Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут”

Кафедра АСОІУ

ЗВІТ

про виконання комп’ютерного практикуму № 8

з дисципліни

“Об’єктно-орієнтоване програмування”

Тема: OPENGL С++

|  |  |  |
| --- | --- | --- |
| Прийняв: |  | Виконав: |
| Головченко М.М. |  | студент 2-го курсу  гр. ІП-52 ФІОТ  Набоков Е.М |

Київ 2016

ЗМІСТ

[1 ПОСТАНОВКА ЗАДАЧІPAGEREF \_Toc462750603 \h3](#_Toc462750603)

[2 ТЕОРЕТИЧНІ ВІДОМОСТІPAGEREF \_Toc462750604 \h4](#_Toc462750604)

[ЗМІСТ 2](#_Toc470092860)

[1 ПОСТАНОВКА ЗАДАЧІ 3](#_Toc470092861)

[2 ТЕОРЕТИЧНІ ВІДОМОСТІ 4](#_Toc470092862)

[3 ВИХІДНИЙ КОД 5](#_Toc470092863)

[4 ВІДЕОКОПІЯ РЕЗУЛЬТАТУ 11](#_Toc470092864)

[ВИСНОВОК 13](#_Toc470092865)

[3 АЛГОРИТМPAGEREF \_Toc462750605 \h5](#_Toc462750605)

[4 ДІАГРАММА КЛАСІВPAGEREF \_Toc462750606 \h8](#_Toc462750606)

[5 ВИХІДНИЙ КОДPAGEREF \_Toc462750607 \h9](#_Toc462750607)

[6 ВІДЕОКОПІЯ РЕЗУЛЬТАТУPAGEREF \_Toc462750608 \h20](#_Toc462750608)

[7 ВИСНОВОКPAGEREF \_Toc462750609 \h21](#_Toc462750609)

1 ПОСТАНОВКА ЗАДАЧІ

Выполнить программную реализацию задания согласно варианту. В программе обязательно предусмотреть вывод информации об исполнителе работы (ФИО), номере варианта, выбранном уровне сложности и задании.

Ключевые моменты программы обязательно должны содержать комментарии. Используя возможности OpenGL выполнить анимацию согласно варианту. Внимание! Рекомендуется вместо freeglut использовать библиотеку GLFW для работы с текстурами и шейдерами.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

OpenGL (Open Graphics Library) - спецификация, определяющая независимый от языка программирования платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику. Включает более 300 функций для рисования сложных трёхмерных сцен из простых примитивов. Используется при создании компьютерных игр, САПР, виртуальной реальности, визуализации в научных исследованиях. На платформе Windows конкурирует с Direct3D. На базовом уровне, OpenGL - это просто спецификация, то есть документ, описывающий набор функций и их точное поведение. Производители оборудования на основе этой спецификации создают реализации - библиотеки функций, соответствующих набору функций спецификации. Реализация призвана эффективно использовать возможности оборудования. Если аппаратура не позволяет реализовать какую-либо возможность, она должна быть эмулирована программно. Производители должны пройти специфические тесты (conformance tests - тесты на соответствие) прежде чем реализация будет классифицирована как OpenGL- реализация. Таким образом, разработчикам программного обеспечения достаточно научиться использовать функции, описанные в спецификации, оставив эффективную реализацию последних разработчикам аппаратного обеспечения. Эффективные реализации OpenGL существуют для Windows, Unix- платформ, PlayStation 3 и Mac OS. Эти реализации обычно предоставляются изготовителями видеоадаптеров и активно используют возможности последних.

3 ВИХІДНИЙ КОД

// May Ang

// mang@ucsc.edu

// CS 161 - Animation & Visualization

// Final Project - Rain/Hail/Snow Simulation

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <GL/glut.h>

#include <GL/gl.h>

#define MAX\_PARTICLES 1000

#define WCX 640

#define WCY 480

#define RAIN 0

#define SNOW 1

#define HAIL 2

float slowdown = 2.0; //static slow down

float velocity = 0.0; //how fast it is

float zoom = -40.0; // increase, decrease

float pan = 0.0;

float tilt = 0.0; //rotate

float hailsize = 0.1; // size of hailgrad

int loop;

int fall;

//floor colors

float r = 0.0;

float g = 1.0;

float b = 0.0;

float ground\_points[21][21][3];

float ground\_colors[21][21][4];

float accum = -10.0;

typedef struct {

// Life

bool alive; // is the particle alive?

float life; // particle lifespan

float fade; // decay

// color

float red;

float green;

float blue;

// Position/direction

float xpos;

float ypos;

float zpos;

// Velocity/Direction, only goes down in y dir

float vel;

// Gravity

float gravity;

}particles;

// Paticle System

particles par\_sys[MAX\_PARTICLES];

void normal\_keys(unsigned char key, int x, int y) {

if (key == 'r') { // Rain

fall = RAIN;

glutPostRedisplay();

}

if (key == 'h') { // Hail

fall = HAIL;

glutPostRedisplay();

}

if (key == 's') { // Snow

fall = SNOW;

glutPostRedisplay();

}

if (key == '=') { //really '+' - make hail bigger

hailsize += 0.01;

}

if (key == '-') { // make hail smaller

if (hailsize > 0.1) hailsize -= 0.01;

}

if (key == ',') { // really '<' slow down

if (slowdown > 4.0) slowdown += 0.01;

}

if (key == '.') { // really '>' speed up

if (slowdown > 1.0) slowdown -= 0.01;

}

if (key == 'q') { // QUIT

exit(0);

}

}

void special\_keys(int key, int x, int y) {

if (key == GLUT\_KEY\_UP) {

zoom += 1.0;

}

if (key == GLUT\_KEY\_DOWN) {

zoom -= 1.0;

}

if (key == GLUT\_KEY\_RIGHT) {

pan += 1.0;

}

if (key == GLUT\_KEY\_LEFT) {

pan -= 1.0;

}

if (key == GLUT\_KEY\_PAGE\_UP) {

tilt -= 1.0;

}

if (key == GLUT\_KEY\_PAGE\_DOWN) {

tilt += 1.0;

}

}

void initParticles(int i) {

par\_sys[i].alive = true;

par\_sys[i].life = 2.0;

par\_sys[i].fade = float(rand() % 100) / 1000.0f + 0.003f;

par\_sys[i].xpos = (float)(rand() % 21) - 10;

par\_sys[i].ypos = 10.0;

par\_sys[i].zpos = (float)(rand() % 21) - 10;

par\_sys[i].red = 0.5;

par\_sys[i].green = 0.5;

par\_sys[i].blue = 1.0;

par\_sys[i].vel = velocity;

par\_sys[i].gravity = -0.8;//-0.8;

}

void init() {

int x, z;

glShadeModel(GL\_SMOOTH);

glClearColor(0.0, 0.0, 0.0, 0.0);

glClearDepth(1.0);

glEnable(GL\_DEPTH\_TEST);

// Ground Verticies

// Ground Colors

for (z = 0; z < 21; z++) {

for (x = 0; x < 21; x++) {

ground\_points[x][z][0] = x - 10.0;

ground\_points[x][z][1] = accum;

ground\_points[x][z][2] = z - 10.0;

ground\_colors[z][x][0] = r; // red value

ground\_colors[z][x][1] = g; // green value

ground\_colors[z][x][2] = b; // blue value

ground\_colors[z][x][3] = 0.0; // acummulation factor

}

}

// Initialize particles

for (loop = 0; loop < MAX\_PARTICLES; loop++) {

initParticles(loop);

}

}

// For Rain

void drawRain() {

float x, y, z;

for (loop = 0; loop < MAX\_PARTICLES; loop = loop + 2) {

if (par\_sys[loop].alive == true) {

x = par\_sys[loop].xpos;

y = par\_sys[loop].ypos;

z = par\_sys[loop].zpos + zoom;

// Draw particles

glColor3f(0.5, 0.5, 1.0);

glBegin(GL\_LINES);

glVertex3f(x, y, z);

glVertex3f(x, y + 0.5, z);

glEnd();

// Update values

//Move

// Adjust slowdown for speed!

par\_sys[loop].ypos += par\_sys[loop].vel / (slowdown \* 1000);

par\_sys[loop].vel += par\_sys[loop].gravity;

// Decay

par\_sys[loop].life -= par\_sys[loop].fade;

if (par\_sys[loop].ypos <= -10) {

par\_sys[loop].life = -1.0;

}

//Revive

if (par\_sys[loop].life < 0.0) {

initParticles(loop);

}

}

}

}

// For Hail

void drawHail() {

float x, y, z;

for (loop = 0; loop < MAX\_PARTICLES; loop = loop + 2) {

if (par\_sys[loop].alive == true) {

x = par\_sys[loop].xpos;

y = par\_sys[loop].ypos;

z = par\_sys[loop].zpos + zoom;

// Draw particles

glColor3f(0.8, 0.8, 0.9);

glBegin(GL\_QUADS);

// Front

glVertex3f(x - hailsize, y - hailsize, z + hailsize); // lower left

glVertex3f(x - hailsize, y + hailsize, z + hailsize); // upper left

glVertex3f(x + hailsize, y + hailsize, z + hailsize); // upper right

glVertex3f(x + hailsize, y - hailsize, z + hailsize); // lower left

//Left

glVertex3f(x - hailsize, y - hailsize, z + hailsize);

glVertex3f(x - hailsize, y - hailsize, z - hailsize);

glVertex3f(x - hailsize, y + hailsize, z - hailsize);

glVertex3f(x - hailsize, y + hailsize, z + hailsize);

// Back

glVertex3f(x - hailsize, y - hailsize, z - hailsize);

glVertex3f(x - hailsize, y + hailsize, z - hailsize);

glVertex3f(x + hailsize, y + hailsize, z - hailsize);

glVertex3f(x + hailsize, y - hailsize, z - hailsize);

//Right

glVertex3f(x + hailsize, y + hailsize, z + hailsize);

glVertex3f(x + hailsize, y + hailsize, z - hailsize);

glVertex3f(x + hailsize, y - hailsize, z - hailsize);

glVertex3f(x + hailsize, y - hailsize, z + hailsize);

//Top

glVertex3f(x - hailsize, y + hailsize, z + hailsize);

glVertex3f(x - hailsize, y + hailsize, z - hailsize);

glVertex3f(x + hailsize, y + hailsize, z - hailsize);

glVertex3f(x + hailsize, y + hailsize, z + hailsize);

//Bottom

glVertex3f(x - hailsize, y - hailsize, z + hailsize);

glVertex3f(x - hailsize, y - hailsize, z - hailsize);

glVertex3f(x + hailsize, y - hailsize, z - hailsize);

glVertex3f(x + hailsize, y - hailsize, z + hailsize);

glEnd();

// Update values

//Move

if (par\_sys[loop].ypos <= -10) {

par\_sys[loop].vel = par\_sys[loop].vel \* -1.0;

}

par\_sys[loop].ypos += par\_sys[loop].vel / (slowdown \* 1000); // \* 1000

par\_sys[loop].vel += par\_sys[loop].gravity;

// Decay

par\_sys[loop].life -= par\_sys[loop].fade;

//Revive

if (par\_sys[loop].life < 0.0) {

initParticles(loop);

}

}

}

}

// For Snow

void drawSnow() {

float x, y, z;

for (loop = 0; loop < MAX\_PARTICLES; loop = loop + 2) {

if (par\_sys[loop].alive == true) {

x = par\_sys[loop].xpos;

y = par\_sys[loop].ypos;

z = par\_sys[loop].zpos + zoom;

// Draw particles

glColor3f(1.0, 1.0, 1.0);

glPushMatrix();

glTranslatef(x, y, z);

glutSolidSphere(0.2, 16, 16);

glPopMatrix();

// Update values

//Move

par\_sys[loop].ypos += par\_sys[loop].vel / (slowdown \* 1000);

par\_sys[loop].vel += par\_sys[loop].gravity;

// Decay

par\_sys[loop].life -= par\_sys[loop].fade;

if (par\_sys[loop].ypos <= -10) {

int zi = z - zoom + 10;

int xi = x + 10;

ground\_colors[zi][xi][0] = 1.0;

ground\_colors[zi][xi][2] = 1.0;

ground\_colors[zi][xi][3] += 1.0;

if (ground\_colors[zi][xi][3] > 1.0) {

ground\_points[xi][zi][1] += 0.1;

}

par\_sys[loop].life = -1.0;

}

//Revive

if (par\_sys[loop].life < 0.0) {

initParticles(loop);

}

}

}

}

// Draw Particles

void drawScene() {

int i, j;

float x, y, z;

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

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glRotatef(pan, 0.0, 1.0, 0.0);

glRotatef(tilt, 1.0, 0.0, 0.0);

// GROUND?!

glColor3f(r, g, b);

glBegin(GL\_QUADS);

// along z - y const

for (i = -10; i + 1 < 11; i++) {

// along x - y const

for (j = -10; j + 1 < 11; j++) {

glColor3fv(ground\_colors[i + 10][j + 10]);

glVertex3f(ground\_points[j + 10][i + 10][0],

ground\_points[j + 10][i + 10][1],

ground\_points[j + 10][i + 10][2] + zoom);

glColor3fv(ground\_colors[i + 10][j + 1 + 10]);

glVertex3f(ground\_points[j + 1 + 10][i + 10][0],

ground\_points[j + 1 + 10][i + 10][1],

ground\_points[j + 1 + 10][i + 10][2] + zoom);

glColor3fv(ground\_colors[i + 1 + 10][j + 1 + 10]);

glVertex3f(ground\_points[j + 1 + 10][i + 1 + 10][0],

ground\_points[j + 1 + 10][i + 1 + 10][1],

ground\_points[j + 1 + 10][i + 1 + 10][2] + zoom);

glColor3fv(ground\_colors[i + 1 + 10][j + 10]);

glVertex3f(ground\_points[j + 10][i + 1 + 10][0],

ground\_points[j + 10][i + 1 + 10][1],

ground\_points[j + 10][i + 1 + 10][2] + zoom);

}

}

glEnd();

// Which Particles

if (fall == RAIN) {

drawRain();

}

else if (fall == HAIL) {

drawHail();

}

else if (fall == SNOW) {

drawSnow();

}

glutSwapBuffers();

}

void reshape(int w, int h) {

if (h == 0) h = 1;

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45, (float)w / (float)h, .1, 200);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void idle() {

glutPostRedisplay();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_RGB | GLUT\_DOUBLE);

glutInitWindowSize(WCX, WCY);

glutCreateWindow("Edward - OpenGL");

init();

glutDisplayFunc(drawScene);

glutReshapeFunc(reshape);

glutKeyboardFunc(normal\_keys);

glutSpecialFunc(special\_keys);

glutIdleFunc(idle);

glutMainLoop();

return 0;

}

4 ВІДЕОКОПІЯ РЕЗУЛЬТАТУ

На рисунку 4.1 показано результат виконання програми.

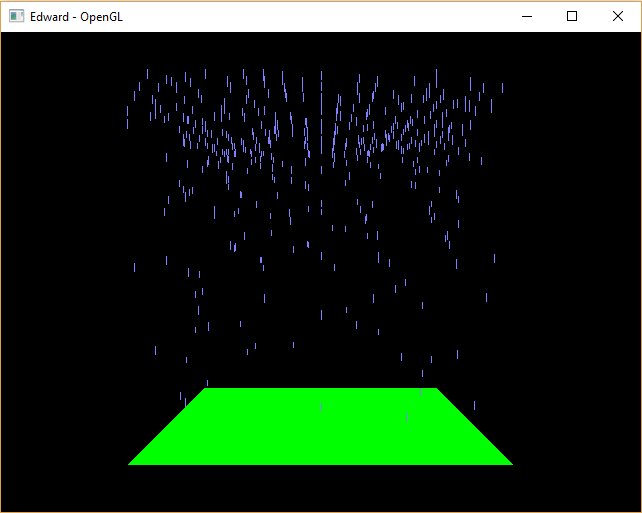


Рисунок 4.1 – Відеокопія результату виконання програми



Рисунок 4.2 – Відеокопія результату виконання програми

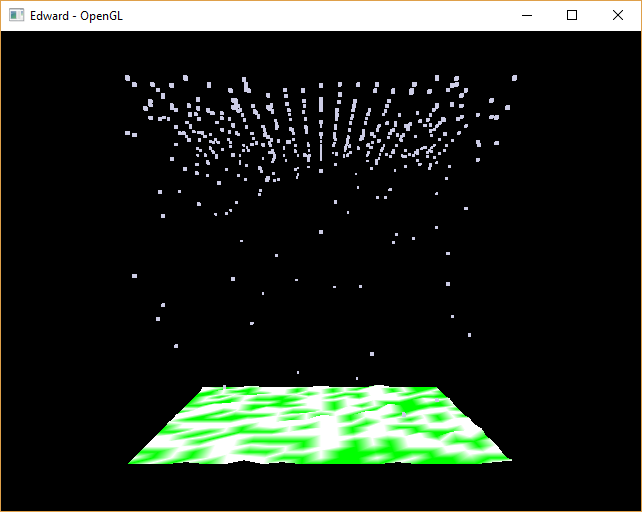


Рисунок 4.3 – Відеокопія результату виконання програми

ВИСНОВОК

У рамках цієї лабораторної роботи я засвоїв основи OpenGL. А саме створив симуляцію доща, граду та снігу використовуючи відомі нам бібліотеки GL, GLFW, FreeGlut. Створив керуючі сигнали callback. Наприклад, зміна стану (дощ, град, сніг).