_technique.h"

#define WINDOW\_WIDTH 1280

#define WINDOW\_HEIGHT 1024

class Main : public ICallbacks

{

public:

Main()

{

m\_pEffect = NULL;

m\_pShadowMapTech = NULL;

m\_pGameCamera = NULL;

m\_pMesh = NULL;

m\_pQuad = NULL;

m\_scale = 0.0f;

m\_spotLight.AmbientIntensity = 0.0f;

m\_spotLight.DiffuseIntensity = 0.9f;

m\_spotLight.Color = Vector3f(1.0f, 1.0f, 1.0f);

m\_spotLight.Attenuation.Linear = 0.01f;

m\_spotLight.Position = Vector3f(-20.0, 20.0, 5.0f);

m\_spotLight.Direction = Vector3f(1.0f, -1.0f, 0.0f);

m\_spotLight.Cutoff = 20.0f;

}

virtual ~Main()

{

SAFE\_DELETE(m\_pEffect);

SAFE\_DELETE(m\_pShadowMapTech);

SAFE\_DELETE(m\_pGameCamera);

SAFE\_DELETE(m\_pMesh);

SAFE\_DELETE(m\_pQuad);

}

bool Init()

{

if (!m\_shadowMapFBO.Init(WINDOW\_WIDTH, WINDOW\_HEIGHT)) {

return false;

}

m\_pGameCamera = new Camera(WINDOW\_WIDTH, WINDOW\_HEIGHT);

m\_pEffect = new LightingTechnique();

if (!m\_pEffect->Init()) {

printf("Error initializing the lighting technique\n");

return false;

}

m\_pShadowMapTech = new ShadowMapTechnique();

if (!m\_pShadowMapTech->Init()) {

printf("Error initializing the shadow map technique\n");

return false;

}

m\_pShadowMapTech->Enable();

m\_pQuad = new Mesh();

if (!m\_pQuad->LoadMesh("../Content/quad.obj")) {

return false;

}

m\_pMesh = new Mesh();

return m\_pMesh->LoadMesh("../Content/phoenix\_ugv.md2");

}

void Run()

{

GLUTBackendRun(this);

}

virtual void RenderSceneCB()

{

m\_pGameCamera->OnRender();

m\_scale += 0.05f;

ShadowMapPass();

RenderPass();

glutSwapBuffers();

}

virtual void ShadowMapPass()

{

m\_shadowMapFBO.BindForWriting();

glClear(GL\_DEPTH\_BUFFER\_BIT);

Pipeline p;

p.Scale(0.2f, 0.2f, 0.2f);

p.Rotate(0.0f, m\_scale, 0.0f);

p.WorldPos(0.0f, 0.0f, 5.0f);

p.SetCamera(m\_spotLight.Position, m\_spotLight.Direction, Vector3f(0.0f, 1.0f, 0.0f));

p.SetPerspectiveProj(60.0f, WINDOW\_WIDTH, WINDOW\_HEIGHT, 1.0f, 50.0f);

m\_pShadowMapTech->SetWVP(p.GetWVPTrans());

m\_pMesh->Render();

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

}

virtual void RenderPass()

{

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

m\_pShadowMapTech->SetTextureUnit(0);

m\_shadowMapFBO.BindForReading(GL\_TEXTURE0);

Pipeline p;

p.Scale(5.0f, 5.0f, 5.0f);

p.WorldPos(0.0f, 0.0f, 10.0f);

p.SetCamera(m\_pGameCamera->GetPos(), m\_pGameCamera->GetTarget(), m\_pGameCamera->GetUp());

p.SetPerspectiveProj(60.0f, WINDOW\_WIDTH, WINDOW\_HEIGHT, 1.0f, 50.0f);

m\_pShadowMapTech->SetWVP(p.GetWVPTrans());

m\_pQuad->Render();

}

virtual void IdleCB()

{

RenderSceneCB();

}

virtual void SpecialKeyboardCB(int Key, int x, int y)

{

m\_pGameCamera->OnKeyboard(Key);

}

virtual void KeyboardCB(unsigned char Key, int x, int y)

{

switch (Key) {

case 'q':

glutLeaveMainLoop();

break;

}

}

virtual void PassiveMouseCB(int x, int y)

{

m\_pGameCamera->OnMouse(x, y);

}

private:

LightingTechnique\* m\_pEffect;

ShadowMapTechnique\* m\_pShadowMapTech;

Camera\* m\_pGameCamera;

float m\_scale;

SpotLight m\_spotLight;

Mesh\* m\_pMesh;

Mesh\* m\_pQuad;

ShadowMapFBO m\_shadowMapFBO;

};

int main(int argc, char\*\* argv)

{

Magick::InitializeMagick(\*argv);

GLUTBackendInit(argc, argv);

if (!GLUTBackendCreateWindow(WINDOW\_WIDTH, WINDOW\_HEIGHT, 32, false, "OpenGL tutors")) {

return 1;

}

Main\* pApp = new Main();

if (!pApp->Init()) {

return 1;

}

pApp->Run();

delete pApp;

return 0;

}