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 № 5

на тему: «Геометрический шейдер, частицы, vertex, deferred shading в OpenGL»

Выполнил:

Студент группы ПРО-233Б Ермолов И О

Проверил:

канд. техн. наук В.А. Котельников

Уфа - 2023

**Цель работы**

Изучение работы с геометрическим шейдером, частицами , Vertex, Deferred shading в OpenGL.

З**адание**

Выполнить 27, 28, 32, 33 уроки по OpenGL <https://triplepointfive.github.io/ogltutor/>

**Ход работы**

1. В урокe 27 я научился создавать billboard и работать с геометрическим шейдером.

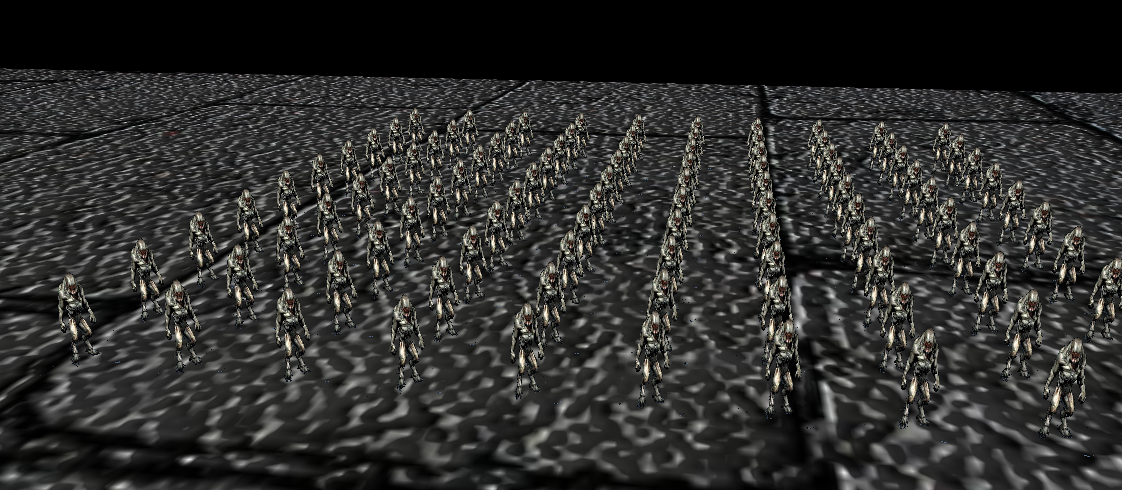


Рисунок 1. Billboard моделей.

1. В 28 уроке я изучил способ создания частиц в OpenGL.

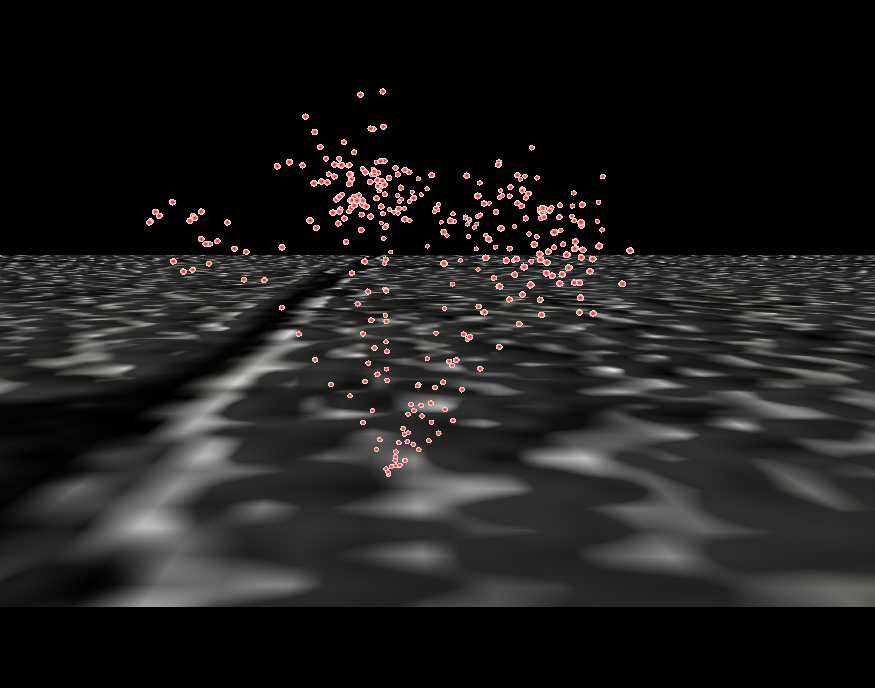


Рисунок 2. Система простых частиц.

1. В урокe 32 я изучил способ работы с Vertex Array Object (или VAO).



Рисунок 3. Использование VAO.

1. В уроке 33 я научился работать с дублирующим рендерингом.

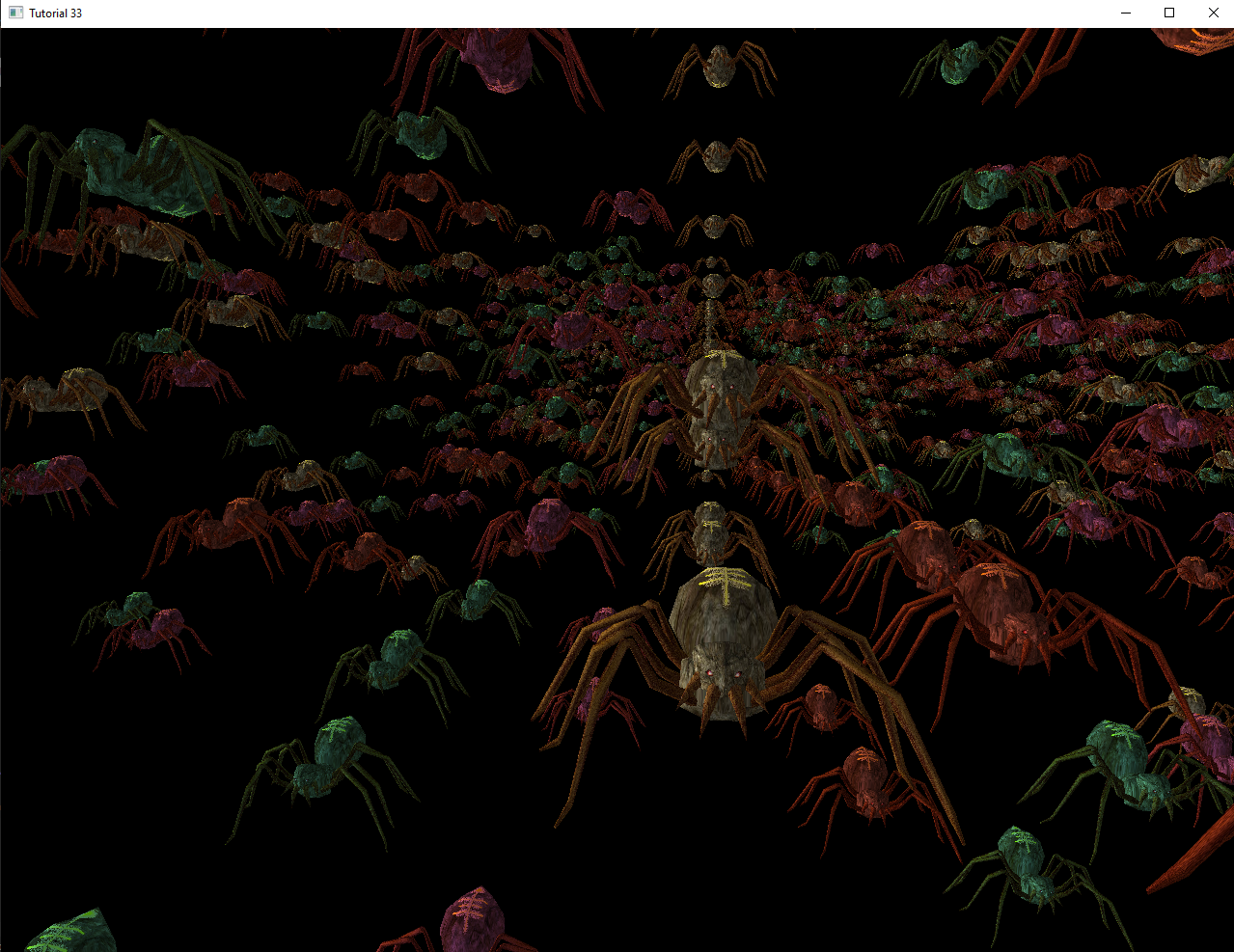


Рисунок 4. Повторяющиеся модели с дублирующим рендерингом.

**Выводы к работе**

При выполнении данной лабораторной работы я познакомился с геометрическим шейдером, частицами , Vertex, Deferred shading в OpenGL. Полный код программ по ссылке: [https://github.com/Ivan122727/EKG/tree/master/LW5](https://github.com/VerVyle/LW_ECG/tree/master/LW5)

**Приложение 1**

**main.cpp (task33)**

#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"

#ifdef FREETYPE

#include "freetypeGL.h"

#endif

#define WINDOW\_WIDTH 1280

#define WINDOW\_HEIGHT 1024

#define NUM\_ROWS 50

#define NUM\_COLS 20

#define NUM\_INSTANCES NUM\_ROWS \* NUM\_COLS

class Tutorial33 : public ICallbacks

{

public:

Tutorial33()

{

m\_pGameCamera = NULL;

m\_pEffect = NULL;

m\_scale = 0.0f;

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

m\_directionalLight.AmbientIntensity = 0.55f;

m\_directionalLight.DiffuseIntensity = 0.9f;

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

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;

m\_pMesh = NULL;

m\_frameCount = 0;

m\_fps = 0.0f;

}

~Tutorial33()

{

SAFE\_DELETE(m\_pEffect);

SAFE\_DELETE(m\_pGameCamera);

SAFE\_DELETE(m\_pMesh);

}

bool Init()

{

Vector3f Pos(7.0f, 3.0f, 0.0f);

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

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

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

m\_pEffect = new LightingTechnique();

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

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

return false;

}

m\_pEffect->Enable();

m\_pEffect->SetColorTextureUnit(COLOR\_TEXTURE\_UNIT\_INDEX);

m\_pEffect->SetDirectionalLight(m\_directionalLight);

m\_pEffect->SetMatSpecularIntensity(0.0f);

m\_pEffect->SetMatSpecularPower(0);

m\_pEffect->SetColor(0, Vector4f(1.0f, 0.5f, 0.5f, 0.0f));

m\_pEffect->SetColor(1, Vector4f(0.5f, 1.0f, 1.0f, 0.0f));

m\_pEffect->SetColor(2, Vector4f(1.0f, 0.5f, 1.0f, 0.0f));

m\_pEffect->SetColor(3, Vector4f(1.0f, 1.0f, 1.0f, 0.0f));

m\_pMesh = new Mesh();

if (!m\_pMesh->LoadMesh("../Content/spider.obj")) {

return false;

}

#ifdef FREETYPE

if (!m\_fontRenderer.InitFontRenderer()) {

return false;

}

#endif

m\_time = glutGet(GLUT\_ELAPSED\_TIME);

CalcPositions();

return true;

}

void Run()

{

GLUTBackendRun(this);

}

virtual void RenderSceneCB()

{

CalcFPS();

m\_scale += 0.005f;

m\_pGameCamera->OnRender();

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

m\_pEffect->Enable();

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

Pipeline p;

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

p.SetPerspectiveProj(m\_persProjInfo);

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

p.Scale(0.005f, 0.005f, 0.005f);

Matrix4f WVPMatrics[NUM\_INSTANCES];

Matrix4f WorldMatrices[NUM\_INSTANCES];

for (unsigned int i = 0 ; i < NUM\_INSTANCES ; i++) {

Vector3f Pos(m\_positions[i]);

Pos.y += sinf(m\_scale) \* m\_velocity[i];

p.WorldPos(Pos);

WVPMatrics[i] = p.GetWVPTrans().Transpose();

WorldMatrices[i] = p.GetWorldTrans().Transpose();

}

m\_pMesh->Render(NUM\_INSTANCES, WVPMatrics, WorldMatrices);

RenderFPS();

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);

}

virtual void MouseCB(int Button, int State, int x, int y)

{

}

private:

void CalcFPS()

{

m\_frameCount++;

int time = glutGet( GLUT\_ELAPSED\_TIME );

if (time - m\_time > 1000) {

m\_fps = (float)m\_frameCount \* 1000.0f / (time - m\_time);

m\_time = time;

m\_frameCount = 0;

}

}

void RenderFPS()

{

char text[32];

SNPRINTF(text, sizeof(text), "FPS: %.2f", m\_fps);

#ifdef FREETYPE

m\_fontRenderer.RenderText(10, 10, text);

#endif

}

void CalcPositions()

{

for (unsigned int i = 0; i < NUM\_ROWS ; i++) {

for (unsigned int j = 0 ; j < NUM\_COLS ; j++) {

unsigned int Index = i \* NUM\_COLS + j;

m\_positions[Index].x = (float)j;

m\_positions[Index].y = RandomFloat() \* 5.0f;

m\_positions[Index].z = (float)i;

m\_velocity[Index] = RandomFloat();

if (i & 1) {

m\_velocity[Index] \*= (-1.0f);

}

}

}

}

LightingTechnique\* m\_pEffect;

Camera\* m\_pGameCamera;

float m\_scale;

DirectionalLight m\_directionalLight;

Mesh\* m\_pMesh;

PersProjInfo m\_persProjInfo;

#ifdef FREETYPE

FontRenderer m\_fontRenderer;

#endif

int m\_time;

int m\_frameCount;

float m\_fps;

Vector3f m\_positions[NUM\_INSTANCES];

float m\_velocity[NUM\_INSTANCES];

};

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

{

Magick::InitializeMagick(\*argv);

GLUTBackendInit(argc, argv);

if (!GLUTBackendCreateWindow(WINDOW\_WIDTH, WINDOW\_HEIGHT, 32, false, "Tutorial 33")) {

return 1;

}

SRANDOM;

Tutorial33\* pApp = new Tutorial33();

if (!pApp->Init()) {

return 1;

}

pApp->Run();

delete pApp;

return 0;

}

**main.cpp (task32)**

#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 Tutorial32 : public ICallbacks

{

public:

Tutorial32()

{

m\_pGameCamera = NULL;

m\_pEffect = NULL;

m\_scale = 0.0f;

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

m\_directionalLight.AmbientIntensity = 0.25f;

m\_directionalLight.DiffuseIntensity = 0.9f;

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

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;

m\_pMesh1 = NULL;

m\_pMesh2 = NULL;

m\_pMesh3 = NULL;

}

~Tutorial32()

{

SAFE\_DELETE(m\_pEffect);

SAFE\_DELETE(m\_pGameCamera);

SAFE\_DELETE(m\_pMesh1);

SAFE\_DELETE(m\_pMesh2);

SAFE\_DELETE(m\_pMesh3);

}

bool Init()

{

Vector3f Pos(3.0f, 7.0f, -10.0f);

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

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

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

m\_pEffect = new LightingTechnique();

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

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

return false;

}

m\_pEffect->Enable();

m\_pEffect->SetColorTextureUnit(COLOR\_TEXTURE\_UNIT\_INDEX);

m\_pEffect->SetDirectionalLight(m\_directionalLight);

m\_pEffect->SetMatSpecularIntensity(0.0f);

m\_pEffect->SetMatSpecularPower(0);

m\_pMesh1 = new Mesh();

if (!m\_pMesh1->LoadMesh("../Content/phoenix\_ugv.md2")) {

return false;

}

m\_pMesh2 = new Mesh();

if (!m\_pMesh2->LoadMesh("../Content/jeep.obj")) {

return false;

}

m\_pMesh3 = new Mesh();

if (!m\_pMesh3->LoadMesh("../Content/hheli.obj")) {

return false;

}

return true;

}

void Run()

{

GLUTBackendRun(this);

}

virtual void RenderSceneCB()

{

m\_scale += 0.01f;

m\_pGameCamera->OnRender();

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

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

Pipeline p;

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

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

p.SetPerspectiveProj(m\_persProjInfo);

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

p.WorldPos(-6.0f, -2.0f, 10.0f);

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

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

m\_pMesh1->Render();

p.Scale(0.01f, 0.01f, 0.01f);

p.WorldPos(6.0f, -2.0f, 10.0f);

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

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

m\_pMesh2->Render();

p.Scale(0.04f, 0.04f, 0.04f);

p.WorldPos(0.0f, 6.0f, 10.0f);

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

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

m\_pMesh3->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);

}

virtual void MouseCB(int Button, int State, int x, int y)

{

}

private:

LightingTechnique\* m\_pEffect;

Camera\* m\_pGameCamera;

float m\_scale;

DirectionalLight m\_directionalLight;

Mesh\* m\_pMesh1;

Mesh\* m\_pMesh2;

Mesh\* m\_pMesh3;

PersProjInfo m\_persProjInfo;

};

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

{

Magick::InitializeMagick(\*argv);

GLUTBackendInit(argc, argv);

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

return 1;

}

Tutorial32\* pApp = new Tutorial32();

if (!pApp->Init()) {

return 1;

}

pApp->Run();

delete pApp;

return 0;

}

**main.cpp (task28)**

#include <stdlib.h>

#include <math.h>

#include <chrono>

#include <assert.h>

#include <GL/glew.h>

#include <GL/freeglut.h>

#ifdef \_\_GNUC\_\_

# if \_\_GNUC\_PREREQ(4,7)

#include <unistd.h>

# endif

#endif

#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"

#include "particle\_system.h"

#define WINDOW\_WIDTH 1920

#define WINDOW\_HEIGHT 1200

static long long GetCurrentTimeMillis() {

auto time = std::chrono::system\_clock::now();

auto since\_epoch = time.time\_since\_epoch();

auto millis = std::chrono::duration\_cast<std::chrono::milliseconds>(since\_epoch);

long long now = millis.count();

return now;

}

class Tutorial28 : public ICallbacks

{

public:

Tutorial28()

{

m\_pLightingTechnique = NULL;

m\_pGameCamera = NULL;

m\_pGround = NULL;

m\_pTexture = NULL;

m\_pNormalMap = 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 = 1.0f;

m\_persProjInfo.zFar = 100.0f;

m\_currentTimeMillis = GetCurrentTimeMillis();

}

~Tutorial28()

{

SAFE\_DELETE(m\_pLightingTechnique);

SAFE\_DELETE(m\_pGameCamera);

SAFE\_DELETE(m\_pGround);

SAFE\_DELETE(m\_pTexture);

SAFE\_DELETE(m\_pNormalMap);

}

bool Init()

{

Vector3f Pos(0.0f, 0.4f, -0.5f);

Vector3f Target(0.0f, 0.2f, 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(COLOR\_TEXTURE\_UNIT\_INDEX);

m\_pLightingTechnique->SetNormalMapTextureUnit(NORMAL\_TEXTURE\_UNIT\_INDEX);

m\_pGround = new Mesh();

if (!m\_pGround->LoadMesh("../Content/quad.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;

}

Vector3f ParticleSystemPos = Vector3f(0.0f, 0.0f, 1.0f);

return m\_particleSystem.InitParticleSystem(ParticleSystemPos);

}

void Run()

{

GLUTBackendRun(this);

}

virtual void RenderSceneCB()

{

long long TimeNowMillis = GetCurrentTimeMillis();

assert(TimeNowMillis >= m\_currentTimeMillis);

unsigned int DeltaTimeMillis = (unsigned int)(TimeNowMillis - m\_currentTimeMillis);

m\_currentTimeMillis = TimeNowMillis;

m\_pGameCamera->OnRender();

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

m\_pLightingTechnique->Enable();

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

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

Pipeline p;

p.Scale(20.0f, 20.0f, 1.0f);

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

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\_pGround->Render();

m\_particleSystem.Render(DeltaTimeMillis, p.GetVPTrans(), m\_pGameCamera->GetPos());

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:

long long m\_currentTimeMillis;

LightingTechnique\* m\_pLightingTechnique;

Camera\* m\_pGameCamera;

DirectionalLight m\_dirLight;

Mesh\* m\_pGround;

Texture\* m\_pTexture;

Texture\* m\_pNormalMap;

PersProjInfo m\_persProjInfo;

ParticleSystem m\_particleSystem;

};

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

{

srand(time(nullptr));

Magick::InitializeMagick(\*argv);

GLUTBackendInit(argc, argv);

if (!GLUTBackendCreateWindow(WINDOW\_WIDTH, WINDOW\_HEIGHT, 32, false, "Tutorial 28")) {

return 1;

}

Tutorial28\* pApp = new Tutorial28();

if (!pApp->Init()) {

return 1;

}

pApp->Run();

delete pApp;

return 0;

}

**main.cpp (task27)**

#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"

#include "billboard\_list.h"

#define WINDOW\_WIDTH 1280

#define WINDOW\_HEIGHT 1024

class Tutorial27 : public ICallbacks

{

public:

Tutorial27()

{

m\_pLightingTechnique = NULL;

m\_pGameCamera = NULL;

m\_pGround = NULL;

m\_pTexture = NULL;

m\_pNormalMap = 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 = 1.0f;

m\_persProjInfo.zFar = 100.0f;

}

~Tutorial27()

{

SAFE\_DELETE(m\_pLightingTechnique);

SAFE\_DELETE(m\_pGameCamera);

SAFE\_DELETE(m\_pGround);

SAFE\_DELETE(m\_pTexture);

SAFE\_DELETE(m\_pNormalMap);

}

bool Init()

{

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

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\_pGround = new Mesh();

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

return false;

}

if (!m\_billboardList.Init("../Content/monster\_hellknight.png")) {

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;

}

return true;

}

void Run()

{

GLUTBackendRun(this);

}

virtual void RenderSceneCB()

{

m\_pGameCamera->OnRender();

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

m\_pLightingTechnique->Enable();

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

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

Pipeline p;

p.Scale(20.0f, 20.0f, 1.0f);

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

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\_pGround->Render();

m\_billboardList.Render(p.GetVPTrans(), m\_pGameCamera->GetPos());

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;

DirectionalLight m\_dirLight;

Mesh\* m\_pGround;

Texture\* m\_pTexture;

Texture\* m\_pNormalMap;

PersProjInfo m\_persProjInfo;

BillboardList m\_billboardList;

};

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

{

Magick::InitializeMagick(\*argv);

GLUTBackendInit(argc, argv);

if (!GLUTBackendCreateWindow(WINDOW\_WIDTH, WINDOW\_HEIGHT, 32, false, "Tutorial 27")) {

return 1;

}

Tutorial27\* pApp = new Tutorial27();

if (!pApp->Init()) {

return 1;

}

pApp->Run();

delete pApp;

return 0;

}