Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут”

Кафедра АСОІУ

**ЗВІТ**

про виконання комп’ютерного практикуму № 8

з дисципліни

“ООП”

Тема: «OpenGL С++»

|  |  |  |
| --- | --- | --- |
| Прийняв: |  | Виконав: |
| Головченко  Максим  Миколайович |  | студент 2-го курсу  гр. ІП-51 ФІОТ  Субцельний Олександр Володимирович |

Київ – 2016

**ЗМІСТ:**

[1 Мета роботи 3](#_Toc463896091)

[2 Постановка задачі (варіант 20, рівень В) 4](#_Toc463896092)

[3 Діаграма класів 5](#_Toc463896093)

[4 Код програми 6](#_Toc463896094)

[5 Приклади виконання програми 10](#_Toc463896095)

[6 Висновок 11](#_Toc463896096)

# Мета роботи

Цель работы – изучить особенности работы OpenGL . Освоить принципы разработки анимации при помощи OpenGL.

# Постановка задачі (варіант 20, рівень В)

Взлет ракеты.

# Код програми

“Display.cpp”

#include <iostream>

#include <SDL2/SDL.h>

#include "display.h"

#include "mesh.h"

#include "shader.h"

#include "texture.h"

#include "transform.h"

#include "camera.h"

static const int DISPLAY\_WIDTH = 1200;

static const int DISPLAY\_HEIGHT = 800;

static const GLfloat g\_vertex\_buffer\_data[] = {

-200, -200, 0,

200, -200, 0,

-200, 200, 0,

200, 200, 0,

-200, 200, 0,

200, -200, 0

};

float xc=1.0f;

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

{

Display display(DISPLAY\_WIDTH, DISPLAY\_HEIGHT, "LAB8\_SUBTSELNYI");

Vertex vertices[] =

{

Vertex(glm::vec3(-1, -1, -1), glm::vec2(1, 0), glm::vec3(0, 0, -1)),

Vertex(glm::vec3(-1, 1, -1), glm::vec2(0, 0), glm::vec3(0, 0, -1)),

Vertex(glm::vec3(1, 1, -1), glm::vec2(0, 1), glm::vec3(0, 0, -1)),

Vertex(glm::vec3(1, -1, -1), glm::vec2(1, 1), glm::vec3(0, 0, -1)),

Vertex(glm::vec3(-1, -1, 1), glm::vec2(1, 0), glm::vec3(0, 0, 1)),

Vertex(glm::vec3(-1, 1, 1), glm::vec2(0, 0), glm::vec3(0, 0, 1)),

Vertex(glm::vec3(1, 1, 1), glm::vec2(0, 1), glm::vec3(0, 0, 1)),

Vertex(glm::vec3(1, -1, 1), glm::vec2(1, 1), glm::vec3(0, 0, 1)),

Vertex(glm::vec3(-1, -1, -1), glm::vec2(0, 1), glm::vec3(0, -1, 0)),

Vertex(glm::vec3(-1, -1, 1), glm::vec2(1, 1), glm::vec3(0, -1, 0)),

Vertex(glm::vec3(1, -1, 1), glm::vec2(1, 0), glm::vec3(0, -1, 0)),

Vertex(glm::vec3(1, -1, -1), glm::vec2(0, 0), glm::vec3(0, -1, 0)),

Vertex(glm::vec3(-1, 1, -1), glm::vec2(0, 1), glm::vec3(0, 1, 0)),

Vertex(glm::vec3(-1, 1, 1), glm::vec2(1, 1), glm::vec3(0, 1, 0)),

Vertex(glm::vec3(1, 1, 1), glm::vec2(1, 0), glm::vec3(0, 1, 0)),

Vertex(glm::vec3(1, 1, -1), glm::vec2(0, 0), glm::vec3(0, 1, 0)),

Vertex(glm::vec3(-1, -1, -1), glm::vec2(1, 1), glm::vec3(-1, 0, 0)),

Vertex(glm::vec3(-1, -1, 1), glm::vec2(1, 0), glm::vec3(-1, 0, 0)),

Vertex(glm::vec3(-1, 1, 1), glm::vec2(0, 0), glm::vec3(-1, 0, 0)),

Vertex(glm::vec3(-1, 1, -1), glm::vec2(0, 1), glm::vec3(-1, 0, 0)),

Vertex(glm::vec3(1, -1, -1), glm::vec2(1, 1), glm::vec3(1, 0, 0)),

Vertex(glm::vec3(1, -1, 1), glm::vec2(1, 0), glm::vec3(1, 0, 0)),

Vertex(glm::vec3(1, 1, 1), glm::vec2(0, 0), glm::vec3(1, 0, 0)),

Vertex(glm::vec3(1, 1, -1), glm::vec2(0, 1), glm::vec3(1, 0, 0)),

};

unsigned int indices[] = {0, 1, 2,

0, 2, 3,

6, 5, 4,

7, 6, 4,

10, 9, 8,

11, 10, 8,

12, 13, 14,

12, 14, 15,

16, 17, 18,

16, 18, 19,

22, 21, 20,

23, 22, 20

};

Mesh mesh(vertices, sizeof(vertices)/sizeof(vertices[0]), indices, sizeof(indices)/sizeof(indices[0]));

Mesh monkey("./res/retro\_rocket.obj");

Mesh m\_background ("./res/hexagon.obj");

Shader shader("./res/basicShader");

Shader shader1("./res/basicShader");

Texture textures("./res/flag.jpg");

Texture background("./res/cos.jpg");

Transform transform;

Transform transform1;

Camera camera(glm::vec3(0.0f, 0.0f, -90.0f), 58.0f, (float)DISPLAY\_WIDTH/(float)DISPLAY\_HEIGHT, 0.1f, 100.0f);

SDL\_Surface \* image = SDL\_LoadBMP("./res/cosmos.bmp");

SDL\_Event e;

bool isRunning = true;

float counter = 0.0f;

float starter = 0.0f;

float actiony = 0.0f;

float mult = 0.000000001f;

float scal = 0.1f;

//if (glfwGetKey( window, GLFW\_KEY\_UP ) == GLFW\_PRESS)

while(isRunning)

{

while(SDL\_PollEvent(&e))

{

if(e.type == SDL\_QUIT)

isRunning = false;

}

//display.m\_glContext = SDL\_GL\_CreateContext(display.m\_window);

display.Clear(0.9f, 0.9f, 0.9f, 1.0f);

/\*

glColor3f(1,1,1);

glEnable(GL\_TEXTURE\_2D);

GLuint texture;

glGenTextures(1, &texture);

glBindTexture(GL\_TEXTURE\_2D,texture);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_NEAREST);

glTexImage2D(GL\_TEXTURE\_2D,0,image->format->BytesPerPixel,image->w,image->h,0,GL\_RGB,GL\_UNSIGNED\_BYTE,image->pixels);

glOrtho(0,640,400,0,-1,1);

glColor3f(1,1,1);

glBindTexture(GL\_TEXTURE\_2D, texture);

glBegin(GL\_QUADS);

glTexCoord2f(0,0);

glVertex3f(0,0,0);

glTexCoord2f(1,0);

glVertex3f(700,0,0);

glTexCoord2f(1,2);

glVertex3f(700,800,0);

glTexCoord2f(0,2);

glVertex3f(0,800,0);

glEnd();

\*/

//laptop

/\* transform1.GetScale()->x = 20.0f;

transform1.GetScale()->y= 20.0f;

transform1.GetScale()->z = 20.0f;

transform1.GetPos()->x = 10.0f ;

transform1.GetPos()->y = -70.0f;\*/

//city

/\* transform1.GetScale()->x = 0.2f;

transform1.GetScale()->y= 0.2f;

transform1.GetScale()->z = 0.1f;

transform1.GetPos()->y = -60.0f;

transform1.GetPos()->x = -10.0f;\*/

//painting

transform1.GetScale()->x = 200.0f;

transform1.GetScale()->y=200.0f;

transform1.GetScale()->z = 80.0f;

transform1.GetRot()->x = 80.0f;

transform1.GetPos()->y = -4.0f;

shader1.Bind();

background.Bind();

shader1.Update(transform1, camera);

m\_background.Draw();

starter+=0.1f;

actiony+=0.0001f;

if (starter>200.0f) scal=scal-0.00001f;

if (starter>20.0f) if (mult<0.5f) mult=mult\*1.01;

if (starter>20.0f) counter=counter+mult/5;

float sinCounter = sinf(counter);

float absSinCounter = abs(sinCounter);

transform.GetPos()->x = 60.0f ;

if(counter<400.0f)

transform.GetRot()->x = counter/600;

transform.GetPos()->y = -30.0f + starter/7;

if (starter>20.0f)

transform.GetPos()->x = 60.0f -counter/50;

//transform.GetRot()->y = counter/800 ;

//transform.GetRot()->z = 0.0f ;

if (starter>200.0f){

transform.GetScale()->x = scal;

transform.GetScale()->y= scal;

transform.GetScale()->z = scal;

}

else{

transform.GetScale()->x = 0.1f;

transform.GetScale()->y= 0.1f;

transform.GetScale()->z = 0.1f;

}

//transform.GetScale()->y = absSinCounter;

// background.Bind();

shader.Bind();

textures.Bind();

shader.Update(transform, camera);

monkey.Draw();

//mesh.Draw();

display.SwapBuffers();

SDL\_Delay(1);

//counter += 0.01f;

}

return 0;

}

“shader.cpp”

#include "shader.h"

#include <iostream>

#include <fstream>

Shader::Shader(const std::string& fileName)

{

m\_program = glCreateProgram();

m\_shaders[0] = CreateShader(LoadShader(fileName + ".vs"), GL\_VERTEX\_SHADER);

m\_shaders[1] = CreateShader(LoadShader(fileName + ".fs"), GL\_FRAGMENT\_SHADER);

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

glAttachShader(m\_program, m\_shaders[i]);

glBindAttribLocation(m\_program, 0, "position");

glBindAttribLocation(m\_program, 1, "texCoord");

glBindAttribLocation(m\_program, 2, "normal");

glLinkProgram(m\_program);

CheckShaderError(m\_program, GL\_LINK\_STATUS, true, "Error linking shader program");

glValidateProgram(m\_program);

CheckShaderError(m\_program, GL\_LINK\_STATUS, true, "Invalid shader program");

m\_uniforms[0] = glGetUniformLocation(m\_program, "MVP");

m\_uniforms[1] = glGetUniformLocation(m\_program, "Normal");

m\_uniforms[2] = glGetUniformLocation(m\_program, "lightDirection");

}

Shader::~Shader()

{

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

{

glDetachShader(m\_program, m\_shaders[i]);

glDeleteShader(m\_shaders[i]);

}

glDeleteProgram(m\_program);

}

void Shader::Bind()

{

glUseProgram(m\_program);

}

void Shader::Update(const Transform& transform, const Camera& camera)

{

glm::mat4 MVP = transform.GetMVP(camera);

glm::mat4 Normal = transform.GetModel();

glUniformMatrix4fv(m\_uniforms[0], 1, GL\_FALSE, &MVP[0][0]);

glUniformMatrix4fv(m\_uniforms[1], 1, GL\_FALSE, &Normal[0][0]);

glUniform3f(m\_uniforms[2], 0.0f, 0.0f, 1.0f);

}

std::string Shader::LoadShader(const std::string& fileName)

{

std::ifstream file;

file.open((fileName).c\_str());

std::string output;

std::string line;

if(file.is\_open())

{

while(file.good())

{

getline(file, line);

output.append(line + "\n");

}

}

else

{

std::cerr << "Unable to load shader: " << fileName << std::endl;

}

return output;

}

void Shader::CheckShaderError(GLuint shader, GLuint flag, bool isProgram, const std::string& errorMessage)

{

GLint success = 0;

GLchar error[1024] = { 0 };

if(isProgram)

glGetProgramiv(shader, flag, &success);

else

glGetShaderiv(shader, flag, &success);

if(success == GL\_FALSE)

{

if(isProgram)

glGetProgramInfoLog(shader, sizeof(error), NULL, error);

else

glGetShaderInfoLog(shader, sizeof(error), NULL, error);

std::cerr << errorMessage << ": '" << error << "'" << std::endl;

}

}

GLuint Shader::CreateShader(const std::string& text, unsigned int type)

{

GLuint shader = glCreateShader(type);

if(shader == 0)

std::cerr << "Error compiling shader type " << type << std::endl;

const GLchar\* p[1];

p[0] = text.c\_str();

GLint lengths[1];

lengths[0] = text.length();

glShaderSource(shader, 1, p, lengths);

glCompileShader(shader);

CheckShaderError(shader, GL\_COMPILE\_STATUS, false, "Error compiling shader!");

return shader;

}

“main.cpp”

#include <iostream>

#include <SDL2/SDL.h>

#include "display.h"

#include "mesh.h"

#include "shader.h"

#include "texture.h"

#include "transform.h"

#include "camera.h"

static const int DISPLAY\_WIDTH = 1200;

static const int DISPLAY\_HEIGHT = 800;

static const GLfloat g\_vertex\_buffer\_data[] = {

-200, -200, 0,

200, -200, 0,

-200, 200, 0,

200, 200, 0,

-200, 200, 0,

200, -200, 0

};

float xc=1.0f;

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

{

Display display(DISPLAY\_WIDTH, DISPLAY\_HEIGHT, "LAB8\_SUBTSELNYI");

Vertex vertices[] =

{

Vertex(glm::vec3(-1, -1, -1), glm::vec2(1, 0), glm::vec3(0, 0, -1)),

Vertex(glm::vec3(-1, 1, -1), glm::vec2(0, 0), glm::vec3(0, 0, -1)),

Vertex(glm::vec3(1, 1, -1), glm::vec2(0, 1), glm::vec3(0, 0, -1)),

Vertex(glm::vec3(1, -1, -1), glm::vec2(1, 1), glm::vec3(0, 0, -1)),

Vertex(glm::vec3(-1, -1, 1), glm::vec2(1, 0), glm::vec3(0, 0, 1)),

Vertex(glm::vec3(-1, 1, 1), glm::vec2(0, 0), glm::vec3(0, 0, 1)),

Vertex(glm::vec3(1, 1, 1), glm::vec2(0, 1), glm::vec3(0, 0, 1)),

Vertex(glm::vec3(1, -1, 1), glm::vec2(1, 1), glm::vec3(0, 0, 1)),

Vertex(glm::vec3(-1, -1, -1), glm::vec2(0, 1), glm::vec3(0, -1, 0)),

Vertex(glm::vec3(-1, -1, 1), glm::vec2(1, 1), glm::vec3(0, -1, 0)),

Vertex(glm::vec3(1, -1, 1), glm::vec2(1, 0), glm::vec3(0, -1, 0)),

Vertex(glm::vec3(1, -1, -1), glm::vec2(0, 0), glm::vec3(0, -1, 0)),

Vertex(glm::vec3(-1, 1, -1), glm::vec2(0, 1), glm::vec3(0, 1, 0)),

Vertex(glm::vec3(-1, 1, 1), glm::vec2(1, 1), glm::vec3(0, 1, 0)),

Vertex(glm::vec3(1, 1, 1), glm::vec2(1, 0), glm::vec3(0, 1, 0)),

Vertex(glm::vec3(1, 1, -1), glm::vec2(0, 0), glm::vec3(0, 1, 0)),

Vertex(glm::vec3(-1, -1, -1), glm::vec2(1, 1), glm::vec3(-1, 0, 0)),

Vertex(glm::vec3(-1, -1, 1), glm::vec2(1, 0), glm::vec3(-1, 0, 0)),

Vertex(glm::vec3(-1, 1, 1), glm::vec2(0, 0), glm::vec3(-1, 0, 0)),

Vertex(glm::vec3(-1, 1, -1), glm::vec2(0, 1), glm::vec3(-1, 0, 0)),

Vertex(glm::vec3(1, -1, -1), glm::vec2(1, 1), glm::vec3(1, 0, 0)),

Vertex(glm::vec3(1, -1, 1), glm::vec2(1, 0), glm::vec3(1, 0, 0)),

Vertex(glm::vec3(1, 1, 1), glm::vec2(0, 0), glm::vec3(1, 0, 0)),

Vertex(glm::vec3(1, 1, -1), glm::vec2(0, 1), glm::vec3(1, 0, 0)),

};

unsigned int indices[] = {0, 1, 2,

0, 2, 3,

6, 5, 4,

7, 6, 4,

10, 9, 8,

11, 10, 8,

12, 13, 14,

12, 14, 15,

16, 17, 18,

16, 18, 19,

22, 21, 20,

23, 22, 20

};

Mesh mesh(vertices, sizeof(vertices)/sizeof(vertices[0]), indices, sizeof(indices)/sizeof(indices[0]));

Mesh monkey("./res/retro\_rocket.obj");

Mesh m\_background ("./res/hexagon.obj");

Shader shader("./res/basicShader");

Shader shader1("./res/basicShader");

Texture textures("./res/flag.jpg");

Texture background("./res/cos.jpg");

Transform transform;

Transform transform1;

Camera camera(glm::vec3(0.0f, 0.0f, -90.0f), 58.0f, (float)DISPLAY\_WIDTH/(float)DISPLAY\_HEIGHT, 0.1f, 100.0f);

SDL\_Surface \* image = SDL\_LoadBMP("./res/cosmos.bmp");

SDL\_Event e;

bool isRunning = true;

float counter = 0.0f;

float starter = 0.0f;

float actiony = 0.0f;

float mult = 0.000000001f;

float scal = 0.1f;

//if (glfwGetKey( window, GLFW\_KEY\_UP ) == GLFW\_PRESS)

while(isRunning)

{

while(SDL\_PollEvent(&e))

{

if(e.type == SDL\_QUIT)

isRunning = false;

}

//display.m\_glContext = SDL\_GL\_CreateContext(display.m\_window);

display.Clear(0.9f, 0.9f, 0.9f, 1.0f);

//painting

transform1.GetScale()->x = 200.0f;

transform1.GetScale()->y=200.0f;

transform1.GetScale()->z = 80.0f;

transform1.GetRot()->x = 80.0f;

transform1.GetPos()->y = -4.0f;

shader1.Bind();

background.Bind();

shader1.Update(transform1, camera);

m\_background.Draw();

starter+=0.1f;

actiony+=0.0001f;

if (starter>200.0f) scal=scal-0.00001f;

if (starter>20.0f) if (mult<0.5f) mult=mult\*1.01;

if (starter>20.0f) counter=counter+mult/5;

float sinCounter = sinf(counter);

float absSinCounter = abs(sinCounter);

transform.GetPos()->x = 60.0f ;

if(counter<400.0f)

transform.GetRot()->x = counter/600;

transform.GetPos()->y = -30.0f + starter/7;

if (starter>20.0f)

transform.GetPos()->x = 60.0f -counter/50;

//transform.GetRot()->y = counter/800 ;

//transform.GetRot()->z = 0.0f ;

if (starter>200.0f){

transform.GetScale()->x = scal;

transform.GetScale()->y= scal;

transform.GetScale()->z = scal;

}

else{

transform.GetScale()->x = 0.1f;

transform.GetScale()->y= 0.1f;

transform.GetScale()->z = 0.1f;

}

//transform.GetScale()->y = absSinCounter;

// background.Bind();

shader.Bind();

textures.Bind();

shader.Update(transform, camera);

monkey.Draw();

//mesh.Draw();

display.SwapBuffers();

SDL\_Delay(1);

//counter += 0.01f;

}

return 0;

}

“camera.h”

#ifndef CAMERA\_INCLUDED\_H

#define CAMERA\_INCLUDED\_H

#include <glm/glm.hpp>

#include <glm/gtx/transform.hpp>

struct Camera

{

public:

Camera(const glm::vec3& pos, float fov, float aspect, float zNear, float zFar)

{

this->pos = pos;

this->forward = glm::vec3(0.0f, 0.0f, 1.0f);

this->up = glm::vec3(0.0f, 1.0f, 0.0f);

this->projection = glm::perspective(fov, aspect, zNear, zFar);

}

inline glm::mat4 GetViewProjection() const

{

return projection \* glm::lookAt(pos, pos + forward, up);

}

//void MoveForward(float amt)

//{

// pos += forward \* amt;

//}

//void MoveRight(float amt)

//{

// pos += glm::cross(up, forward) \* amt;

//}

//void Pitch(float angle)

//{

// glm::vec3 right = glm::normalize(glm::cross(up, forward));

// forward = glm::vec3(glm::normalize(glm::rotate(angle, right) \* glm::vec4(forward, 0.0)));

// up = glm::normalize(glm::cross(forward, right));

//}

//void RotateY(float angle)

//{

// static const glm::vec3 UP(0.0f, 1.0f, 0.0f);

// glm::mat4 rotation = glm::rotate(angle, UP);

// forward = glm::vec3(glm::normalize(rotation \* glm::vec4(forward, 0.0)));

// up = glm::vec3(glm::normalize(rotation \* glm::vec4(up, 0.0)));

//}

protected:

private:

glm::mat4 projection;

glm::vec3 pos;

glm::vec3 forward;

glm::vec3 up;

};

#endif

“mesh.cpp”

#include "mesh.h"

#include "util.h"

#include "debugTimer.h"

#define GLEW\_STATIC

#include <map>

#include <algorithm>

#include <fstream>

#include <iostream>

#include <stdlib.h>

Mesh::Mesh(const std::string& fileName)

{

InitMesh(OBJModel(fileName).ToIndexedModel());

}

void Mesh::InitMesh(const IndexedModel& model)

{

m\_numIndices = model.indices.size();

glGenVertexArrays(1, &m\_vertexArrayObject);

glBindVertexArray(m\_vertexArrayObject);

glGenBuffers(NUM\_BUFFERS, m\_vertexArrayBuffers);

glBindBuffer(GL\_ARRAY\_BUFFER, m\_vertexArrayBuffers[POSITION\_VB]);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(model.positions[0]) \* model.positions.size(), &model.positions[0], GL\_STATIC\_DRAW);

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 0, 0);

glBindBuffer(GL\_ARRAY\_BUFFER, m\_vertexArrayBuffers[TEXCOORD\_VB]);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(model.texCoords[0]) \* model.texCoords.size(), &model.texCoords[0], GL\_STATIC\_DRAW);

glEnableVertexAttribArray(1);

glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, 0, 0);

glBindBuffer(GL\_ARRAY\_BUFFER, m\_vertexArrayBuffers[NORMAL\_VB]);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(model.normals[0]) \* model.normals.size(), &model.normals[0], GL\_STATIC\_DRAW);

glEnableVertexAttribArray(2);

glVertexAttribPointer(2, 3, GL\_FLOAT, GL\_FALSE, 0, 0);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, m\_vertexArrayBuffers[INDEX\_VB]);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, sizeof(model.indices[0]) \* model.indices.size(), &model.indices[0], GL\_STATIC\_DRAW);

glBindVertexArray(0);

}

Mesh::Mesh(Vertex\* vertices, unsigned int numVertices, unsigned int\* indices, unsigned int numIndices)

{

IndexedModel model;

for(unsigned int i = 0; i < numVertices; i++)

{

model.positions.push\_back(\*vertices[i].GetPos());

model.texCoords.push\_back(\*vertices[i].GetTexCoord());

model.normals.push\_back(\*vertices[i].GetNormal());

}

for(unsigned int i = 0; i < numIndices; i++)

model.indices.push\_back(indices[i]);

InitMesh(model);

}

Mesh::~Mesh()

{

glDeleteBuffers(NUM\_BUFFERS, m\_vertexArrayBuffers);

glDeleteVertexArrays(1, &m\_vertexArrayObject);

}

void Mesh::Draw()

{

glBindVertexArray(m\_vertexArrayObject);

//glDrawElements(GL\_TRIANGLES, m\_numIndices, GL\_UNSIGNED\_INT, 0);

glDrawElementsBaseVertex(GL\_TRIANGLES, m\_numIndices, GL\_UNSIGNED\_INT, 0, 0);

glBindVertexArray(0);

}

“texture.cpp”

#include "texture.h"

#include "stb\_image.h"

#include <iostream>

Texture::Texture(const std::string& fileName)

{

int width, height, numComponents;

unsigned char\* data = stbi\_load((fileName).c\_str(), &width, &height, &numComponents, 4);

if(data == NULL)

std::cerr << "Unable to load texture: " << fileName << std::endl;

glGenTextures(1, &m\_texture);

glBindTexture(GL\_TEXTURE\_2D, m\_texture);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, width, height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, data);

stbi\_image\_free(data);

}

Texture::~Texture()

{

glDeleteTextures(1, &m\_texture);

}

void Texture::Bind()

{

glBindTexture(GL\_TEXTURE\_2D, m\_texture);

}

#endif // ORDER\_H

“texture.h”

#ifndef TEXTURE\_H

#define TEXTURE\_H

#include <string>

#include <GL/glew.h>

class Texture

{

public:

Texture(const std::string& fileName);

void Bind();

virtual ~Texture();

protected:

private:

Texture(const Texture& texture) {}

void operator=(const Texture& texture) {}

GLuint m\_texture;

};

#endif

“mesh.h”

#ifndef MESH\_INCLUDED\_H

#define MESH\_INCLUDED\_H

#define GLEW\_STATIC

#include <GL/glew.h>

#include <glm/glm.hpp>

#include <string>

#include <vector>

#include "obj\_loader.h"

struct Vertex

{

public:

Vertex(const glm::vec3& pos, const glm::vec2& texCoord, const glm::vec3& normal)

{

this->pos = pos;

this->texCoord = texCoord;

this->normal = normal;

}

glm::vec3\* GetPos() { return &pos; }

glm::vec2\* GetTexCoord() { return &texCoord; }

glm::vec3\* GetNormal() { return &normal; }

private:

glm::vec3 pos;

glm::vec2 texCoord;

glm::vec3 normal;

};

enum MeshBufferPositions

{

POSITION\_VB,

TEXCOORD\_VB,

NORMAL\_VB,

INDEX\_VB

};

class Mesh

{

public:

Mesh(const std::string& fileName);

Mesh(Vertex\* vertices, unsigned int numVertices, unsigned int\* indices, unsigned int numIndices);

void Draw();

void Draw\_b(const GLfloat[]);

virtual ~Mesh();

protected:

private:

static const unsigned int NUM\_BUFFERS = 4;

void operator=(const Mesh& mesh) {}

Mesh(const Mesh& mesh) {}

void InitMesh(const IndexedModel& model);

GLuint m\_vertexArrayObject;

GLuint m\_vertexArrayBuffers[NUM\_BUFFERS];

unsigned int m\_numIndices;

};

#endif

“shader.h”

#ifndef SHADER\_INCLUDED\_H

#define SHADER\_INCLUDED\_H

#define GLEW\_STATIC

#include <string>

#include <GL/glew.h>

#include "transform.h"

class Shader

{

public:

Shader(const std::string& fileName);

void Bind();

void Update(const Transform& transform, const Camera& camera);

virtual ~Shader();

protected:

private:

static const unsigned int NUM\_SHADERS = 2;

static const unsigned int NUM\_UNIFORMS = 3;

void operator=(const Shader& shader) {}

Shader(const Shader& shader) {}

std::string LoadShader(const std::string& fileName);

void CheckShaderError(GLuint shader, GLuint flag, bool isProgram, const std::string& errorMessage);

GLuint CreateShader(const std::string& text, unsigned int type);

GLuint m\_program;

GLuint m\_shaders[NUM\_SHADERS];

GLuint m\_uniforms[NUM\_UNIFORMS];

};

#endif

“Display.h”

#ifndef SHADER\_INCLUDED\_H

#define SHADER\_INCLUDED\_H

#define GLEW\_STATIC

#include <string>

#include <GL/glew.h>

#include "transform.h"

class Shader

{

public:

Shader(const std::string& fileName);

void Bind();

void Update(const Transform& transform, const Camera& camera);

virtual ~Shader();

protected:

private:

static const unsigned int NUM\_SHADERS = 2;

static const unsigned int NUM\_UNIFORMS = 3;

void operator=(const Shader& shader) {}

Shader(const Shader& shader) {}

std::string LoadShader(const std::string& fileName);

void CheckShaderError(GLuint shader, GLuint flag, bool isProgram, const std::string& errorMessage);

GLuint CreateShader(const std::string& text, unsigned int type);

GLuint m\_program;

GLuint m\_shaders[NUM\_SHADERS];

GLuint m\_uniforms[NUM\_UNIFORMS];

};

#endif

}

# Приклади виконання програми

Приклад виконання програми наведений на рисунку 5.1, рисунку 5.2:

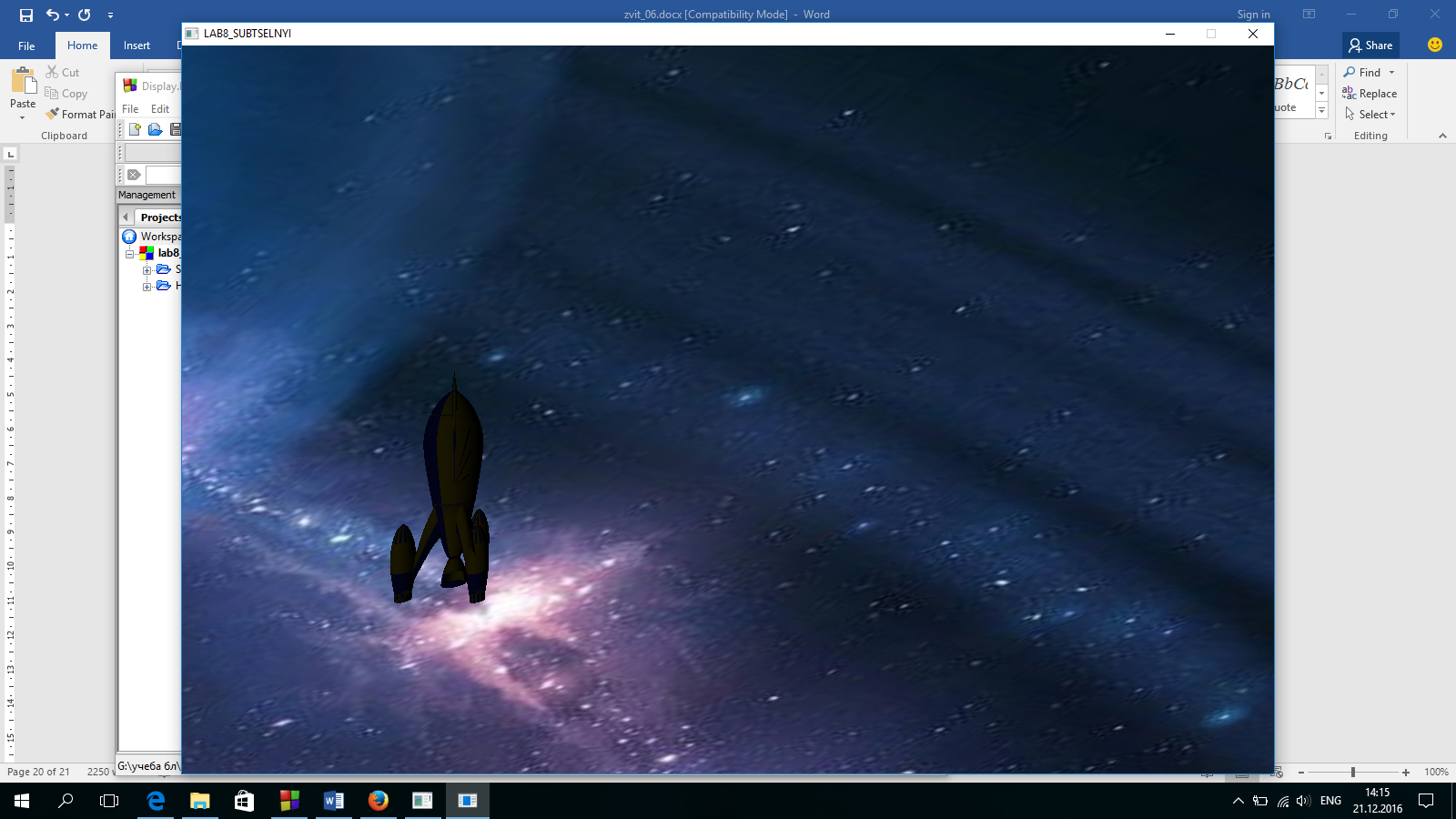


Рисунок 5.1 – Приклад виконання програми

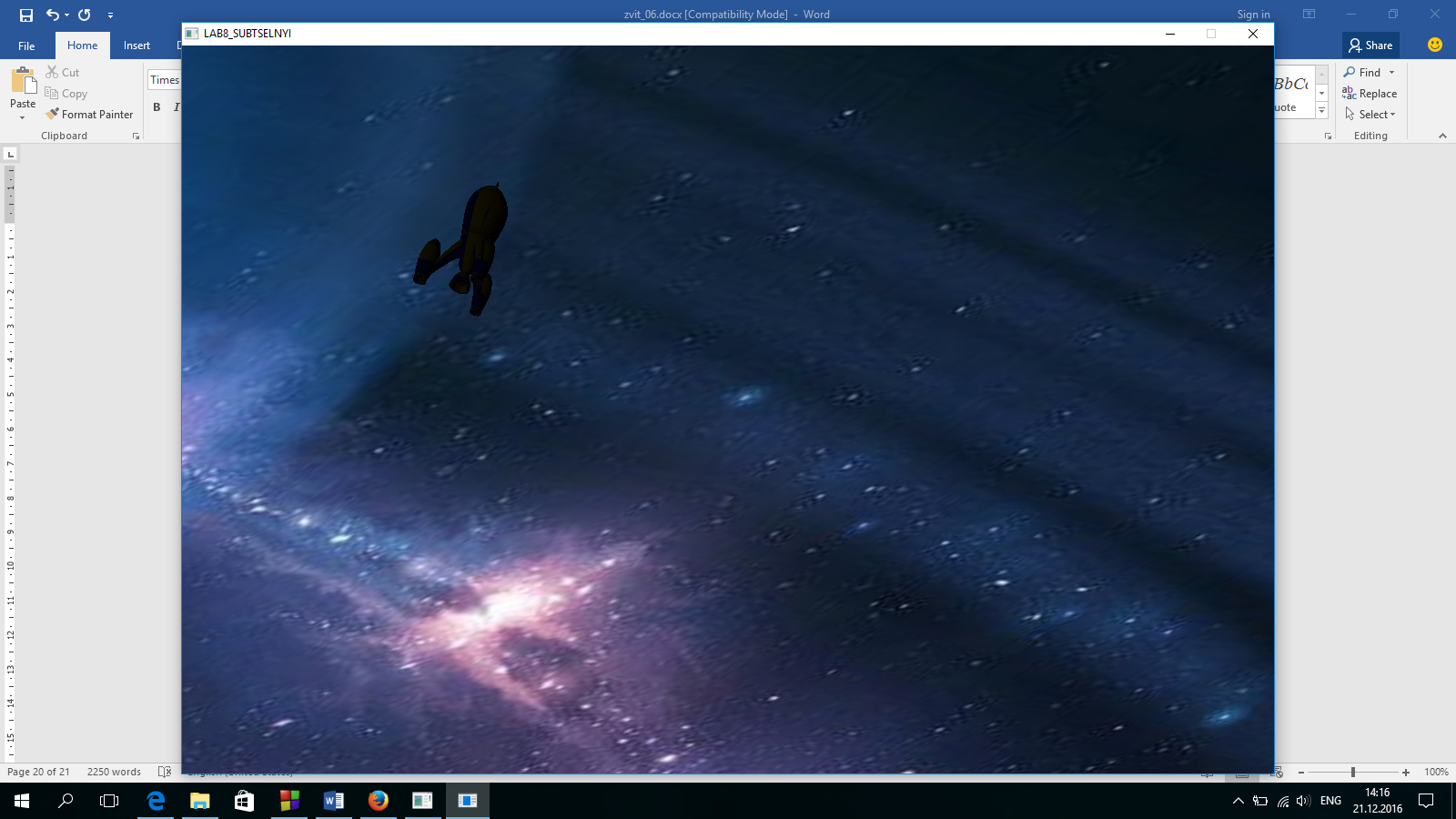


Рисунок 5.2 – Приклад виконання програми

# Висновок

Отже, дана програма ємулює взлету ракети. На фон можна накладати різні текстури різних форматів. Ракета є 3Dобєктом, яка замальована в текстуру українского прапора. При взліті ракета спочатку підіймається в повітря, потім повертається по осі і направляється в космос.