**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**
**Факультет физико-математических и естественных наук**

# ОТЧЕТ
# ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

*дисциплина: Компьютерная графика*

Студент: Мухамедияр Адиль

Группа: НКНбд-01-20

**МОСКВА** 2022г.

# Задание

Задание: Модернизировать компьютерную программу для построения изображения выпуклого трехмерного тела в виде каркасной модели, разработанную при выполнении лабораторной работы № 3. Программа должна соответствовать следующим требованиям:

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

- Объектная модель должна быть дополнена описанием треугольных граней. Соответствующий класс должен содержать метод, выявляющий лицевые грани;

- Метод, отвечающий за построение изображения, должен быть дополнен алгоритмом удаления нелицевых граней;

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

Результаты выполнения работы должны содержать:
- Отчет о выполнении лабораторной работы;
- Текст компьютерной программы;
- Файлы описания объектов;
- Изображения – результат работы компьютерной программы.

# Ход решения

```cpp
#include <SFML/Graphics.hpp>
#include <fstream>
#include <cmath>
#include <vector>
#include <string>
#include <sstream>

sf::Image img;

void myline(int x1, int y1, int x2, int y2, sf::Image& img, sf::Color color) {

    const int deltaX = abs(x2 - x1);
    const int deltaY = abs(y2 - y1);
    const int signX = x1 < x2 ? 1 : -1;
    const int signY = y1 < y2 ? 1 : -1;
    int error = deltaX - deltaY;
    img.setPixel(x2, y2, color);
    while(x1 != x2 || y1 != y2)
    {
        img.setPixel(x1, y1, color);
        int error2 = error * 2;
        if(error2 > -deltaY)
        {
            error -= deltaY;
            x1 += signX;
        }
        if(error2 < deltaX)
        {
            error += deltaX;
            y1 += signY;
        }
    }

}

void myline1(int x1, int y1, int x2, int y2, sf::RenderWindow& window, sf::Color color) {
    sf::Vertex vertices[2];

    vertices[0].position = sf::Vector2f (x1, y1);
```

```
039         vertices[0].color = sf::Color::Yellow;
040         vertices[1].position = sf::Vector2f (x2, y2);
041         vertices[1].color = sf::Color::Yellow;
042 
043 
044         window.draw(vertices, 2, sf::Lines);
045 
046     }
047 
048     struct Point{
049         double x, y, z;
050     };
051 
052     class Vertex {
053 
054     private:
055         Point world;
056         Point view;
057 
058         friend class Edge;
059         friend class Surface;
060         friend class Triangle;
061     public:
062         Vertex(double x, double y, double z): world{x,y,z}
063         {}
064 
065         explicit Vertex() : world{0,0,0}
066         {}
067 
068 
069         void SetViewCoord(Point sphere) {
070             double st = sin(sphere.y);
071             double ct = cos(sphere.y);
072             double sf = sin(sphere.z);
073             double cf = cos(sphere.z);
074 
075             view.x = world.x*(-sf)+world.y*cf;
076             view.y = world.x*(-ct*cf) + world.y*(-ct*sf) + world.z*st;
077             view.z = world.x*(-st*cf) + world.y*(-st*sf) + world.z*(-ct) + sphere.x;
```

```
078     }
079
080     bool operator==(Vertex& v) const {
081         return (view.x == v.view.x && view.y == v.view.y && view.z == v.view.z);
082     }
083 };
084
085 class Edge {
086 private:
087     Vertex s_vert;
088     Vertex f_vert;
089
090     friend class Surface;
091     friend class Triangle;
092 public:
093     Edge(Vertex a, Vertex b) {
094         s_vert.view = {a.view.x, a.view.y, a.view.z};
095         f_vert.view = {b.view.x, b.view.y, b.view.z};
096     }
097
098     Edge(Edge& e) {
099         s_vert = e.s_vert;
100         f_vert = e.f_vert;
101     }
102
103     Edge& operator=(Edge e) {
104         s_vert = e.s_vert;
105         f_vert = e.f_vert;
106
107         return *this;
108     }
109
110     void DrawEdge(double yu, double yd, double xl, double xr, double ju, double jd, double il, double ir, double rr) {
111         double x,y,x1,y1;
112         int i, j, i1, j1;
113         x = (rr/(2.0*(s_vert.view.z)))*s_vert.view.x;
114         y = (rr/(2.0*(s_vert.view.z)))*s_vert.view.y;
115         x1 = (rr/(2.0*(f_vert.view.z)))*f_vert.view.x;
116         y1 = (rr/(2.0*(f_vert.view.z)))*f_vert.view.y;
117
118         i = (int)(ir + ((x-xr)*(il-ir))/(xl-xr));
119         j = (int)(ju + ((y-yu)*(jd-ju))/(yd-yu));
120         i1 = (int)(ir + ((x1-xr)*(il-ir))/(xl-xr));
121         j1 = (int)(ju + ((y1-yu)*(jd-ju))/(yd-yu));
122
123         myline(i,j,i1,j1,img,sf::Color::Yellow);
```

```
124     }
125
126 };
127
128 class Triangle{
129 private:
130     Vertex x_;
131     Vertex y_;
132     Vertex z_;
133     Vertex n_;
134     bool flag_;
135
136     friend class Surface;
137
138     static double Sqr(double x) {
139         return x*x;
140     }
141
142     void CalcNorm() {
143         Vertex v1, v2;
144         v1.view.x = x_.view.x - y_.view.x;
145         v1.view.y = x_.view.y - y_.view.y;
146         v1.view.z = x_.view.z - y_.view.z;
147
148         v1.view.x = y_.view.x - z_.view.x;
149         v2.view.y = y_.view.y - z_.view.y;
150         v2.view.z = y_.view.z - z_.view.z;
151
152         auto wrki = sqrt(Sqr(v1.view.y * v2.view.z - v1.view.z * v2.view.y) +
153                          Sqr(v1.view.z * v2.view.x - v1.view.x * v2.view.z) +
154                          Sqr(v1.view.x * v2.view.y - v1.view.y * v2.view.x));
155
156         n_.view.x = (v1.view.y*v2.view.z - v1.view.z*v2.view.y) / wrki;
157         n_.view.y = (v1.view.z*v2.view.x - v1.view.x*v2.view.z) / wrki;
158         n_.view.z = (v1.view.x*v2.view.y - v1.view.y*v2.view.x) / wrki;
159     }
160
161     bool Test(const Edge& e) {
162
163     }
164
165
166 public:
167     Triangle(Vertex x, Vertex y, Vertex z) : x_(x), y_(y), z_(z), flag_(false)
168     {
```

```
169             double x1, y1, x2, y2, x3, y3, k;
170             x1=x_.view.x;
171             y1=x_.view.y;
172             x2=y_.view.x;
173             y2=y_.view.y;
174             x3=z_.view.x;
175             y3=z_.view.y;
176             k=x1*y2*1+y1*1*x3+1*x2*y3-1*y2*x3-x1*1*y3-y1*x2*1;
177             if (k>0) flag_ = true;
178         }
179
180
181     };
182
183     class Surface {
184
185     private:
186         friend class Edge;
187         std::vector<Edge*> e;
188         std::vector<Vertex*> v;
189         std::vector<Triangle*> t;
190         Point viewpoint;
191         double yu, yd, xl, xr, ju, jd, il, ir;
192         int n;
193     public:
194
195         void DrawSurface() {
196             for(int i = 0; i < n; ++i) {
197                 for(int j = 0; j < t.size(); ++j) {
198                     if(t[j]->flag_) {
199                         auto f1 = e[i]->s_vert == t[j]->x_ || e[i]->s_vert == t[j]->y_ || e[i]->s_vert == t[j]->z_;
200                         auto f2 = e[i]->f_vert == t[j]->x_ || e[i]->f_vert == t[j]->y_ || e[i]->f_vert == t[j]->z_;
201
202                         if(f1 && f2)
203                             e[i]->DrawEdge(yu, yd, xl, xr, ju, jd, il, ir, viewpoint.x);
204                     }
205                 }
206             }
207
208             /*for(int i = 0; i < n; ++i) {
209                e[i]->DrawEdge(yu, yd, xl, xr, ju, jd, il, ir, viewpoint.x);
210             }
211              */
212         }
213
```

```
214     void SetViewPoint(Point& p) {
215         viewpoint = p;
216         yu = -2;
217         yd = 2;
218         xl = -2;
219         xr = 2;
220         il = 0;
221         ir = 1024;
222         ju = 0;
223         jd = 1024;
224     }
225
226     void Load() {
227         int v_num;
228         double x, y, z;
229         int a, b;
230         char ch;
231         int tx, ty, tz;
232         std::ifstream fin("D:\DevC++_project\test.txt");
233
234         fin >> n >> v_num;
235
236         v.resize(v_num);
237
238         for(int i = 0; i < v_num; ++i) {
239             fin >> x >> y >> z;
240             v[i] = new Vertex(x, y, z);
241             v[i]->SetViewCoord(viewpoint);
242         }
243
244         e.reserve(n);
245         for(int i = 0; i < n; ++i) {
246             fin >> a >> b;
247             e.push_back(new Edge(*v[a-1], *v[b-1]));
248         }
249
250         t.reserve(8);
251         for(int i = 0; i < 8; ++i) {
252             fin >> tx >> ty >> tz;
253             t.push_back(new Triangle(*v[tx-1], *v[ty-1], *v[tz-1]));
254         }
255
256     }
257 };
258
259
```

```cpp
int main() {
    img.create(1024, 1024, sf::Color::Green);
    Surface s;
    Point p = {50,M_PI/4,M_PI/4};
    s.SetViewPoint(p);
    s.Load();
    s.DrawSurface();

    img.saveToFile("test.png");

    return 0;
}
```

## Исполнение программы