**Лабораторная работа н.3**

**Цель работы:** Реализовать программу для рисование трехмерной проволочной модели в реальном времени с загрузкой данных из файла срествами базовых методов бибилотеки graphics.h

**Ход Работы:**

1. Подключим нужные библиотеки(Рис. 1):
2. #include <iostream>
3. #include "graphics.h"
4. #include <fstream>
5. #include <math.h>
6. #include <time.h>
7. using namespace std;
8. const double PI = 3.1415;

рис. 1. Небоходимые библиотеки

**2.** Первоначально опишем класс матрицы поворота и реализуем алгоритм для подсчета коэффицентов в ней(рис. 2):

class V\_matrix{

public:

    double ro, phi, theta;

    double sinth, costh, sinph, cosph;

    V\_matrix(){

        this->ro = 0;

        this->theta = 0;

        this->phi = 0;

    }

    void calculate(double ro, double phi, double theta){

        this->ro = ro;

        this->phi = phi;

        this->theta = theta;

        this->sinth = sin(this->theta\*PI/180);

        this->costh = cos(this->theta\*PI/180);

        this->sinph = sin(this->phi\*PI/180);

        this->cosph = cos(this->phi\*PI/180);

    }

};

Рис. 2. Класс матрицы поворота.

Сама матрица выглядит вот так(рис. 3):

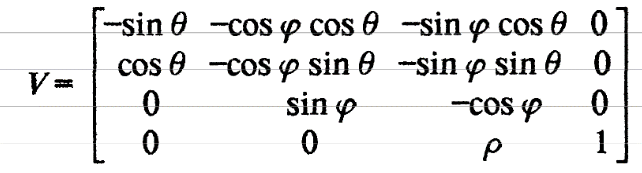


Рис. 3. Реализуемая матрица.

Углы ro, phi и theta – углы в сферических координатах(рис. 4):

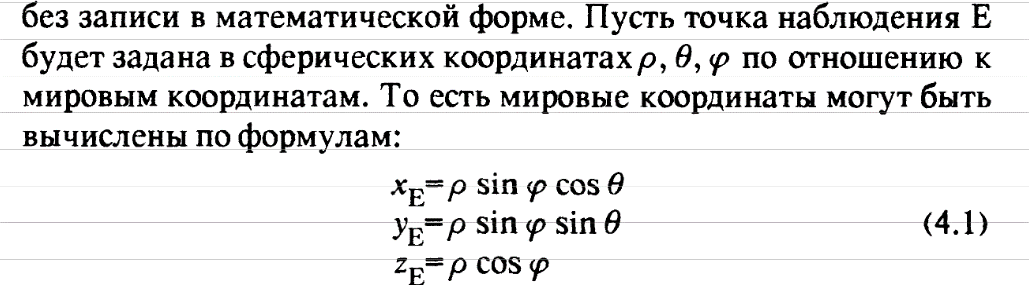


Рис. 4. Сферические координаты



Рис. 5. Пребразование мировых координат точки к видовым

**3.** Теперь опишем классы точки, вершины и ребра(Рис. 6):

struct vector3d{

    double x, y, z; // int

    vector3d(){

        this->x = 0;

        this->y = 0;

        this->z = 0;

    }

};

struct Vertex{

    vector3d worldcoord, viewcoord;

    int num;

    Vertex\* next;

    Vertex(double x, double y, int z, int num){ // int

        this->worldcoord.x = x;

        this->worldcoord.y = y;

        this->worldcoord.z = z;

        this->num = num;

    }

    void setviewcoord(V\_matrix v){

        this->viewcoord.x = -this->worldcoord.x \* v.sinth + this->worldcoord.y \* v.costh;

        this->viewcoord.y = -this->worldcoord.x \* v.cosph \* v.costh - this->worldcoord.y \* v.cosph \* v.sinth  + this->worldcoord.z \* v.sinph;

        this->viewcoord.z = -this->worldcoord.x \* v.sinph \* v.costh - this->worldcoord.y \* v.sinph \* v.sinth  - this->worldcoord.z \* v.cosph + v.ro;

    }

};

class Edge{

public:

    Vertex\* startvertex;

    Vertex\* finishvertex;

    Edge\* next;

    Edge(){}

    void drawedge(V\_matrix v, int d, int c1, int c2){

        srand(time(0));

        startvertex->setviewcoord(v);

        finishvertex->setviewcoord(v);

        setcolor(GREEN);

        line(int(d \* startvertex->viewcoord.x / startvertex->viewcoord.z + c1/2), int(d \* startvertex->viewcoord.y / startvertex->viewcoord.z + c2/2),

        int(d \* finishvertex->viewcoord.x / finishvertex->viewcoord.z + c1/2), int(d \* finishvertex->viewcoord.y / finishvertex->viewcoord.z + c2/2));

    }

};

Рис. 6. Классы точки, вершины и ребра

Класс точки содержит всего три атрибута – координаты по трем измерениям.

Класс вершины представляет из себя, для начала, два вида класса точки – мировые и видовые координаты(worldcoord и viewcoord), а также номер вершины(num). Для того, чтобы хранить все вершины фигуры, мы будем использовать односвязный список, чтобы при построении проходить вершины по очереди. Класс вершины также содержит конструктор, в который мы передаем заданные мировые координаты и инкрементриует значение переменной num, чтобы создать правильную нумерацию вершин. Последний метод – преобразование координат вершины из мировых в видовые. Этот метод принимает экземпляр класса матрица поворота и преобразует мировые координаты в видовые.

Класс ребра представляет из себя также односвязный список(Edge\* next) и хранит ссылки на начальную и конечную вершины(startvertex и finishvertex). Метод у данного класса один – рисование ребра(drawedge), который принимает матрицу(переменная v) и три других атрибута(рис. 7): расстояние от точки наблюдения до экрана(d), и значения разрешения экрана(с1 по ширине и c2 по высоте). Если разрешение экрана 1200 на 600, то c1 и с2 должны принимать значения 600 и 300 соответственно.

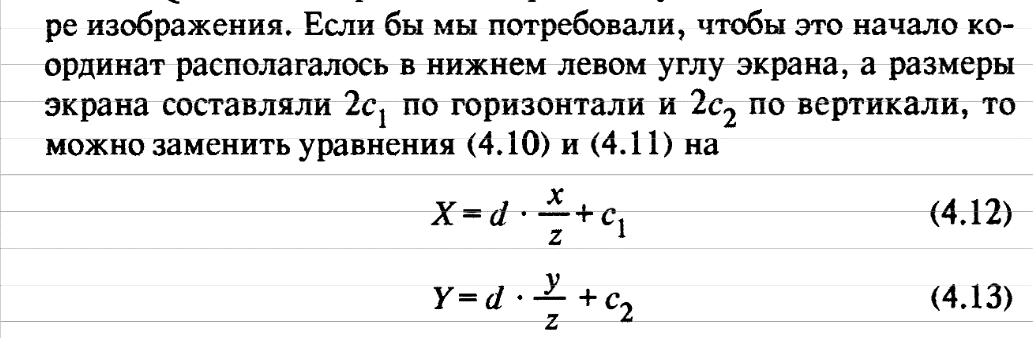


Рис. 7. Преобразование трехмерных координат к экранным

4. Последний класс, который нам пригодится – класс поверхности.

class Surface{

    Vertex\* vlist;

    Edge\* elist;

    V\_matrix\* v;

    int num;

public:

    Surface(){}

    void load(){

        vlist = nullptr;elist = nullptr; Vertex\* vtmp; Edge\* etmp;

        int i, n, m;

        double xx, yy, zz, n1, n2;

        //Обращение к файлу с вершинами и ребрами

        ifstream InputFile("pira.txt");

        //Считывание координат вершин

        InputFile >> n;

        for(i = 0; i < n; i++){

            InputFile>>xx>>yy>>zz;

            vtmp = new Vertex(xx, yy, zz, i+1);

            if(vlist == nullptr)

                vlist = vtmp;

            else{

                vtmp->next = vlist;

                vlist = vtmp;

            }

        }

        InputFile>>m;

        this->num = m;

        for(i = 0; i < m; i++){

            InputFile >> n1 >> n2;

            etmp = new Edge();

            etmp->next == nullptr;

            vtmp = vlist;

            etmp->startvertex=nullptr;

            while(etmp->startvertex == nullptr && vtmp != nullptr){

                if(vtmp->num == n1)

                    etmp->startvertex = vtmp;

                else

                    vtmp=vtmp->next;

            }

            vtmp = vlist;

            etmp->finishvertex=nullptr;

            while(etmp->finishvertex == nullptr && vtmp != nullptr){

                if(vtmp->num==n2)

                    etmp->finishvertex=vtmp;

                else

                    vtmp = vtmp->next;

            }

            if(elist == nullptr)

                elist = etmp;

            else{

                etmp->next = elist;

                elist = etmp;

            }

        }

        InputFile.close();

    }

    void drawsurface(double ro, double phi, double theta, int d, int c1, int c2){

        this->v = new V\_matrix();

        this->v->calculate(ro, phi, theta);

        Edge\* etmp = elist;

        int i = 0;

        while(etmp != nullptr && this->num > i ){

            // cout << "Drawedge" << endl;

            etmp->drawedge(\*this->v, d, c1, c2);

            etmp = etmp->next;

            i++;

        }

        delete v;

    }

};

Рис. 8. Класс поверхности.

Класс включает в себя 4 атрибута – список вершин и ребер, числительное, чтобы нумеровать по очереди вершины и искать по ним, и ссылку на матрицу поворота.

Класс содержит два метода – загрузка данных из файла и рисование поверхности.

Метод загрузки работает следующим образом: загружаем мы данные всегда из файла с одним именем, сначала мы загружаем определенное количество вершин, отмеченное в файле, потом итерируемся и поочередно создаем новый экземпляр, на который указывает предыдущий экземпляер из предыдущей итерации.

Метод отрисовки граней работает так: мы создаем новую матрицу и просчитаем ее значения для заданных значений ro, phi и theta, потом создаем указатель на начало списка ребер и запускам цикл, который работает до тех пор, пока либо не попали в нулевое значение в памяти, либо не нарисовали больше вершин, чем у нас есть. В цикле мы используем встроенный метод класса ребра, передаем нужные параметры из предыдущих шагов и переходим на следующий элемент по списку. Файл с данными выглядит вот так(рис. 9):

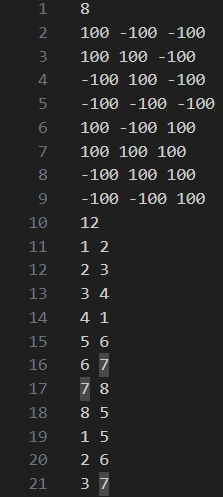


Рис. 9. Вид файла pira.txt с данными координат вершин и ребер.

5. Теперь опишем функцию main для того, чтобы поворачивать квадрат в пространстве в реальном времени(рис. 10):

int main(){

    int gddriver = DETECT, gmode, errorcode;

    initgraph(&gddriver, &gmode, "");

    double phi=30;

    double theta=50;

    double ro = 300;

    Surface\* s1 = new Surface();

    s1->load();

    int screenWidth = 400;

    int screenHeight = 300;

    int dist = 150;

    for(int i = 1; i < 100; i++){

        s1->drawsurface(ro, phi, theta, dist, screenWidth, screenHeight);

        phi = phi + 10;

        theta = theta + 30;

        delay(250);

        cleardevice();

    }

    getch();

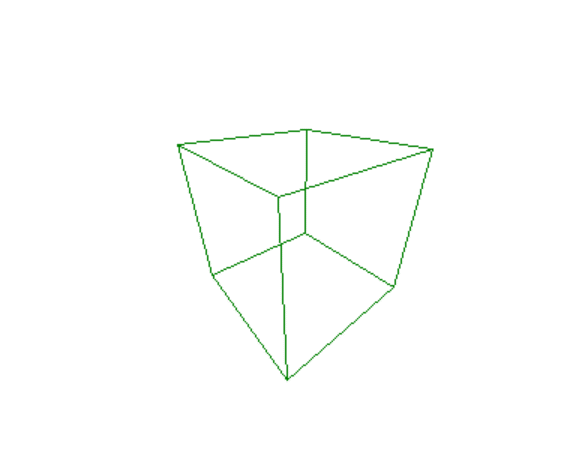
closegraph();

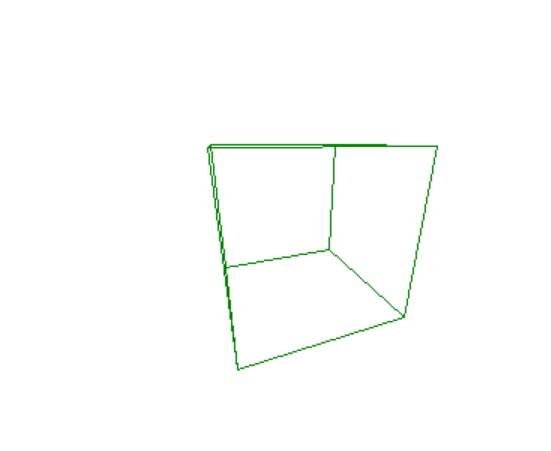
    return 0;

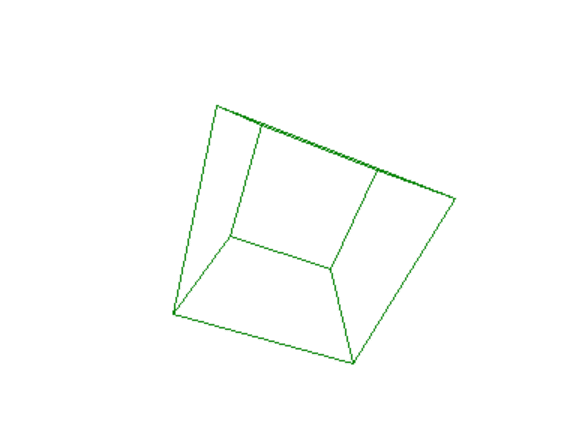
}

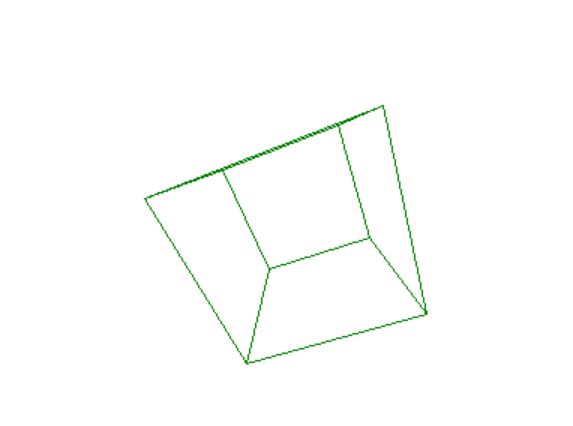
Рис. 10. Функция main

В функции main мы создаем матрицу, выбираем произвольные углы и расстояние до фигуры и экрана, выбираем произвольное разрешение экрана, создаем экземпляр класса Surface, загружаем для него данных из файла pira.txt и начинаем цикл, где будем с задержкой в 250 мс увеличивать угол phi на 10 и угол theta на 30. На рисунках ниже пример работы.









**Вывод:** с помощью данной лабораторной работы мы научились отрисовывать проволочную трехмерную фигурами средствами математического аппарата и базовых методов библиотеки graphics.h.

**Источники:**

1. Лекции по Компьютерной графике. Фомин М.Б.
2. Принципы программирования в машинной графике. Л. Аммерал.