**Міністерство освіти і науки України**

**Запорізький національний технічний університет**

кафедра програмних засобів

**ЗВІТ**

з лабораторної роботи №3

з дисципліни «Методи та алгоритми комп'ютерної графіки» на тему:

"Накладення текстури та освітлення"

Варіант 22

Виконала:

студентка групи КНТ-415 В.С. Хохлова

Прийняв:

асистент Ж.К. Камінська

2018

1. Мета роботи

Навчитися накладати текстури та освітлення на об’єкти.

1. Завдання

Використовуючи різні графічні примітиви намалювати 3d- об’єкт призму або піраміду, в основі якої лежить багатокутник з кількістю вершин 24 і накласти на нього довільну текстуру.

1. Хід роботи

#include "my\_paint.h"

#include <QtOpenGL>

#include <GL/gl.h>

#include <GL/glut.h>

#include <iostream>

#include <complex>

#include <vector>

#include <math.h>

#define PI 3.141592653f

#define STB\_IMAGE\_IMPLEMENTATION

#include "stb\_image.h"

typedef double T\_coord;

typedef std::complex<T\_coord> T\_point;

typedef std::vector<T\_point> T\_figure;

GLuint texture;

int light\_sample = 1;

void **DrawPolygon**(int n){

T\_point center(0.0, 0.0);

T\_point vertex(0.0, 1.0);

T\_figure pyramid;

pyramid.push\_back(vertex);

// Розраховуємо координати многокутника

T\_point radius\_next = vertex - center;

for(int i = 0; i < n - 1; ++i)

{

radius\_next \*= std::polar(1.0, acos(-1.0) \* 2 / n );

T\_point vertex\_next = center + radius\_next;

pyramid.push\_back(vertex\_next);

}

// Малюємо основу - многокутник

glBegin(GL\_POLYGON);

glColor3f(1.0f, 1.0f, 1.0f);

for(int i = 0; i < n; i++) {

glTexCoord2f((1-pyramid[i].real())/2, (1+pyramid[i].imag())/2);

glVertex3f(pyramid[i].real(), -1.0f, pyramid[i].imag());

}

glEnd();

// Малюємо грані піраміди - трикутники

glBegin(GL\_TRIANGLES);

glColor3f(0.0f, 0.0f, 1.0f);

for(int i = 0; i < n - 1; i++) {

glVertex3f( pyramid[i].real(), -1.0f, pyramid[i].imag());

glVertex3f( 0.0f, 1.0f, 0.0f);

glVertex3f( pyramid[i+1].real(), -1.0f, pyramid[i+1].imag());

glColor3f(0.0f+i\*1.0f/n, 0.0f+i\*1.0f/n, 1.0f);

}

glVertex3f( pyramid[n-1].real(), -1.0f, pyramid[n-1].imag());

glVertex3f( 0.0f, 1.0f, 0.0f);

glVertex3f( pyramid[0].real(), -1.0f, pyramid[0].imag());

glEnd();

}

void **LoadTextures**() {

glGenTextures(1, &texture);

glBindTexture(GL\_TEXTURE\_2D, texture);

// set the texture wrapping/filtering options (on the currently bound texture object)

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

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

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

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

// load and generate the texture

int width, height, nrChannels;

unsigned char \*data = stbi\_load("sea.jpg", &width, &height, &nrChannels, 0);

if (data) {

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, width, height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, data);

}

else {

std::cout << "Failed to load texture" << std::endl;

}

stbi\_image\_free(data);

}

void **light**(void) {

// властивости материалу

GLfloat material\_diffuse[] = {1.0, 1.0, 1.0, 1.0};

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, material\_diffuse);

// встановлення джерела світла

if (light\_sample == 1) {

// направлене джерело світла

GLfloat light0\_diffuse[] = {1.5f, 1.5f, 1.5f};

GLfloat light0\_direction[] = {0.0, 0.0, 1.0, 0.0};

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light0\_diffuse);

glLightfv(GL\_LIGHT0, GL\_POSITION, light0\_direction);

}

if (light\_sample == 2) {

// точкове джерело світла

// спадання інтенсивності з відстанню відключено (за замовчуванням)

GLfloat light1\_diffuse[] = {1.5f, 1.5f, 1.5f};

GLfloat light1\_position[] = {0.0, 0.0, 1.0, 1.0};

glEnable(GL\_LIGHT1);

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, light1\_diffuse);

glLightfv(GL\_LIGHT1, GL\_POSITION, light1\_position);

}

if (light\_sample == 6) {

// декілька джерел світла

GLfloat light5\_diffuse[] = {1.0, 0.0, 0.0};

GLfloat light5\_position[] = {0.5f \* cos(0.0f), 0.5f \* sin(0.0f), 1.0, 1.0};

glEnable(GL\_LIGHT5);

glLightfv(GL\_LIGHT5, GL\_DIFFUSE, light5\_diffuse);

glLightfv(GL\_LIGHT5, GL\_POSITION, light5\_position);

glLightf(GL\_LIGHT5, GL\_CONSTANT\_ATTENUATION, 0.0);

glLightf(GL\_LIGHT5, GL\_LINEAR\_ATTENUATION, 0.4f);

glLightf(GL\_LIGHT5, GL\_QUADRATIC\_ATTENUATION, 0.8f);

GLfloat light6\_diffuse[] = {0.0, 1.0, 0.0};

GLfloat light6\_position[] = {0.5f \* cos(2.0f \* PI / 3.0f), 0.5f \* sin(2.0f \* PI / 3.0f), 1.0, 1.0};

glEnable(GL\_LIGHT6);

glLightfv(GL\_LIGHT6, GL\_DIFFUSE, light6\_diffuse);

glLightfv(GL\_LIGHT6, GL\_POSITION, light6\_position);

glLightf(GL\_LIGHT6, GL\_CONSTANT\_ATTENUATION, 0.0);

glLightf(GL\_LIGHT6, GL\_LINEAR\_ATTENUATION, 0.4f);

glLightf(GL\_LIGHT6, GL\_QUADRATIC\_ATTENUATION, 0.8f);

GLfloat light7\_diffuse[] = {0.0, 0.0, 1.0};

GLfloat light7\_position[] = {0.5f \* cos(4.0f \* PI / 3), 0.5f \* sin(4 \* PI / 3), 1.0, 1.0};

glEnable(GL\_LIGHT7);

glLightfv(GL\_LIGHT7, GL\_DIFFUSE, light7\_diffuse);

glLightfv(GL\_LIGHT7, GL\_POSITION, light7\_position);

glLightf(GL\_LIGHT7, GL\_CONSTANT\_ATTENUATION, 0.0);

glLightf(GL\_LIGHT7, GL\_LINEAR\_ATTENUATION, 0.4f);

glLightf(GL\_LIGHT7, GL\_QUADRATIC\_ATTENUATION, 0.8f);

}

glNormal3f(1.0, -1.0, -1.0);

DrawPolygon(24);

// відключення джерел світла

glDisable(GL\_LIGHT0);

glDisable(GL\_LIGHT1);

glDisable(GL\_LIGHT2);

glDisable(GL\_LIGHT6);

glDisable(GL\_LIGHT7);

}

My\_Paint::**My\_Paint**(QWidget \*parent) // конструктор

: QGLWidget(parent) {

resize(1000, 600);

//resize(1000, 600);

xRotation = 0;

yRotation = 0;

zRotation = 0;

scale = 1;

}

void My\_Paint::***initializeGL***() {

LoadTextures(); // Загрузка текстур

glEnable(GL\_TEXTURE\_2D); // Разрешение наложение текстуры

// розрахунок освітлення

glEnable(GL\_LIGHTING);

// двухсторонній розрахунок освітлення

glLightModelf(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_TRUE);

// автоматичне приведення нормалей до

// одиничної довжини

glEnable(GL\_NORMALIZE);

qglClearColor(QColor(245, 245, 208)); // заполняем фон

glEnable(GL\_DEPTH\_TEST); // задаем глубину проверки пикселей - будут ли видны противоположные грани

glShadeModel(GL\_FLAT); // убираем режим сглаживания цветов

glPolygonMode(GL\_BACK, GL\_FRONT\_FACE); // фигуры будут закрашены с обеих сторон

// Дозволяємо очищення буфера глибини

glClearDepth (1.0); // Дозволяємо очищення буфера глибини

// Тип тесту глибини

glDepthFunc (GL\_LEQUAL);

// Поліпшення в обчисленні перспективи

glHint (GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST);

// Дозволяємо змішування

glEnable (GL\_BLEND);

// Вказуємо спосіб змішування

glBlendFunc (GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glClearDepth(1.0); // Set background depth to farthest

glShadeModel(GL\_SMOOTH); // Enable smooth shading

}

void My\_Paint::***resizeGL***(int nWidth, int nHeight) {

glViewport(0, 0, nHeight, nWidth); // установка точки обзора

glMatrixMode(GL\_PROJECTION); // установка режима матрицы

glLoadIdentity(); // загрузка матрицы

}

void My\_Paint::***paintGL***() {

*resizeGL*(800, 1000);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // очистка экрана

glMatrixMode(GL\_MODELVIEW); // задаем модельно-видовую матрицу

glLoadIdentity(); // загрузка единичную матрицу

glScalef(0.5, 0.5, 0.5); // масштабирование

glRotatef(xRotation, 1.0f, 0.0f, 0.0f); // поворот по X

glRotatef(yRotation, 0.0f, 1.0f, 0.0f); // поворот по Y

glRotatef(zRotation, 0.0f, 0.0f, 1.0f); // поворот по Z

light();

}

void My\_Paint::***keyPressEvent***(QKeyEvent \*event) {

if (event->key() == Qt::Key\_1) light\_sample = 1;

if (event->key() == Qt::Key\_2) light\_sample = 2;

if (event->key() == Qt::Key\_3) light\_sample = 3;

if (event->key() == Qt::Key\_4) light\_sample = 4;

if (event->key() == Qt::Key\_5) light\_sample = 5;

if (event->key() == Qt::Key\_6) light\_sample = 6;

*updateGL*(); // обновляем изображение

}

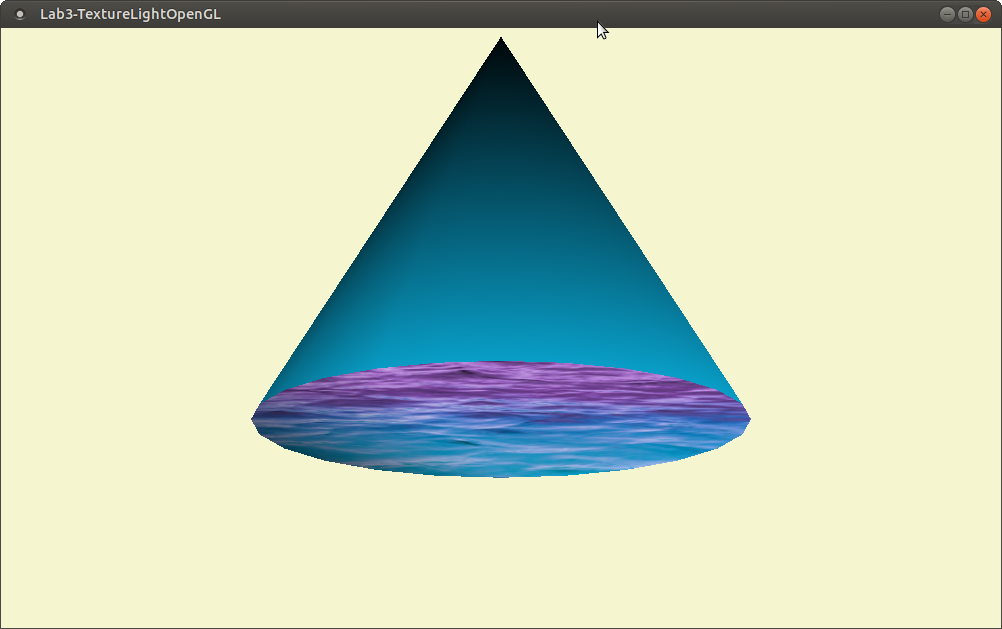


Рисунок 3.1 – Малюнок з використанням OpenGL

1. Висновки

Виконуючи лабораторну роботу, я навчилася накладати текстури та освітлення на об’єкти.