**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

**ОТЧёТ**

**по лабораторной работе №3**

*дисциплина: Компьютерная графика*

Студент: Рыбалко Элина Павловна

Группа: НПИбд-02-19

**МОСКВА**

2021 г.

**Цель работы:**

Задание: написать компьютерную программу для построения изображения трехмерного тела в виде каркасной модели. Программа должна соответствовать следующим требованиям:

- Загружать данные из файла описания объекта типа .dat, состоящего из двух разделов – описание координат вершин и описание ребер. Файл может содержать данные о числе вершин и числе ребер;

- Должен быть реализован объектно-ориентированный подход, моделирующий основные элементы объекта (пример в файле objects.rtf);

- Должна быть возможность изменять положение точки наблюдения;

- Должны быть построены изображения куба (тестовое изображение) и объекта, спроектированного автором программы.

Используемая графическая библиотека SFML в MS Visual Sudio 2022.

**Листинг:**

#include <SFML/Graphics.hpp>

#include <Windows.h>

#include <iostream>

#include <fstream>

#include <math.h>

#define wx 1000

#define wy 1000

using namespace sf;

using namespace std;

void draw\_line(int x1, int y1, int x2, int y2, RenderWindow& window) { // Заготовка линии

Vertex line[] = {

Vertex(Vector2f(x1, y1)),

Vertex(Vector2f(x2, y2))

};

window.draw(line, 2, Lines);

}

void put\_pixel(int x, int y, Color color, RenderWindow& window) { //функция отрисовки точки

RectangleShape rectangle1(Vector2f(3.f, 3.f)); //прямоугольников размером 3х3

rectangle1.move(x, y); //смещаем в позицию с координатами x, y

rectangle1.setFillColor(color); //задаём цвет

window.draw(rectangle1); //отрисовка

}

void myline(int x1, int y1, int x2, int y2, Color co, RenderWindow& window) //функция отрисовки линии

{

const int N = abs(x2 - x1); //расстояние по X

const int M = abs(y2 - y1); //расстояние по Y

//определяем направление линии для смещения текущей позиции точки линии

const int signX = x1 < x2 ? 1 : -1; //если слева-направо, то 1, иначе -1

const int signY = y1 < y2 ? 1 : -1; //если сверху-вниз, то 1, иначе -1

int buf = N - M; //разница расстояний по X и Y

put\_pixel(x2, y2, co, window); //вызов функции отрисовки точки конца линии

while (x1 != x2 || y1 != y2) //пока не дошли от начала до конца линии

{

put\_pixel(x1, y1, co, window); //вызов функции отрисовки точек линии

int buf\_ = buf \* 2; //промежуточное значение удвоенной разницы расстояний X и Y

if (buf\_ > -M) //если больше расстояния по Y с отрицательным значением

{

buf -= M; //вычитаем расстояние по Y из разницы расстояний по X и Y

x1 += signX; //смещаем текущую позицию линии по X

}

if (buf\_ < N) //если меньше расстояния по X

{

buf += N; //прибавляем расстояние по X к разнице расстояний по X и Y

y1 += signY; //смещаем текущую позицию линии по Y

}

}

}

// Координаты

class point

{

public:

double x; double y; double z;

};

// Вершины

class vertex

{

public:

point worldcoord; // Мировые координаты

point viewcoord; // Видовые координаты

// Экранные координаты

double X; double Y;

vertex() {}

vertex(double x, double y, double z) // Задаём мировые координаты

{

worldcoord.x = x; worldcoord.y = y; worldcoord.z = z;

}

void setviewcoord(point viewpoint, double vert, double hor, double d) // Преобразуем мировые координаты в видовые

{

X = hor;

Y = vert;

//Преобразование координат точки наблюдения в радианы - получение углов ро, тета и фи

double temp = atan(1.0) / 45.0;

double ro = viewpoint.x \* temp;

double teta = viewpoint.y \* temp;

double fi = viewpoint.z \* temp;

// Матрицы

double wmatrix[4] = { worldcoord.x, worldcoord.y, worldcoord.z, 1 }; // Матрица мировых координат

double vmatrix[4][4] = {

{-sin(teta), -cos(fi)\*cos(teta), -sin(fi)\*cos(teta), 0},

{cos(teta), -cos(fi)\*sin(teta), -sin(fi)\*sin(teta), 0},

{0, sin(fi), -cos(fi), 0},

{0, 0, ro, 1}

}; // Преобразование мировых координат

// Матрица видового преобразования

double ematrix[4];

// Вычисление

for (int i = 0; i < 4; i++)

{

ematrix[i] = 0;

for (int j = 0; j < 4; j++)

{

ematrix[i] += wmatrix[j] \* vmatrix[j][i];

}

}

// Задаём видовые координаты из полученных перемножений матриц

viewcoord.x = ematrix[0];

viewcoord.y = ematrix[1];

viewcoord.z = ematrix[2];

//Вычисление экранных координат

X += d\*viewcoord.x/viewcoord.z;

Y += d\*viewcoord.y/viewcoord.z;

}

};

// Рёбра

class edge

{

public:

// Начало и конец ребра

vertex start;

vertex end;

edge() {}

edge(vertex svertex, vertex evertex)

{

start = svertex; end = evertex;

}

//Отрисовка ребра

void drawedge(Color color, RenderWindow& window)

{

myline(start.X, start.Y, end.X, end.Y, color, window);

}

};

// Объект

class surface

{

private:

vector<vertex> vlist; // Список вершин

vector<vector<int>> elist; // Список рёбер

public:

point viewpoint;

void load(const char\* filename)

{

ifstream f(filename);

if (!f.is\_open()) // если файл не был открыт

{

cout << "Файл не может быть открыт или создан\n"; // напечатать соответствующее сообщение

exit(1); // выполнить выход из программы

}

// Считываем координаты вершин

int vcount; // Количество вершин

int coord[3]; // 3 координаты x, y, z

f >> vcount;

for (int i = 0; i < vcount; i++)

{

for (int j = 0; j < 3; j++)

{

f >> coord[j];

}

vlist.push\_back(vertex(coord[0], coord[1], coord[2])); // Добавляем в список вершин

}

//Загрузка информации о ребрах и вершинах, между которыми они находятся

//string ecount; // Количество рёбер

int ecount; // Количество рёбер

int vert[2]; // 2 вершины

f >> ecount;

for (int i = 0; i < ecount; i++)

{

for (int j = 0; j < 2; j++)

{

f >> vert[j];

}

// Добавляем в список рёбер

elist.push\_back({});

elist[i].push\_back(vert[0]);

elist[i].push\_back(vert[1]);

}

f.close(); // Закрываем файл

}

//Установка точки наблюдения

void setviewpoint(double ro, double theta, double fi)

{

viewpoint.x = ro; viewpoint.y = theta; viewpoint.z = fi;

}

//Отрисовка каркасной модели

void drawsurface(double vert, double hor, double d, Color color, RenderWindow& window)

{

// Преобразование координат вершин из списка вершин

for (int i = 0; i < vlist.size(); i++)

{

vlist[i].setviewcoord(viewpoint, vert, hor, d);

}

// Отрисовка рёбер

for (int i = 0; i < elist.size(); i++)

{

edge edge1 = edge(vlist[elist[i][0] - 1], vlist[elist[i][1] - 1]);

edge1.drawedge(color, window);

}

}

};

int main()

{

RenderWindow window(VideoMode(wx, wy), "SFML Works!!!"); //Главное окно приложения

while (window.isOpen()) // Главный цикл приложения: выполняется, пока открыто окно

{

Event event; // Обработка событий в цикле

while (window.pollEvent(event))

{

if (event.type == Event::Closed) // Собитие закрытия окна

window.close();

}

window.clear(Color(5, 0, 50, 0)); // Цвет фона

// Объявление переменных

Color color = Color(105, 0, 200); // Цвет линий

double P1 = 0.3; // Угол тета

double P2 = 80.0; // Угол фи

const char\* inputfile = "C:\\Users\\rybalko\_pu\\Documents\\Study\\3 курс\\2 модуль\\Компьютерная графика\\Лаб 3\\Cube.dat"; // Путь до файла координат

//const char\* inputfile = "C:\\Users\\rybalko\_pu\\Documents\\Study\\3 курс\\2 модуль\\Компьютерная графика\\Лаб 3\\House.dat"; surface s1;

s1.load(inputfile);

for (int i = 1; i < 10; i++) { // Движение точки наблюдения

s1.setviewpoint(1000, P1, P2); // Точка наблюдения

s1.drawsurface(wx/2, wy/2, 3000, color, window); // Отрисовка модели

// Смещение точки наблюдения

P1 += 0.5;

P2 += 5;

Sleep(500); // Задержка по времени

window.display(); // Отрисовка окна

window.clear(Color(5, 0, 50, 0)); // Цвет фона

}

}

return 0;

}

**Координаты:**

1. **Тестовый объект – куб**

8

1 1 -1

-1 1 -1

-1 -1 -1

1 -1 -1

1 1 1

-1 1 1

-1 -1 1

1 -1 1

12

1 2

2 3

3 4

4 1

5 6

6 7

7 8

8 5

1 5

2 6

3 7

4 8

1. **Авторский объект – домик**

9

1 1 -1

-1 1 -1

-1 -1 -1

1 -1 -1

1 1 1

-1 1 1

-1 -1 1

1 -1 1

0 0 -2

16

1 2

2 3

3 4

4 1

5 6

6 7

7 8

8 5

1 5

2 6

3 7

4 8

1 9

2 9

3 9

4 9

**Результат:**

