РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ
Факультет физико-математических и естественных наук
Кафедра информационных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

дисциплина: компьютерная графика

Студент: Абу Сувейлим Мухаммед Мунифович
Группа: НКНбд-01-21

МОСКВА

2023 г.

Содержание

I. Цель работы:

Задание: написать компьютерную программу для построения трехмерного куба.

II. Ход работы:

Для выполнения данной лабораторной работы необходимо написать программу, которая будет читать из файла координаты в мировой системе координат (либо напимую в main() будут координаты), будет читать номера вершин координат, между которыми нужно будет провести грани, будет рисовать каркасную модель на экране с помощью полученных данных.

Для построения каркасной модели трехмерного объекта, заданного мировыми координатами, необходимо мировые координаты (x_w, y_w, z_w)преобразовать в видовые координаты (x_e,y_e,z_e), сделать перспективное преобразование, затем, координаты, которые мы получили, преобразовать в экранные координаты (i,j).

III. Анализ результатов

Ниже результаты выполнения программы.

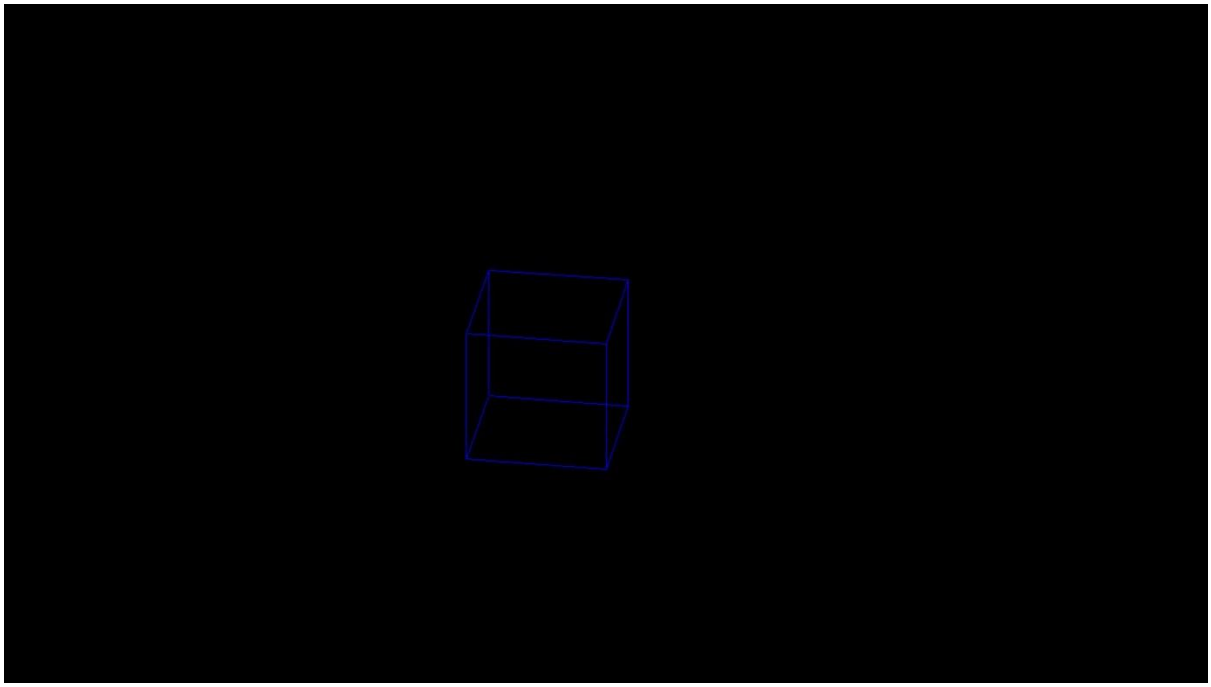

Фото 1: каркас куба

IV. Польный код

Ниже полностью указан код мой программы, написанной на языке программирования c++.

```
#include <iostream>
#include <vector>
#include "graphics.h"
#include <cmath>

#define PI 3.14159265358979323846

class Point {
public:
    double x, y, z;

    Point(double xc, double yc, double zc) {
        setPoint(xc, yc, zc);
    }

    void setPoint(double xc, double yc, double zc) {
        x = xc;
        y = yc;
        z = zc;
    }

    Point() {}
};

class Vertex {
public:
    Point worldCoord; // World global coordinates
    Point viewCoord;  // Coordinates from our viewpoint in the world
coordinates system

    static Vertex* first;

    Vertex(Point& p) {
        this->next = first;
        first = this;
        worldCoord = p;
    }

    void setViewCoord(Point viewpoint)
     {
        double ro = viewpoint.x;
        double theta = viewpoint.y;
        double fi = viewpoint.z;

        // Transformation matrix components
```

```
        double ST = sin(theta);
        double CT = cos(theta);
        double SF = sin(fi);
        double CF = cos(fi);

        // Transformation matrix
        double V[4][4] = {
            {-ST, CT, 0, 0},
            {-CF * CT, -CF * ST, SF, 0},
            {-SF * CT, -SF * ST, -CF, ro},
            {0, 0, 0, 1}
        };

        // Homogeneous coordinates for the world point
        double worldCoords[4] = {worldCoord.x, worldCoord.y,
worldCoord.z, 1};

        // Initialize viewCoords to zero
        double viewCoords[4] = {0};

        // Perform matrix-vector multiplication
        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                viewCoords[i] += V[i][j] * worldCoords[j];
            }
        }

        // Set the transformed coordinates to viewCoord
        viewCoord.x = viewCoords[0];
        viewCoord.y = viewCoords[1];
        viewCoord.z = viewCoords[2];
    }

    Vertex* next;
};

Vertex* Vertex::first = NULL;

class Edge {
public:
    Vertex* startVertex;
    Vertex* finishVertex;

    static Edge* first;

    Edge(Vertex* start, Vertex* finish) {
        this->next = first;
        first = this;
        startVertex = start;
        finishVertex = finish;
    }
```

```
    void drawEdge() {
        double X1 = (startVertex->viewCoord.z * startVertex-
>viewCoord.x) / (2 * startVertex->viewCoord.z);
        double Y1 = (startVertex->viewCoord.z * startVertex-
>viewCoord.y) / (2 * startVertex->viewCoord.z);
        double X2 = (finishVertex->viewCoord.z * finishVertex-
>viewCoord.x) / (2 * finishVertex->viewCoord.z);
        double Y2 = (finishVertex->viewCoord.z * finishVertex-
>viewCoord.y) / (2 * finishVertex->viewCoord.z);

        int i1 = getmaxx() / 2 + X1;
        int j1 = getmaxy() / 2 - Y1;
        int i2 = getmaxx() / 2 + X2;
        int j2 = getmaxy() / 2 - Y2;

        line(i1, j1, i2, j2);
    }

    Edge* next;
};

Edge* Edge::first = NULL;

class Surface {
public:
    static Point viewpoint;

    void setViewpoint(double ro, double theta, double fi) {
        viewpoint.setPoint(ro, theta, fi);
    }

    void drawSurface() {
        Edge* tmp = Edge::first;
        while (tmp != NULL)
           {
            tmp->drawEdge();
            tmp = tmp->next;
        }
    }
};

Point Surface::viewpoint; // Initialize static member

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    Point p1(0, 0, 0);
    Point p2(300, 0, 0);
    Point p3(0, 300, 0);
```

```
    Point p4(0, 0,300);
    Point p5(300, 300, 0);
    Point p6(0, 300, 300);
    Point p7(300, 0, 300);
    Point p8(300, 300, 300);

    Surface C;
    //C.setViewpoint(10000, PI, PI*4);

    Vertex* v1 = new Vertex(p1);
    Vertex* v2 = new Vertex(p2);
    Vertex* v3 = new Vertex(p3);
    Vertex* v4 = new Vertex(p4);
    Vertex* v5 = new Vertex(p5);
    Vertex* v6 = new Vertex(p6);
    Vertex* v7 = new Vertex(p7);
    Vertex* v8 = new Vertex(p8);

    Vertex* tmp = Vertex::first;
    while (tmp != NULL) {
        tmp->setViewCoord(Surface::viewpoint);
        tmp = tmp->next;
    }

    setbkcolor(0);
    setcolor(9);

    Edge* e1 = new Edge(v1, v2);
    Edge* e2 = new Edge(v1, v3);
    Edge* e3 = new Edge(v1, v4);
    Edge* e4 = new Edge(v5, v2);
    Edge* e5 = new Edge(v7, v2);
    Edge* e6 = new Edge(v3, v5);
    Edge* e7 = new Edge(v3, v6);
    Edge* e8 = new Edge(v4, v6);
    Edge* e9 = new Edge(v4, v7);
    Edge* e10 = new Edge(v5, v8);
    Edge* e11 = new Edge(v6, v8);
    Edge* e12 = new Edge(v7, v8);

    double angle = 0;
     while (true) {
        cleardevice();
        C.setViewpoint(10, angle, angle*3);
        angle += PI / 100;
        C.drawSurface();
        tmp = Vertex::first;
        while (tmp != NULL)
           {
            tmp->setViewCoord(Surface::viewpoint);
            tmp = tmp->next;
```

```
        }
        delay(10);
    }

    getch();
    closegraph();
    return 0;
}
```

V. Список литературы

1. Л. Аммерал, принципы программирования в машиной графике.