МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Компьютерная графика»

Тема: Исследование алгоритмов выявления видимости сложных сцен

Ларионова Е.Е.

Студенты гр. 8362 _____ Матвеев Н.Д.

Преподаватель _____ Матвеева И. В.

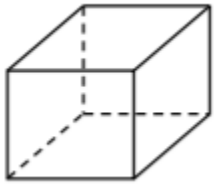
Санкт-Петербург
2021

ЗАДАНИЕ

Обеспечить реализацию алгоритма выявления видимых граней и ребер для одиночного выпуклого объемного тела.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Алгоритм удаления невидимых линий, используемый при изображении отдельных выпуклых тел.



Рассмотрим алгоритм определения видимости простого выпуклого тела: Он основывается на том, что:

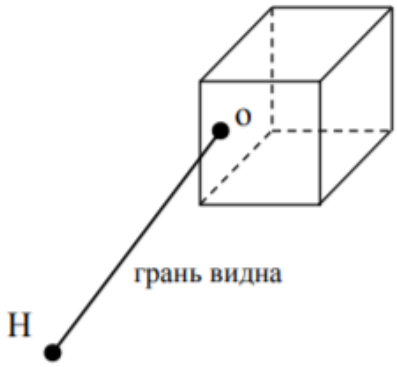
- если грань видна – то и все ребра такой грани видны:

- если грань не видна – то и все ребра будут не видны

Если найти координаты внутренней точки выпуклого тела “О” и провести через нее и точку “Н”, в которой расположен наблюдатель, прямую линию и сформировать плоскость, которая совпадает с гранью выпуклого тела, то можно довольно просто определить, видна или не видна эта грань.

Так, если плоскость пересекает линию на участке от “О” (внутренней точки) до точки “Н”, то такая грань будет видна и будут видны все ребра тела, принадлежащие этой грани.

Иначе грань не видна.



Для этого в качестве тестовой функции можно использовать матричное уравнение плоскости, определяемое тремя принадлежащими грани точками:

$$f(x,y,z)=\begin{vmatrix} x-x_1 & y-y_1 & z-z_1 \\ x_2-x_1 & y_2-y_1 & z_2-z_1 \\ x_3-x_1 & y_3-y_1 & z_3-z_1 \end{vmatrix}$$

При этом если при подстановки в это уравнение координат точки$f(x,y,z)=0$ – то точка $P(x,y,z)$ принадлежит плоскости, иначе (если > 0 или < 0) данная точка лежит в одном из полупространств, т.е. точка лежит вне плоскости.

Так как для каждой грани выпуклого тела можно определить три точки и составить тестовую функцию для соответствующей грани. В эту функцию подставляют координаты внутренней точки “О” и точки наблюдателя “Н” определяют значения тестовых функций:

$f(x_0,y_0,z_0)=f(O)$

$f(x_H,y_H,z_H)=f(H)$

Если эти функции разного знака:

$f(O)*f(H)<0$ – то плоскость пересекает прямую на участке от “О” до “Н” и анализируемая грань видна, и все ребра ее видны – их все можно отобразить;

иначе: $f(H)*f(O\geq 0$ – грань не видна. (Равенство 0 говорит о том, что плоскость проходит через точку “Н”, и плоскость рассматриваемой грани для наблюдателя вырождается в линию.) Ребра невидимых граней можно считать невидимыми и не изображать или изображать невидимыми, например, пунктирными линиями.

Координаты внутренней точки “О” выпуклого тела можно определить, сложив координаты 3-х точек, которые принадлежат одной грани тела и не лежат на одной прямой, с координатами 4-ой точки, принадлежащей любой другой грани, и разделив результат суммирования на 4.

РЕАЛИЗАЦИЯ ПРОГРАММЫ

При запуске программы открываются 2 окна «MainWindow» и «Form». В «MainWindow» задаются координаты точек. В «Form» служит для отрисовки куба (Рисунок 1 и Рисунок 2).

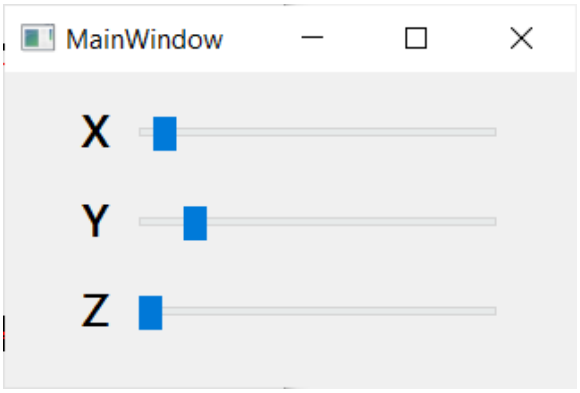


Рисунок 1 – Окно «MainWindow»

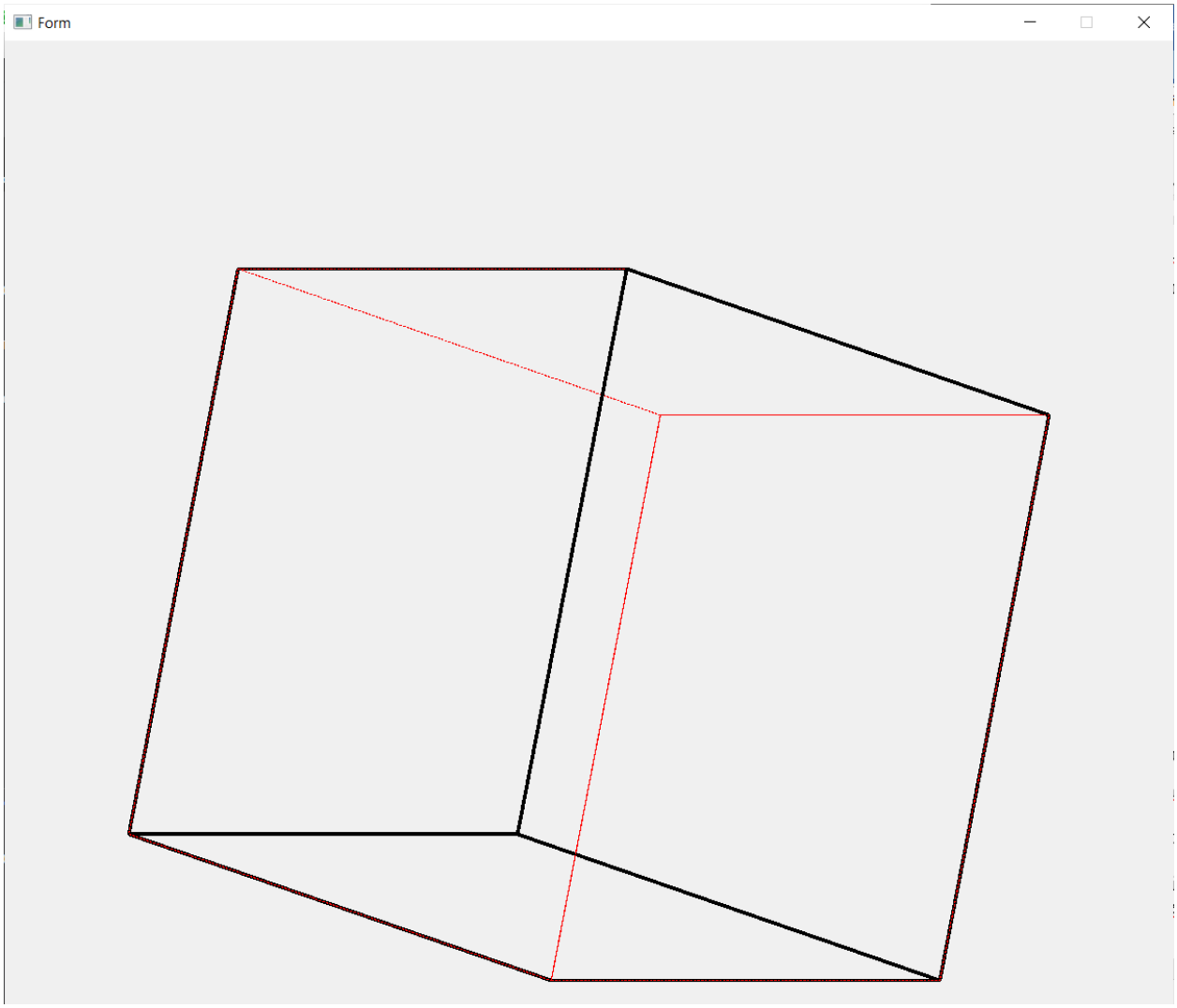


Рисунок 2 – Окно «Form»

ПРИЛОЖЕНИЕ 1 – КОД ПРОГРАММЫ

Файл main.cpp

```
#include <application.h>

int main(int argc, char *argv[])
{
   Application a(argc, argv);
   return a.exec();
}
```

Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
   : QMainWindow(parent)
   , ui(new Ui::MainWindow)
{
   ui->setupUi(this);
}

MainWindow::~MainWindow()
{
   delete ui;
}

ControlState* MainWindow::get_state()
{
   ControlState *tmp = new ControlState;

   tmp->x_ang = static_cast <qreal> (ui->horizontalSlider_X->value()) / 10;
   tmp->y_ang = static_cast <qreal> (ui->horizontalSlider_Y->value()) / 10;
   tmp->z_ang = static_cast <qreal> (ui->horizontalSlider_Z->value()) / 10;

   return tmp;
}

void MainWindow::on_horizontalSlider_X_valueChanged(int value)
{
   Q_UNUSED(value);
   emit send_control(get_state());
```

```
}

void MainWindow::on_horizontalSlider_Y_valueChanged(int value)
{
   Q_UNUSED(value);
   emit send_control(get_state());
}

void MainWindow::on_horizontalSlider_Z_valueChanged(int value)
{
   Q_UNUSED(value);
   emit send_control(get_state());
}
```

Файл drawwindow.cpp

```
#include "drawwindow.h"
#include "ui_drawwindow.h"

DrawWindow::DrawWindow(QWidget *parent) :
   QWidget(parent),
   ui(new Ui::DrawWindow)
{
   ui->setupUi(this);
   d = nullptr;
}

DrawWindow::~DrawWindow()
{
   delete ui;
}

void DrawWindow::get_draw(DrawState *rds)
{
   d=rds;
   repaint();
}

void DrawWindow::paintEvent (QPaintEvent *event)
{
   Q_UNUSED(event);
   QPainter painter(this);

   support_state s;
```

```
    s.cw = 0.5*rect().width();
    s.ch = 0.5*rect().height();
    s.ew = (s.cw) / (MY_SIZE*2);
    s.eh = (s.ch) / (MY_SIZE*2);

    if (d != nullptr)
    {
        QPen vp;
        vp.setColor(Qt::black);
        vp.setWidth(3);
        QPen np;
        np.setColor(Qt::red);
        np.setStyle(Qt::DotLine);
        vp.setWidth(3);


        for (qsizetype i = 0; i < d->faces.length(); i++)
        {
            if (d->faces[i].visible)
            {
                painter.setPen(vp);
            }
            else
            {
                painter.setPen(np);
            }
            painter.drawPolygon(transform(d->faces[i],s),d->faces[i].points.length());
        }
    }
}

QPointF* DrawWindow::transform(Face f, support_state s)
{
    QPointF *tmp = new QPointF[f.points.length()];
    for (qsizetype i = 0; i < f.points.length(); i++)
    {
        tmp[i].setX(s.cw + f.points[i].x*s.ew);
        tmp[i].setY(s.ch - f.points[i].y*s.eh);
    }
    return tmp;
}
```

Файл application.cpp

```
#include "application.h"
```

```
#define SIZE 5

Application::Application(int argc, char *argv[])
   : QApplication(argc,argv)
{
   d = new DrawWindow;
   d->show();
   m = new MainWindow;
   m->show();
   default_points.push_back(PointF3D(-MY_SIZE,MY_SIZE,MY_SIZE));
   default_points.push_back(PointF3D(MY_SIZE,MY_SIZE,MY_SIZE));
   default_points.push_back(PointF3D(MY_SIZE,-MY_SIZE,MY_SIZE));
   default_points.push_back(PointF3D(-MY_SIZE,-MY_SIZE,MY_SIZE));
   default_points.push_back(PointF3D(-MY_SIZE,MY_SIZE,-MY_SIZE));
   default_points.push_back(PointF3D(MY_SIZE,MY_SIZE,-MY_SIZE));
   default_points.push_back(PointF3D(MY_SIZE,-MY_SIZE,-MY_SIZE));
   default_points.push_back(PointF3D(-MY_SIZE,-MY_SIZE,-MY_SIZE));
   O = PointF3D(0,0,0);
   H = PointF3D(0,0,MY_SIZE*50);

   connect(m,SIGNAL(send_control(ControlState*)),
         this,SLOT(get_control(ControlState*)));
   connect(this,SIGNAL(send_draw(DrawState*)),
         d,SLOT(get_draw(DrawState*)));
   emit m->send_control(m->get_state());
}

void Application::get_control(ControlState *c)
{
   DrawState *d = new DrawState;
   QVector <PointF3D> points;
   double x_ang = 2*M_PI*c->x_ang/360;
   double y_ang = 2*M_PI*c->y_ang/360;
   double z_ang = 2*M_PI*c->z_ang/360;

   Mmatrix Tx(3,3),Ty(3,3),Tz(3,3);

   //Матрица поворота по x
   Tx.data[0][0] = 1;
   Tx.data[0][1] = 0;
   Tx.data[0][2] = 0;
   Tx.data[1][0] = 0;
   Tx.data[1][1] = cos(x_ang);
   Tx.data[1][2] = -sin(x_ang);
```

```
Tx.data[2][0] = 0;
Tx.data[2][1] = sin(x_ang);
Tx.data[2][2] = cos(x_ang);

//Матрица поворота по y
Ty.data[0][0] = cos(y_ang);
Ty.data[0][1] = 0;
Ty.data[0][2] = sin(y_ang);
Ty.data[1][0] = 0;
Ty.data[1][1] = 1;
Ty.data[1][2] = 0;
Ty.data[2][0] = -sin(y_ang);
Ty.data[2][1] = 0;
Ty.data[2][2] = cos(y_ang);

//Матрица поворота по z
Tz.data[0][0] = cos(z_ang);
Tz.data[0][1] = -sin(z_ang);
Tz.data[0][2] = 0;
Tz.data[1][0] = sin(z_ang);
Tz.data[1][1] = cos(z_ang);
Tz.data[1][2] = 0;
Tz.data[2][0] = 0;
Tz.data[2][1] = 0;
Tz.data[2][2] = 1;

for (qsizetype i = 0; i < default_points.length(); i++)
{
   Mmatrix tmp(1,3),res(1,3);
   tmp.data[0][0] = default_points[i].x;
   tmp.data[0][1] = default_points[i].y;
   tmp.data[0][2] = default_points[i].z;

   res = ((tmp * Tx) *Ty) * Tz;

   points.push_back(PointF3D(res.data[0][0],res.data[0][1],res.data[0][2]));
}

d->faces.push_back(Face(points[0],points[1],points[2],points[3]));
d->faces.push_back(Face(points[0],points[4],points[5],points[1]));
d->faces.push_back(Face(points[3],points[2],points[6],points[7]));
d->faces.push_back(Face(points[0],points[4],points[7],points[3]));
d->faces.push_back(Face(points[1],points[5],points[6],points[2]));
d->faces.push_back(Face(points[4],points[7],points[6],points[5]));
```

```
    for (qsizetype i = 0; i < d->faces.length(); i++)
    {
        d->faces[i].visible = is_visible(d->faces[i]);
    }

    emit send_draw(d);
    return;
}

bool Application::is_visible(Face f)
{
    Mmatrix To(3,3),Th(3,3);

    //Матрица для O
    To.data[0][0] = O.x-f.points[0].x;
    To.data[0][1] = O.y-f.points[0].y;
    To.data[0][2] = O.z-f.points[0].z;
    To.data[1][0] = f.points[1].x-f.points[0].x;
    To.data[1][1] = f.points[1].y-f.points[0].y;
    To.data[1][2] = f.points[1].z-f.points[0].z;
    To.data[2][0] = f.points[2].x-f.points[0].x;
    To.data[2][1] = f.points[2].y-f.points[0].y;
    To.data[2][2] = f.points[2].z-f.points[0].z;

    //Матрица для H
    Th.data[0][0] = H.x-f.points[0].x;
    Th.data[0][1] = H.y-f.points[0].y;
    Th.data[0][2] = H.z-f.points[0].z;
    Th.data[1][0] = f.points[1].x-f.points[0].x;
    Th.data[1][1] = f.points[1].y-f.points[0].y;
    Th.data[1][2] = f.points[1].z-f.points[0].z;
    Th.data[2][0] = f.points[2].x-f.points[0].x;
    Th.data[2][1] = f.points[2].y-f.points[0].y;
    Th.data[2][2] = f.points[2].z-f.points[0].z;

    if (To.determinant()*Th.determinant() < 0)
    {
        return true;
    }
    {
        return false;
    }
}
```