МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

ОТЧЕТ

по лабораторной работе №6

по дисциплине «Компьютерная графика»

Тема: Формирования реалистических изображений с использованием простых моделей освещения одним или двумя точечными источниками

Студенты гр. 9308 Степовик В.С.
Соболев М.С.
Дубенков С.А.

Преподаватель Матвеева И.В.

Санкт-Петербург
2022

Содержание

Цель работы

Сформировать реалистическое изображение с использованием простых моделей освещения одним или двумя точечными источниками.

Задание

Сформировать тени при освещении многоугольников и поверхностей, сформированных при выполнении темы 5, точечным источником освещения без учета интенсивности освящения тел, участвующих в сцене (без учета зеркальной и диффузионной составляющих освещения). Обеспечить преобразование сцены при изменении координат источников освещения или наблюдателя.

Используемые ресурсы

Для выполнения лабораторной работы использовался язык C++ с использованием фреймворка QT, с применением единственной функции для отрисовки void QPainter::drawPoint(int x, int y), которая размещает пиксель согласно поданным координатам.

Основные теоретические положения

Предположим, у нас есть треугольник, который проецируется на экран. Ко всему прочему добавляется источник освещения S. Получим следующую картинку:

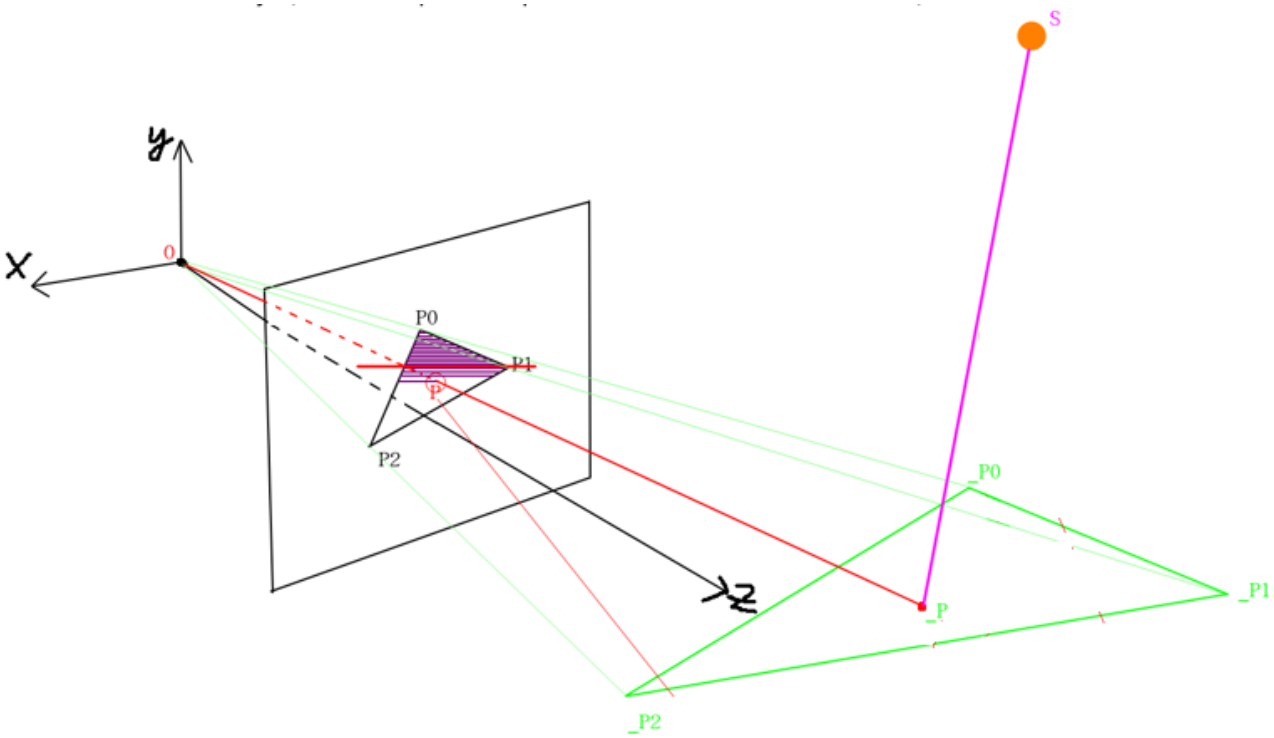

Рисунок 1. Проецирование треугольника с добавлением освещения

Если на одной плоскости в 3D пространстве даны 4 точки A, B,C, 0, то, чтобы проверить, что треугольник ABC содержит точку O, нужно найти b и c из уравнения:

$$\overrightarrow{AO} = b\overrightarrow{AB} + c\overrightarrow{AC}$$

Если выполняются одновременно условия:

$$b \geq 0, c \geq 0, b + c \leq 1$$

То точка O принадлежит треугольнику ABC.

Найдя точку _P, нужно проверить со всеми другими треугольниками (они же полигоны), что _PS не пересекается с ними. Если пересекается хоть с одним, то нужно “затемнить” этот пиксель на экране в точке P.

Также нужно убедиться, что пересечение будет именно с теми треугольниками, с которыми необходимо:

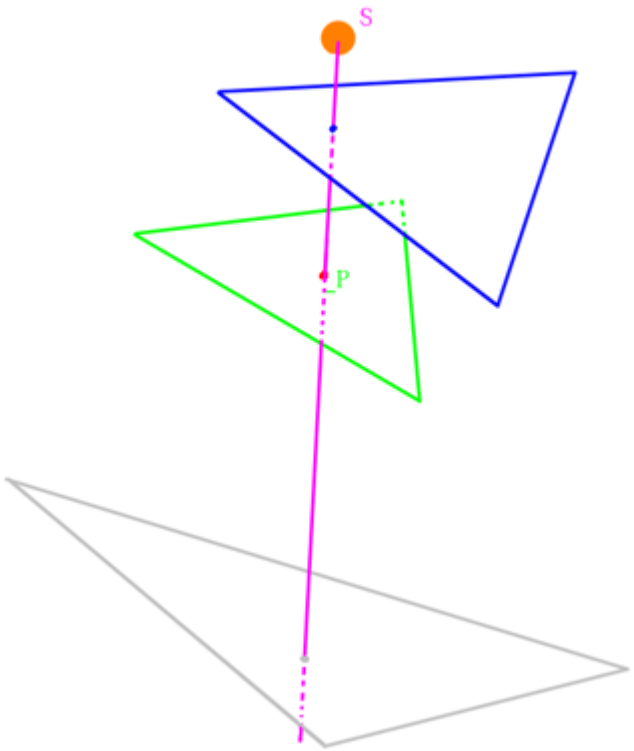

Рисунок 2. Треугольники для рассмотрения

Рассмотрим относительно зеленого треугольника:

- не нужно искать пересечение с изначальным треугольником
- пересечение с серым треугольником на данной картинке не должно означать, что пиксель на экране нужно будет затемнить. Ведь серый треугольник “сзади”.
- Если пересечение произошло с синим треугольником, то вот тогда затемнить нужно, ведь синий треугольник “загораживает” источник освещения в точке _P.

Уравнение для прямой _PS будет иметь следующий вид:

K = _P + (S - _P)*t, где K – точка на прямой _PS.

Если t < 0 или t > 1, то рассматривать пересечение с таким треугольником не нужно.

Фрагменты кода

Отрисовка точки на экране

```
void MainWindow::putPointOnScreen(int x, int y, unsigned **display, unsigned
colo, double **z_buffer, const Triangle &tri)
{
    Point O(0.0, 0.0, 0.0);
    double _x = ((double)x*r*2.0)/((double)W-1) - r;
    double _y = ((double)y*t*2.0)/((double)H-1) - t;
    Point P(_x, n, _y);

    Point crossP = tri.crossLine(O, P);
    double z_crossed = crossP.y();

    if(z_crossed < z_buffer[x][y])
    {
        if(n <= z_crossed && z_crossed <= f)
        {
            z_buffer[x][y] = z_crossed;
            size_t display_y = (size_t)y;
            size_t display_x = (size_t)x;
            Point cam_lightP = projectionOnCamera(lightPoint);
            unsigned shadowColo = shadow(crossP, colo, cam_lightP, h_lightPoint,
h_ambientLighting, *cam_polis, tri);
            display[H-1-display_y][display_x] = shadowColo;
        }
    }
}
```

Отрисовка полигона

```
void MainWindow::drawPoligon3D(const Triangle &tri, unsigned colo)
{
    Triangle tri_camera(projectionOnCamera(tri.p1()),
                        projectionOnCamera(tri.p2()),
                        projectionOnCamera(tri.p3()) );
    if(tri_camera.p1().y() < n && tri_camera.p2().y() < n && tri_camera.p3().y()
< n)
        return;
    Point A;
    Point B;
    Point C;
    if(tri_camera.p1().y() < 0)
    {
        double x_ = -tri_camera.p1().x();
        double z_ = -tri_camera.p1().z();
        double y_ =  tri_camera.p1().y();

        double xp = ((n*x_)/y_);
        double zp = ((n*z_)/y_);
        double x_res, y_res;
        x_res = ((xp+r)*(W-1))/(r+r);
        y_res = ((zp+t)*(H-1))/(t+t);
        A.copy(Point(x_res, y_res, 0.0));
    }
    else
        A.copy(projection3DOn2D(tri.p1()));

    if(tri_camera.p2().y() < 0)
    {
        double x_ = -tri_camera.p2().x();
```

```
        double z_ = -tri_camera.p2().z();
        double y_ =  tri_camera.p2().y();

        double xp = ((n*x_)/y_);
        double zp = ((n*z_)/y_);
        double x_res, y_res;
        x_res = ((xp+r)*(W-1))/(r+r);
        y_res = ((zp+t)*(H-1))/(t+t);
        B.copy(Point(x_res, y_res, 0.0));
    }
    else
        B.copy(projection3DOn2D(tri.p2()));

    if(tri_camera.p3().y() < 0)
    {
        double x_ = -tri_camera.p3().x();
        double z_ = -tri_camera.p3().z();
        double y_ =  tri_camera.p3().y();

        double xp = ((n*x_)/y_);
        double zp = ((n*z_)/y_);
        double x_res, y_res;
        x_res = ((xp+r)*(W-1))/(r+r);
        y_res = ((zp+t)*(H-1))/(t+t);
        C.copy(Point(x_res, y_res, 0.0));
    }
    else
        C.copy(projection3DOn2D(tri.p3()));

    int x0 = (int)(A.x()), y0 = (int)(A.y());
    int x1 = (int)(B.x()), y1 = (int)(B.y());
    int x2 = (int)(C.x()), y2 = (int)(C.y());

    int tmp = 0;
    if(y0 > y1)
    {
        tmp = y0;
        y0 = y1;
        y1 = tmp;
        tmp = x0;
        x0 = x1;
        x1 = tmp;
    }
    if(y0 > y2)
    {
        tmp = y0;
        y0 = y2;
        y2 = tmp;
        tmp = x0;
        x0 = x2;
        x2 = tmp;
    }
    if(y1 > y2)
    {
        tmp = y1;
        y1 = y2;
        y2 = tmp;
        tmp = x1;
        x1 = x2;
        x2 = tmp;
    }

    int cross_x1 = 0, cross_x2 = 0;
```

```
    int dx1 = x1 - x0;
    int dy1 = y1 - y0;
    int dx2 = x2 - x0;
    int dy2 = y2 - y0;

    int top_y = y0;

    while(top_y < y1)
    {
        cross_x1 = x0 + dx1 * (top_y - y0) / dy1;
        cross_x2 = x0 + dx2 * (top_y - y0) / dy2;
        printLineBeziers(cross_x1, top_y, cross_x2, top_y, colo, display,
z_buffer, tri_camera);
        ++top_y;
    }

    dx1 = x2 - x1;
    dy1 = y2 - y1;

    while(top_y < y2)
    {
        cross_x1 = x1 + dx1 * (top_y - y1) / dy1;
        cross_x2 = x0 + dx2 * (top_y - y0) / dy2;
        printLineBeziers(cross_x1, top_y, cross_x2, top_y, colo, display,
z_buffer, tri_camera);
        ++top_y;
    }
}
```

Пример работы программы

При запуске программы открывается следующее окно, где отображается стартовая сцена без выявления теней:

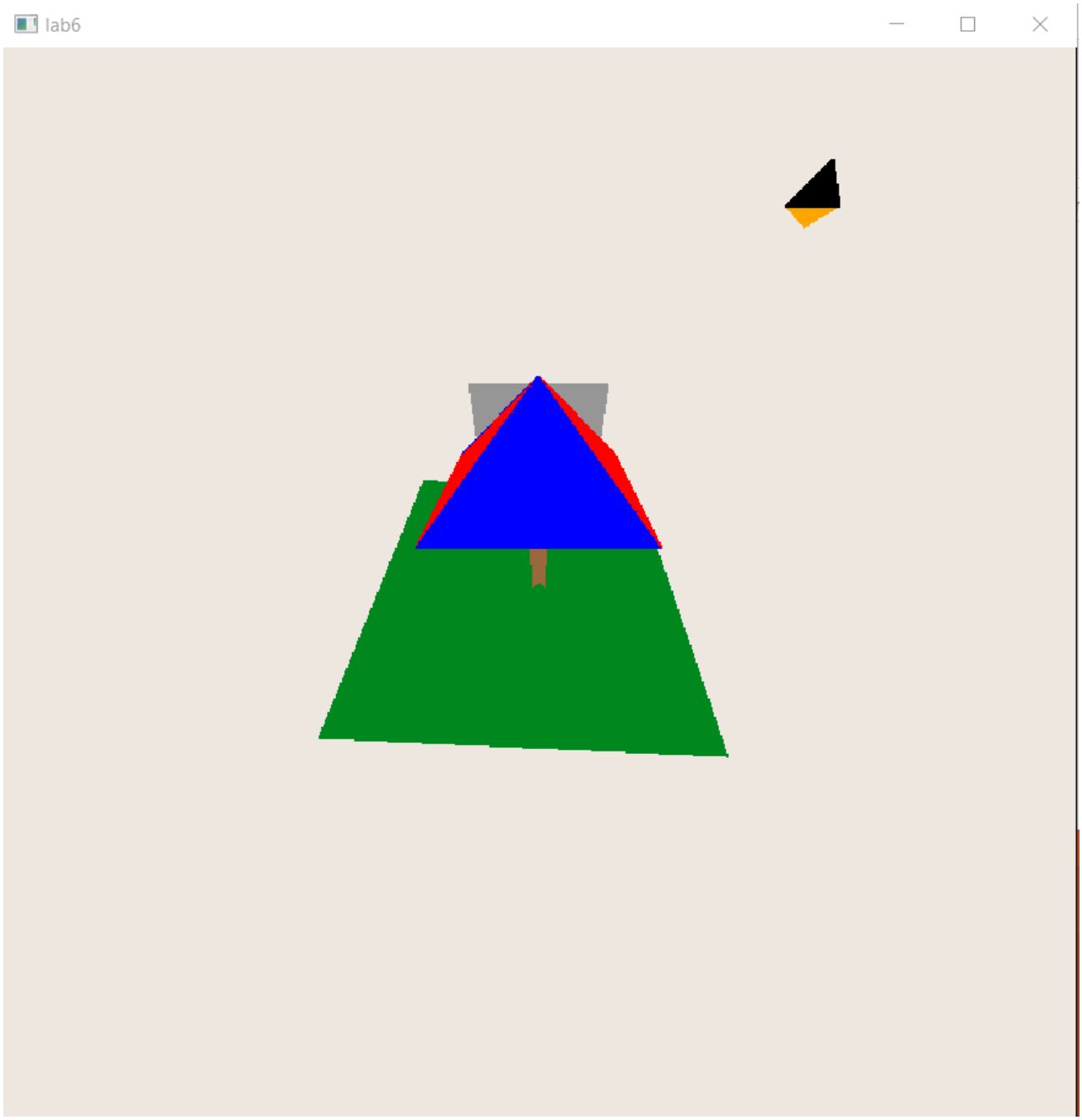

Рисунок 3. Стартовое окно программы

Источником света на сцене является оранжево-чёрный тетраэдр, который мигает чёрно-оранжевым, при его перемещении. Сцена запускается при нажатии любой кнопки (отображаются тени):

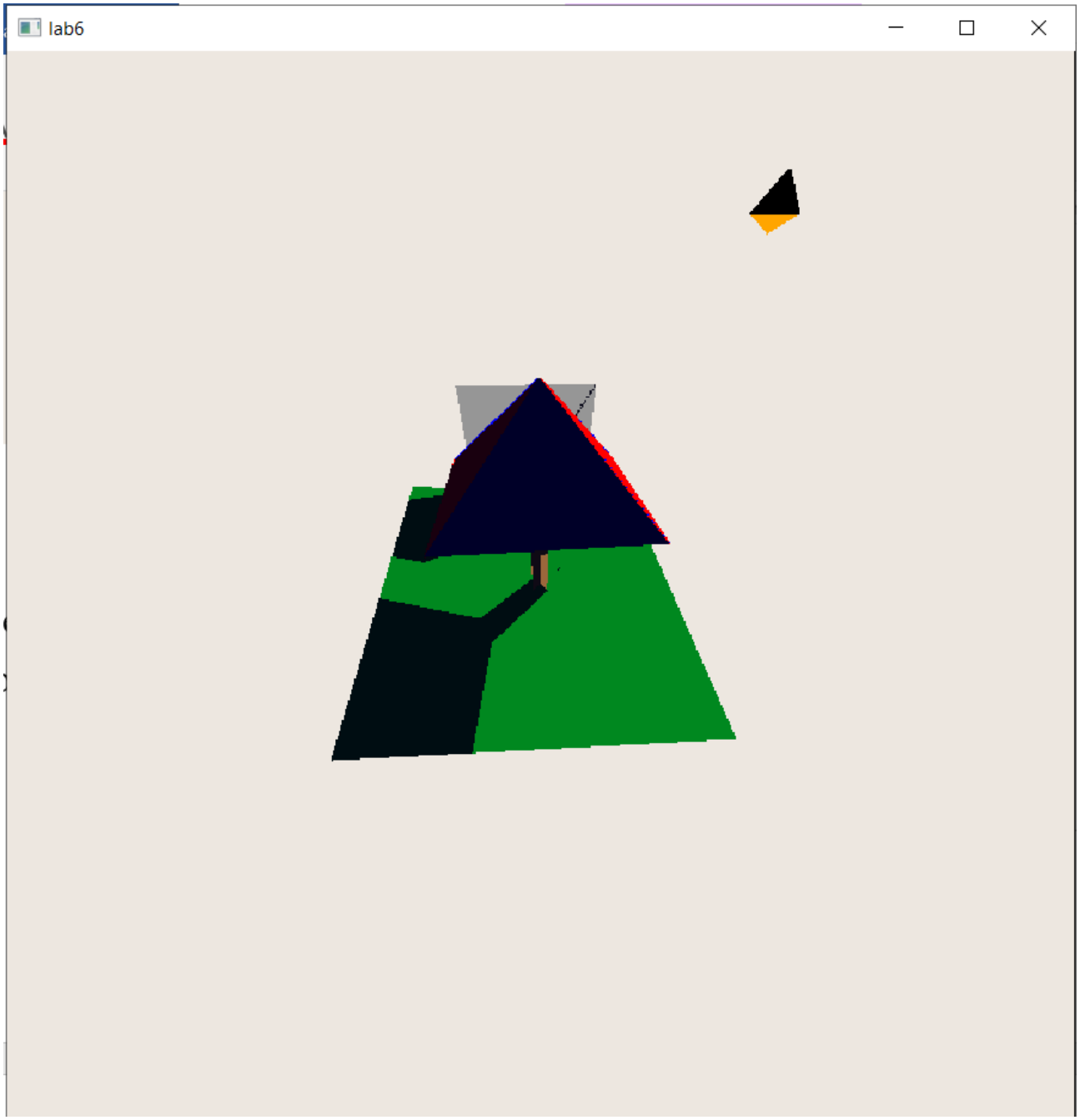

Рисунок 4. Стартовое окно программы (сцена)

Далее представлены различные кадры после изменения положения камеры и источника освещения относительно сцены:

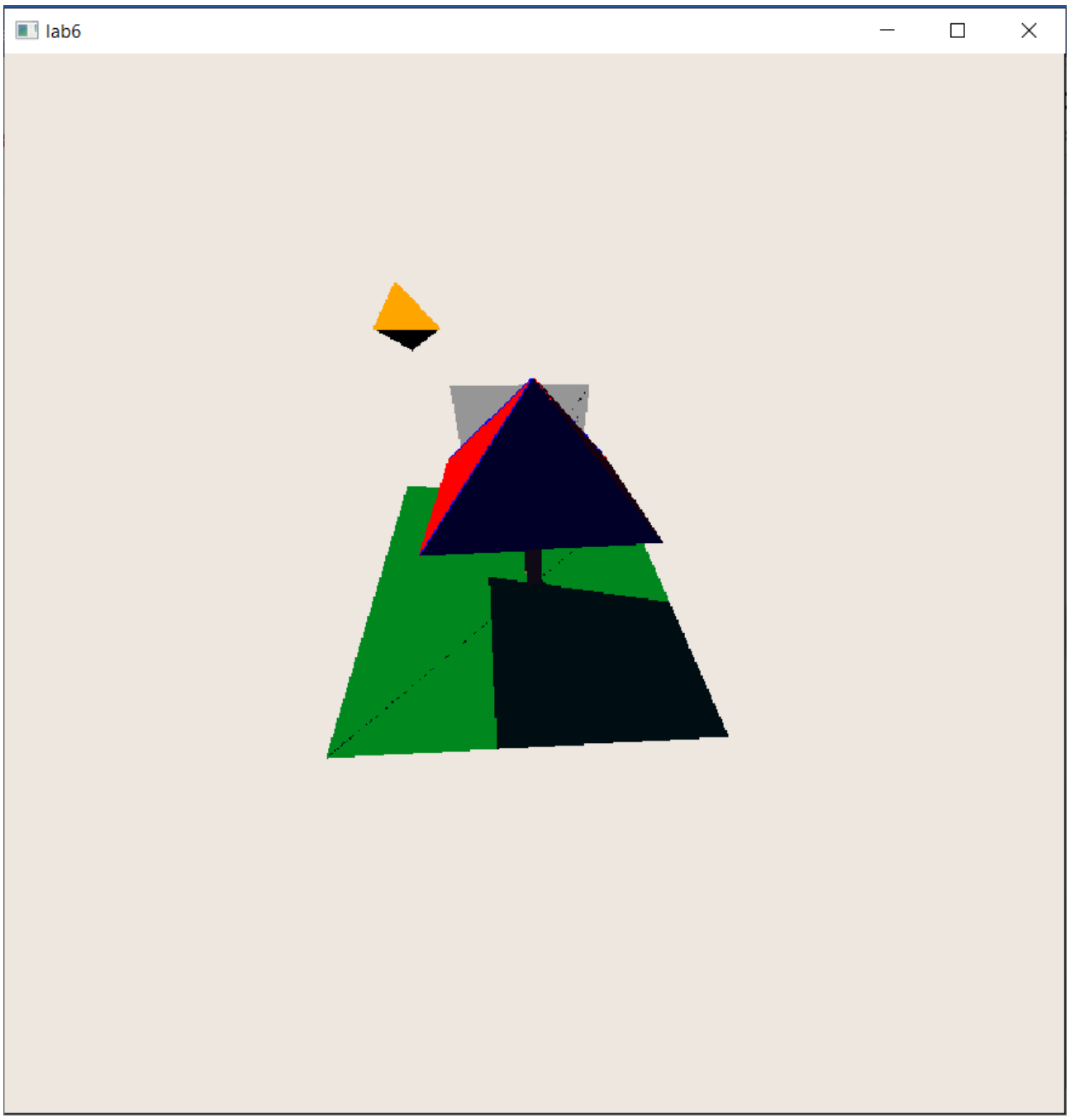

Рисунок 5. Кадр из приложения

На кадре ниже мы видим, что при отсутствии источника света (“солнце” спрятано за “стену” – серый прямоугольник) все многоугольники на сцене покрываются тенью.

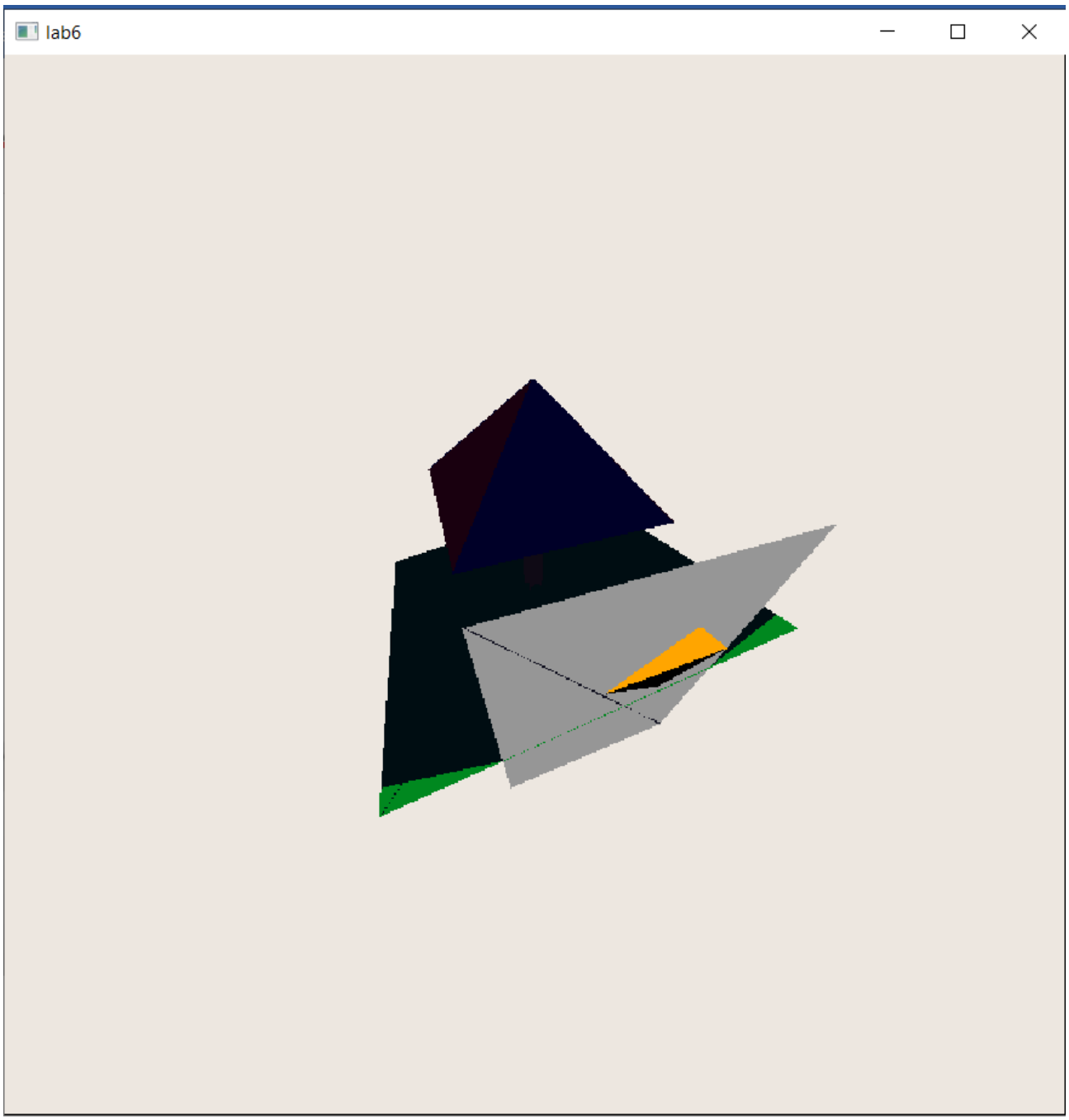

Рисунок 6. Кадр из приложения

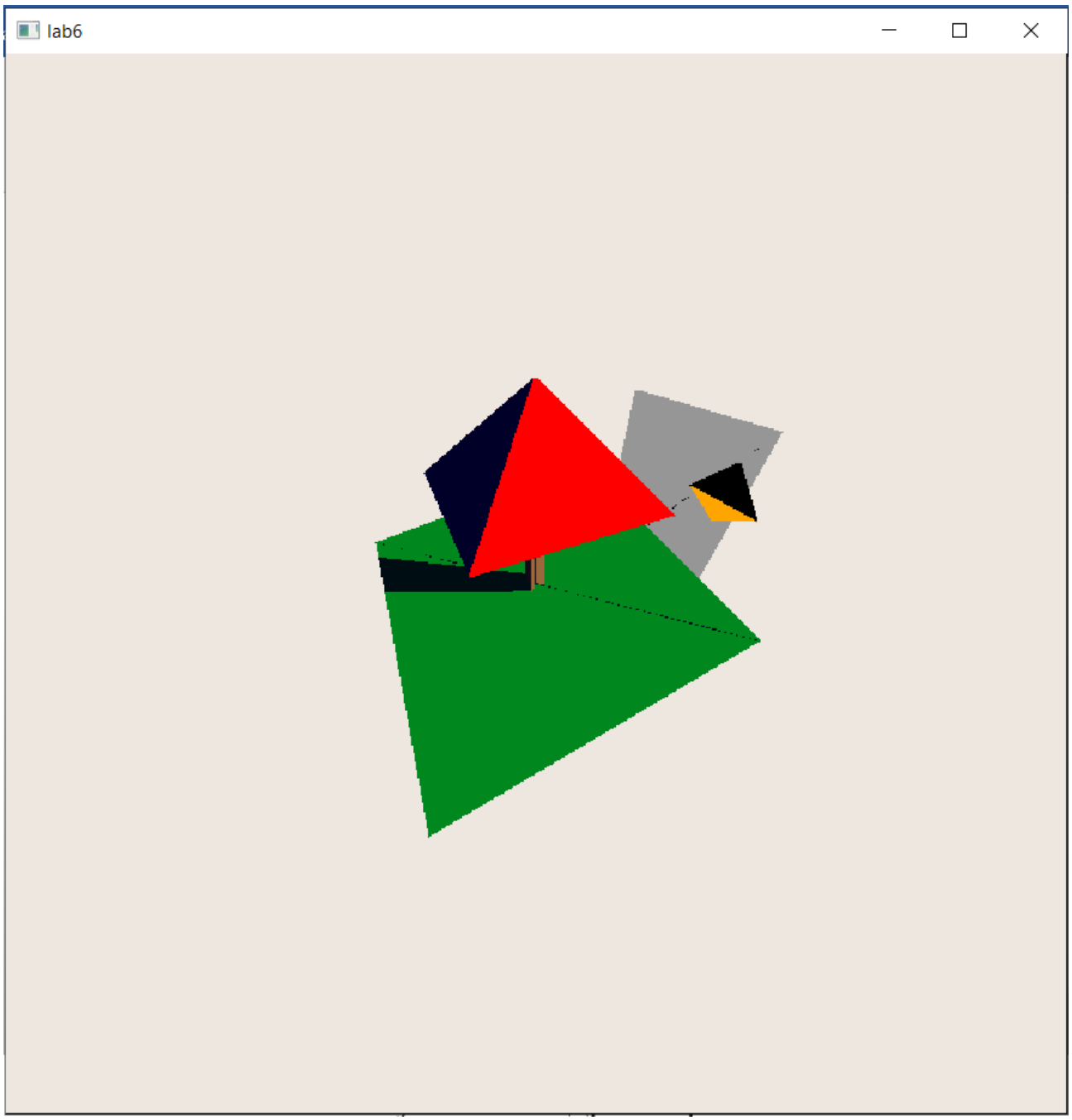

Рисунок 7. Кадр из приложения

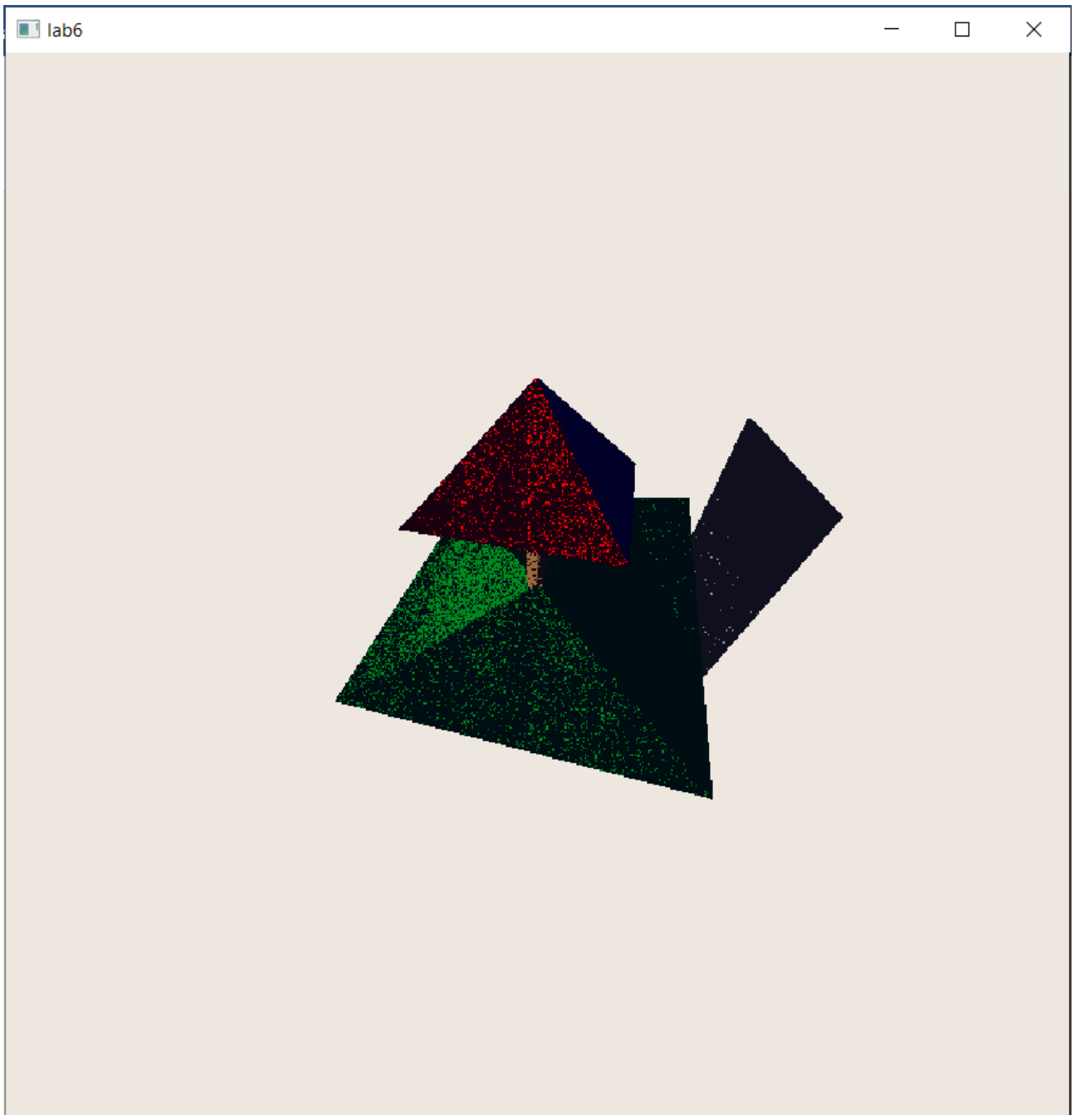

Рисунок 8. Кадр из приложения

Так будет выглядеть сцена, если скрыть источник освещения внутри пирамиды.

Вывод

При выполнении лабораторной работы были сформировано реалистическое изображение с использованием простых моделей освещения одним или двумя точечными источниками. В частности, было сформированы тени при освещении многоугольников и поверхностей точечным источником освещения без учета интенсивности освящения тел, участвующих в сцене (без учета зеркальной и диффузионной составляющих освещения).

Исходный код программы

Файл mainWindow.cpp

```
#include <vector>

#include <QWidget>
#include <QPainter>
#include <QKeyEvent>
#include <QColor>
#include <cmath>
#include <iostream>
#include <limits>

#include "./include/mainWindow.h"

#include "./include/Point.h"
#include "./include/Triangle.h"
#include "./include/matrix.h"


using namespace std;

double deg2rad(double a) { return (M_PI*a) / 180.0; }

unsigned calcColour(unsigned char r, unsigned char g, unsigned char b)
{
    unsigned res = 0, buff = 0;
    buff = b;
    res |= buff << 0;
    buff = g;
    res |= buff << 8;
    buff = r;
    res |= buff << 16;
    buff = 255;
    res |= buff << 24;
    return res;
}

unsigned calcColour2(unsigned rgb, unsigned char *r, unsigned char *g, unsigned
char *b)
{
    unsigned buff;

    buff = rgb >> 0;
    //*alpha = (unsigned char)buff;
    *b = (unsigned char)buff;

    buff = rgb >> 8;
    *g = (unsigned char)buff;

    buff = rgb >> 16;
    *r = (unsigned char)buff;

    return rgb;
}

MainWindow::MainWindow(QWidget *parent) : QWidget(parent)
{
```

```
    FIRST_ENTER = true;
    cam = new Camera();

    resize(W, H);
    this->setStyleSheet("background-color: rgb(200,200,200); margin:0px;
border:1px solid rgb(0, 0, 0); ");

    double newy = cam_begin_R*cos(deg2rad(cam_begin_angle));
    double newz = cam_begin_R*sin(deg2rad(cam_begin_angle));
    cam->move(0.0, -newy, newz);
    // cam.rotateOX(cam_begin_angle);
    cranch(false);

    lightPoint.copy(Point(20, 20, 20));
    h_lightPoint = 2;
    lol = 0;

    polis =     new vector<Triangle>();
    cam_polis = new vector<Triangle>();
    cols =      new vector<unsigned>();

    C_ = NULL;
    refresh_C_();

    display = new unsigned*[H];
    for(size_t li = 0; li < H; ++li)
        display[li] = new unsigned[W];
    refresh_display();

    z_buffer = new double*[W];
    for(size_t li = 0; li < W; ++li)
        z_buffer[li] = new double[H];
    refresh_z_buffer();
}

MainWindow::~MainWindow()
{
    if(polis != NULL)
        delete polis;
    if(cam_polis != NULL)
        delete cam_polis;
    if(cols != NULL)
        delete cols;

    delete cam;

    if(C_ != NULL)
        delete C_;

    for(size_t li = 0; li < H; ++li)
        delete display[li];
    delete display;

    for(size_t li = 0; li < W; ++li)
        delete z_buffer[li];
    delete z_buffer;
}

void MainWindow::paintEvent(QPaintEvent *e)
{
    Q_UNUSED(e);

    refresh_display();
```

```
    refresh_z_buffer();
    if(FIRST_ENTER)
    {
        FIRST_ENTER = false;

        putRectangle3D(Point(-10, 10, 3), Point(10, 10, 0), Point(-10, -10, 0),
Point(10, -10, -3),
                       calcColour(0, 136, 31));

        putRectangle3D(Point(-5, 10, 10), Point(5, 10, 10), Point(-5, 10, -5),
Point(5, 10, -5),
                       calcColour(150, 150, 150));

        putRectangle3D(Point(-0.5, -0.5, 10), Point(0.5, 0.5, 10), Point(-0.5, -
0.5, -5), Point(0.5, 0.5, -5),
                       calcColour(155, 103, 60));
        putRectangle3D(Point(-0.5, 0.5, 10), Point(0.5, -0.5, 10), Point(-0.5,
0.5, -5), Point(0.5, -0.5, -5),
                       calcColour(155, 103, 60));
        putTriangle3D(Triangle(Point(-5, 5, 7), Point(-5, -5, 7), Point(0, 0,
12)),
                              calcColour(255, 0, 0));
        putTriangle3D(Triangle(Point(5, -5, 7), Point(5, 5, 7), Point(0, 0,
12)),
                              calcColour(255, 0, 0));
        putTriangle3D(Triangle(Point(5, 5, 7), Point(-5, 5, 7), Point(0, 0,
12)),
                              calcColour(0, 0, 255));
        putTriangle3D(Triangle(Point(5, -5, 7), Point(-5, -5, 7), Point(0, 0,
12)),
                              calcColour(0, 0, 255));
    }

    unsigned int col_i = 0;
    for(Triangle &el : *polis)
        drawPoligon3D(el, (*cols)[col_i++]);


    Triangle lightTri(
        Point(lightPoint.x()-2, lightPoint.y(), lightPoint.z()-1),
        Point(lightPoint.x()+2, lightPoint.y(), lightPoint.z()-1),
        Point(lightPoint.x(), lightPoint.y(), lightPoint.z()+1)
    );
    if(lol % 2 == 0)
        drawPoligon3D(lightTri, calcColour(255, 165, 0));
    else
        drawPoligon3D(lightTri, calcColour(0, 0, 0));
    Triangle lightTri2(
        Point(lightPoint.x()-2, lightPoint.y(), lightPoint.z()-1),
        Point(lightPoint.x()+2, lightPoint.y(), lightPoint.z()-1),
        Point(lightPoint.x(), lightPoint.y(), lightPoint.z()-2)
    );
    if(lol % 2 == 0)
        drawPoligon3D(lightTri2, calcColour(0, 0, 0));
    else
        drawPoligon3D(lightTri2, calcColour(255, 165, 0));
    ++lol;

    QPainter qp(this);
    for(size_t li = 0; li < H; ++li)
        for(size_t lj = 0; lj < W; ++lj)
            {
                qp.setPen(QColor(display[li][lj]));
```

```
                qp.drawPoint(lj, li);
            }
}

void MainWindow::cranch(bool where)
{
    const double drotate = 1;
    int times = (int)(cam_begin_angle/drotate + 0.5);
    for(int i = 0; i < times; ++i)
    {
        cam->rotateOX(where?drotate:-drotate);
    }
}

void MainWindow::keyPressEvent(QKeyEvent *event)
{
    int key = event->key();

    double dd = 5;
    double d = 1;

    if(key == Qt::Key_Left)
    {
        //cam.rotateOX(60);
        cranch(true);

        double x_new, y_new;
        rotateVector(cam->o().x(), cam->o().y(), -dd, &x_new, &y_new);
        double dx = x_new - cam->o().x();
        double dy = y_new - cam->o().y();
        cam->move(dx, dy, 0.0);
        cam->rotateOZ(-dd);

        //cam.rotateOX(-60);
        cranch(false);
    }
    else if(key == Qt::Key_Right)
    {
        //cam.rotateOX(60);
        cranch(true);

        double x_new, y_new;
        rotateVector(cam->o().x(), cam->o().y(), dd, &x_new, &y_new);
        double dx = x_new - cam->o().x();
        double dy = y_new - cam->o().y();
        cam->move(dx, dy, 0.0);
        cam->rotateOZ(dd);

        //cam.rotateOX(-60);
        cranch(false);
    }
    else if(key == Qt::Key_A) // -x
    {
        lightPoint.add(Point(-d, 0.0, 0.0));
    }
    else if(key == Qt::Key_D) // +x
    {
        lightPoint.add(Point(d, 0.0, 0.0));
    }
    else if(key == Qt::Key_W) // +y
    {
        lightPoint.add(Point(0.0, d, 0.0));
    }
```

```
    else if(key == Qt::Key_S) // -y
    {
        lightPoint.add(Point(0.0, -d, 0.0));
    }
    else if(key == Qt::Key_C) // +z
    {
        lightPoint.add(Point(0.0, 0.0, d));
    }
    else if(key == Qt::Key_X) // -z
    {
        lightPoint.add(Point(0.0, 0.0, -d));
    }

    refresh_C_();
    update();
}

void MainWindow::rotateVector(double x_old, double y_old, double angle_degrees,
double *x_new, double *y_new)
{
    double a = deg2rad(angle_degrees);

    double x = x_old, y = y_old;

    double si = sin(a);
    double co = cos(a);

    *x_new = x*co - y*si;
    *y_new = x*si + y*co;
}

Point MainWindow::projectionOnCamera(const Point &p)
{
    return projectionOnCamera(p.x(), p.y(), p.z());
}

Point MainWindow::projectionOnCamera(double x, double y, double z)
{
    Point cam_o = cam->o();
    double obj_x = x - cam_o.x();
    double obj_y = y - cam_o.y();
    double obj_z = z - cam_o.z();

    Matrix<double> old_v(3, 1);
    old_v.set(obj_x, 0, 0); old_v.set(obj_y, 1, 0); old_v.set(obj_z, 2, 0);

    Matrix<double> new_v = C_->multiply(old_v);

    double x_ = new_v.get(0, 0);
    double y_ = new_v.get(1, 0);
    double z_ = new_v.get(2, 0);

    return Point(x_, y_, z_);
}

Point MainWindow::projection3DOn2D(const Point &p)
{
    return projection3DOn2D(p.x(), p.y(), p.z());
}

Point MainWindow::projection3DOn2D(double x, double y, double z)
{
    Point cam_o = cam->o();
```

```
    double obj_x = x - cam_o.x();
    double obj_y = y - cam_o.y();
    double obj_z = z - cam_o.z();

    Matrix<double> old_v(3, 1);
    old_v.set(obj_x, 0, 0); old_v.set(obj_y, 1, 0); old_v.set(obj_z, 2, 0);

    Matrix<double> new_v = C_->multiply(old_v);
    double x_ = new_v.get(0, 0);
    double y_ = new_v.get(1, 0);
    double z_ = new_v.get(2, 0);

    double xp, zp;

    xp = ((n*x_)/y_);
    zp = ((n*z_)/y_);

    double x_res, y_res;
    x_res = ((xp+r)*(W-1))/(r+r);
    y_res = ((zp+t)*(H-1))/(t+t);

    return Point(x_res, y_res, 0.0);
}

bool MainWindow::isOnDisplay(int x, int y)
{
    if(x < 0 || x >= (int)W)
        return false;
    if(y < 0 || y >= (int)H)
        return false;
    return true;
}

void MainWindow::refresh_C_()
{
    Point vx(cam->vr());
    Point vy(cam->vf());
    Point vz(cam->vu());

    Matrix<double> C(3, 3);
    C.set(vx.x(), 0, 0); C.set(vy.x(), 0, 1); C.set(vz.x(), 0, 2);
    C.set(vx.y(), 1, 0); C.set(vy.y(), 1, 1); C.set(vz.y(), 1, 2);
    C.set(vx.z(), 2, 0); C.set(vy.z(), 2, 1); C.set(vz.z(), 2, 2);
    //std::cout << C.toString() << std::endl;
    Matrix<double> C_buff = C.inverse();

    if(C_ != NULL)
        delete C_;
    C_ = new Matrix<double>(C_buff);

    for(size_t i = 0; i < cam_polis->size(); ++i)
    {
        Triangle tri_camera(   projectionOnCamera((*polis)[i].p1()),
                               projectionOnCamera((*polis)[i].p2()),
                               projectionOnCamera((*polis)[i].p3()) );
        (*cam_polis)[i].copy(tri_camera);
    }
}

void MainWindow::refresh_display()
{
    for(size_t li = 0; li < H; ++li)
        for(size_t lj = 0; lj < W; ++lj)
```

```
            display[li][lj] = GlobalBackgroundColor;
}

void MainWindow::refresh_z_buffer()
{
    double max_double = numeric_limits<double>::infinity();
    for(size_t li = 0; li < W; ++li)
        for(size_t lj = 0; lj < H; ++lj)
            z_buffer[li][lj] = max_double;
}

unsigned enhanceColour(unsigned rgb, double k)
{
    unsigned char r = 0, g = 0, b = 0;
    calcColour2(rgb, &r, &g, &b);

    int _r = r, _g = g, _b = b;

    _r = (int)((double)r*k);
    _r = _r>255?255:_r;
    _r = _r<0?0:_r;

    _g = (int)((double)g*k);
    _g = _g>255?255:_g;
    _g = _g<0?0:_g;

    _b = (int)((double)b*k);
    _b = _b>255?255:_b;
    _b = _b<0?0:_b;

    return calcColour((unsigned char)_r, (unsigned char)_g, (unsigned char)_b);
}

unsigned addColour(unsigned argb1, unsigned argb2)
{
    unsigned char r1 = 0, g1 = 0, b1 = 0;
    calcColour2(argb1, &r1, &g1, &b1);

    unsigned char r2 = 0, g2 = 0, b2 = 0;
    calcColour2(argb2, &r2, &g2, &b2);

    int _r = r1, _g = g1, _b = b1;

    _r += r2;
    _r = _r>255?255:_r;
    _r = _r<0?0:_r;

    _g += g2;
    _g = _g>255?255:_g;
    _g = _g<0?0:_g;

    _b += b2;
    _b = _b>255?255:_b;
    _b = _b<0?0:_b;

    return calcColour((unsigned char)_r, (unsigned char)_g, (unsigned char)_b);
}

unsigned MainWindow::shadow(const Point &P, unsigned colo, const Point &lightP,
double h_lightP, double h_world, vector<Triangle> &allPoli, const Triangle
&curTri)
{
```

```
    //sPoint P_lightP = sPoint(lightP.x() - P.x(), lightP.y() - P.y(),
lightP.z() - P.z());

    for(Triangle &poli_el : allPoli)
    {
        if(curTri.equals(poli_el) == false)
        {
            Point crossP;
            bool ifPoliCross = poli_el.crossLine2(P, lightP, crossP);
            if(ifPoliCross)
            {
                double t = (crossP.x() - P.x()) / (lightP.x() - P.x());
                if(0 <= t && t <= 1)
                {
                    (void)h_lightP; // =/

                    unsigned resColo = enhanceColour(colo, 0.1);
                    resColo = addColour(resColo, h_world);
                    return resColo;
                }
            }
        }
    }
    return colo;
}

void MainWindow::putPointOnScreen(int x, int y, unsigned **display, unsigned
colo, double **z_buffer, const Triangle &tri)
{
    Point O(0.0, 0.0, 0.0);
    double _x = ((double)x*r*2.0)/((double)W-1) - r;
    double _y = ((double)y*t*2.0)/((double)H-1) - t;
    Point P(_x, n, _y);

    Point crossP = tri.crossLine(O, P);
    double z_crossed = crossP.y();

    if(z_crossed < z_buffer[x][y])
    {
        if(n <= z_crossed && z_crossed <= f)
        {
            z_buffer[x][y] = z_crossed;
            size_t display_y = (size_t)y;
            size_t display_x = (size_t)x;
            Point cam_lightP = projectionOnCamera(lightPoint);
            unsigned shadowColo = shadow(crossP, colo, cam_lightP, h_lightPoint,
h_ambientLighting, *cam_polis, tri);
            display[H-1-display_y][display_x] = shadowColo;
        }
    }
}

void MainWindow::printLineBeziers(int x1, int y1, int x2, int y2, unsigned colo,
unsigned **display, double **z_buffer, const Triangle &tri)
{
    int deltaX = abs(x2 - x1);
    int deltaY = abs(y2 - y1);
    int signX = x1 < x2 ? 1 : -1;
    int signY = y1 < y2 ? 1 : -1;

    int error = deltaX - deltaY;

    //qp.drawPoint(x2, y2);
```

```
    if(isOnDisplay(x2, y2))
        putPointOnScreen(x2, y2, display, colo, z_buffer, tri);
    while(x1 != x2 || y1 != y2)
    {
        //qp.drawPoint(x1, y1);
        if(isOnDisplay(x1, y1))
            putPointOnScreen(x1, y1, display, colo, z_buffer, tri);
        int error2 = error * 2;
        //
        if(error2 > -deltaY)
        {
            error -= deltaY;
            x1 += signX;
        }
        if(error2 < deltaX)
        {
            error += deltaX;
            y1 += signY;
        }
    }
}

void MainWindow::drawPoligon3D(const Triangle &tri, unsigned colo)
{
    Triangle tri_camera(projectionOnCamera(tri.p1()),
                        projectionOnCamera(tri.p2()),
                        projectionOnCamera(tri.p3()) );
    if(tri_camera.p1().y() < n && tri_camera.p2().y() < n && tri_camera.p3().y()
< n)
        return;
    Point A;
    Point B;
    Point C;
    if(tri_camera.p1().y() < 0)
    {
        double x_ = -tri_camera.p1().x();
        double z_ = -tri_camera.p1().z();
        double y_ =  tri_camera.p1().y();

        double xp = ((n*x_)/y_);
        double zp = ((n*z_)/y_);
        double x_res, y_res;
        x_res = ((xp+r)*(W-1))/(r+r);
        y_res = ((zp+t)*(H-1))/(t+t);
        A.copy(Point(x_res, y_res, 0.0));
    }
    else
        A.copy(projection3DOn2D(tri.p1()));

    if(tri_camera.p2().y() < 0)
    {
        double x_ = -tri_camera.p2().x();
        double z_ = -tri_camera.p2().z();
        double y_ =  tri_camera.p2().y();

        double xp = ((n*x_)/y_);
        double zp = ((n*z_)/y_);
        double x_res, y_res;
        x_res = ((xp+r)*(W-1))/(r+r);
        y_res = ((zp+t)*(H-1))/(t+t);
        B.copy(Point(x_res, y_res, 0.0));
    }
    else
```

```
    B.copy(projection3DOn2D(tri.p2()));

if(tri_camera.p3().y() < 0)
{
    double x_ = -tri_camera.p3().x();
    double z_ = -tri_camera.p3().z();
    double y_ =  tri_camera.p3().y();

    double xp = ((n*x_)/y_);
    double zp = ((n*z_)/y_);
    double x_res, y_res;
    x_res = ((xp+r)*(W-1))/(r+r);
    y_res = ((zp+t)*(H-1))/(t+t);
    C.copy(Point(x_res, y_res, 0.0));
}
else
    C.copy(projection3DOn2D(tri.p3()));

int x0 = (int)(A.x()), y0 = (int)(A.y());
int x1 = (int)(B.x()), y1 = (int)(B.y());
int x2 = (int)(C.x()), y2 = (int)(C.y());

int tmp = 0;
if(y0 > y1)
{
    tmp = y0;
    y0 = y1;
    y1 = tmp;
    tmp = x0;
    x0 = x1;
    x1 = tmp;
}
if(y0 > y2)
{
    tmp = y0;
    y0 = y2;
    y2 = tmp;
    tmp = x0;
    x0 = x2;
    x2 = tmp;
}
if(y1 > y2)
{
    tmp = y1;
    y1 = y2;
    y2 = tmp;
    tmp = x1;
    x1 = x2;
    x2 = tmp;
}

int cross_x1 = 0, cross_x2 = 0;

int dx1 = x1 - x0;
int dy1 = y1 - y0;
int dx2 = x2 - x0;
int dy2 = y2 - y0;

int top_y = y0;

while(top_y < y1)
{
    cross_x1 = x0 + dx1 * (top_y - y0) / dy1;
```

```
        cross_x2 = x0 + dx2 * (top_y - y0) / dy2;
        printLineBeziers(cross_x1, top_y, cross_x2, top_y, colo, display,
z_buffer, tri_camera);
        ++top_y;
    }

    dx1 = x2 - x1;
    dy1 = y2 - y1;

    while(top_y < y2)
    {
        cross_x1 = x1 + dx1 * (top_y - y1) / dy1;
        cross_x2 = x0 + dx2 * (top_y - y0) / dy2;
        printLineBeziers(cross_x1, top_y, cross_x2, top_y, colo, display,
z_buffer, tri_camera);
        ++top_y;
    }
}

void MainWindow::putTriangle3D(const Triangle &tri, unsigned colo)
{
    polis->push_back(tri);
    cam_polis->push_back(tri);
    cols->push_back(colo);
}

void MainWindow::putRectangle3D(const Point &lu, const Point &ru, const Point
&ld, const Point &rd, unsigned colo)
{
    Triangle one1(lu, ru, ld);
    Triangle two2(ld, rd, ru);

    putTriangle3D(one1, colo);
    putTriangle3D(two2, colo);
}
```

Файл Point.cpp

```
#include <QWidget>
#include <QPainter>
#include <cmath>

#include "./include/Point.h"

Point::Point()
: _x(0.0), _y(0.0), _z(0.0){}

Point::Point(double X, double Y, double Z)
: _x(X), _y(Y), _z(Z){}

Point::Point(const Point& toCopied)
: _x(toCopied.getX()), _y(toCopied.getY()), _z(toCopied.getZ()){}

double Point::getX() const
{
    return _x;
}

double Point::getY() const
{
```

```
    return _y;
}

double Point::getZ() const
{
    return _z;
}

void Point::setX(double new_x)
{
    _x = new_x;
}

void Point::setY(double new_y)
{
    _y = new_y;
}

void Point::setZ(double new_z)
{
    _z = new_z;
}

double Point::x() const {return getX();}

double Point::y() const {return getY();}

double Point::z() const {return getZ();}

void Point::add(const Point& other)
{
    _x += other._x;
    _y += other._y;
    _z += other._z;
}

void Point::sub(const Point& other)
{
    _x -= other._x;
    _y -= other._y;
    _z -= other._z;
}

void Point::mul(double x)
{
    _x *= x;
    _y *= x;
    _z *= x;
}

std::string Point::print(std::string prefix) const
{
    std::string res = prefix + "(" + std::to_string(_x) + ", " +
std::to_string(_y) + ", " + std::to_string(_z) + ")";
    return res;
}

double Point::vector_len() const
{
    return sqrt(_x*_x + _y*_y + _z*_z);
}

void Point::copy(const Point& other)
```

```
{
    _x = other._x;
    _y = other._y;
    _z = other._z;
}
```

Файл matrix.cpp

```
#include <cstdlib>
#include <iostream>
#include <sstream>
#include <string>
#include <iomanip>
#include <cmath>

#include "./include/matrix.h"

using namespace std;

/*
template<typename T>
class Matrix;

template<typename T>
std::ostream& operator<<(std::ostream&, const Matrix<T>&);
*/

template <typename T>
Matrix<T>::Matrix(size_t n_row, size_t m_column): n(n_row), m(m_column)
{
    a = (T**)malloc(n*sizeof(T*));

    for(size_t i = 0; i < n; ++i)
        a[i] = (T*)malloc(m*sizeof(T));

    for(size_t i = 0; i < n; ++i)
        for(size_t j = 0; j < m; ++j)
            a[i][j] = 0;
}

template <typename T>
Matrix<T>::Matrix(const Matrix& clonner) : n(clonner.n), m(clonner.m)
{
    a = (T**)malloc(n*sizeof(T*));

    for(size_t i = 0; i < n; ++i)
        a[i] = (T*)malloc(m*sizeof(T));

    for(size_t i = 0; i < n; ++i)
        for(size_t j = 0; j < m; ++j)
            a[i][j] = clonner.a[i][j];
}

template<typename T>
Matrix<T>::~Matrix()
{
    for(size_t i = 0; i < n; ++i)
        free(a[i]);
    free(a);
}

template<typename T>
T Matrix<T>::get(size_t i, size_t j)
```

```
{
    if(/*i < 0 || */i >= n)
        throw printOutOfBound("i", i);
    if(/*j < 0 || */j >= m)
        throw printOutOfBound("j", j);
    return a[i][j];
}

template<typename T>
void Matrix<T>::set(T value, size_t i, size_t j)
{
    if(/*i < 0 || */i >= n)
        throw printOutOfBound("i", i);
    if(/*j < 0 || */j >= m)
        throw printOutOfBound("j", j);
    a[i][j] = value;
}

template<typename T>
Matrix<T> Matrix<T>::multiply(const Matrix& one, const Matrix& two)
{
    if(one.m != two.n)
        throw "Matrix is not multiplable (one.m_column = " + to_string(one.m) +
", two.n_row = " + to_string(two.n) + "). ";

    /*n1xm1 * n2xm2 -> n1xm2*/
    Matrix<T> res(one.n, two.m);

    for(size_t res_i = 0; res_i < res.n; ++res_i)
        for(size_t res_j = 0; res_j < res.m; ++res_j)
        {
            T buff = 0;
            for(size_t other_i = 0; other_i < one.m; ++other_i)
                buff += one.a[res_i][other_i] * two.a[other_i][res_j];
            res.a[res_i][res_j] = buff;
        }
    return res;
}

template<typename T>
Matrix<T> Matrix<T>::multiply(const Matrix& other) const
{
    return Matrix<T>::multiply(*this, other);
}

template<typename T>
Matrix<T> Matrix<T>::add(const Matrix& one, const Matrix& two)
{
    if(!(one.n == two.n && one.m == two.m))
        throw "Matrix is not addeble. Its size different. ";

    /*n1xm1 * n2xm2 -> n1xm2*/
    Matrix<T> res(one.n, two.m);

    for(size_t i = 0; i < res.n; ++i)
        for(size_t j = 0; j < res.m; ++j)
            res.a[i][j] = one.a[i][j] + two.a[i][j];
    return res;
}

template<typename T>
Matrix<T> Matrix<T>::add(const Matrix& other) const
{
```

```
    return Matrix<T>::add(*this, other);
}

template<typename T>
string Matrix<T>::toString()
{
    ostringstream res;
    for(size_t i = 0; i < n; ++i)
    {
        if(i == 0)
            res << "";
        for(size_t j = 0; j < m; ++j)
        {
            res << setw(5) << a[i][j];
            if(i == n-1)
            {
                if(j != m-1)
                    res << ", ";
            }
            else
                res << ", ";
        }
        if(i == n-1)
            res << "";
        else
            res << "\n";
    }
    return res.str();
}

template<typename T>
string Matrix<T>::str()
{
    return toString();
}

/*template<class T>
ostream& operator<< (ostream &out, const Matrix<T> &ma)
{
    out << ma.toString();
    return out;
}*/

template<typename T>
string Matrix<T>::printOutOfBound(string ij, size_t ij_val)
{
    return "Out of bounds: n_row = " + to_string(n) + ", m_column = " +
to_string(m) + ", and " + ij + " = " + to_string(ij_val) + ". ";
}

template<typename T>
Matrix<T> Matrix<T>::inverse() const
{
    if(this->n != this->m)
        throw "The matrix must be square: n=" + std::to_string(this->n) + ", " +
std::to_string(this->m) + ". ";
    Matrix<T> res(*this);
    bool check = matrix_inverse(this->a, res.a, this->n);
    if(check == false)
        throw "Cannot inverse this matrix. ";
    return res;
}
```

```
// Функция, производящая обращение матрицы.
// Принимает:
//     matrix - матрица для обращения
//     result - матрица достаточного размера для вмещения результата
//     size   - размерность матрицы
// Возвращает:
//     true в случае успешного обращения, false в противном случае
bool matrix_inverse(double **matrix, double **result, const int size)
{
    // Изначально результирующая матрица является единичной
    // Заполняем единичную матрицу
    for (int i = 0; i < size; ++i)
    {
        for (int j = 0; j < size; ++j)
            result[i][j] = 0.0;

        result[i][i] = 1.0;
    }

    // Копия исходной матрицы
    double **copy = new double *[size]();

    // Заполняем копию исходной матрицы
    for (int i = 0; i < size; ++i)
    {
        copy[i] = new double [size];

        for (int j = 0; j < size; ++j)
            copy[i][j] = matrix[i][j];
    }

    // Проходим по строкам матрицы (назовём их исходными)
    // сверху вниз. На данном этапе происходит прямой ход
    // и исходная матрица превращается в верхнюю треугольную
    for (int k = 0; k < size; ++k)
    {
        // Если элемент на главной диагонали в исходной
        // строке - нуль, то ищем строку, где элемент
        // того же столбца не нулевой, и меняем строки
        // местами
        if (fabs(copy[k][k]) < 1e-8)
        {
            // Ключ, говорящий о том, что был произведён обмен строк
            bool changed = false;

            // Идём по строкам, расположенным ниже исходной
            for (int i = k + 1; i < size; ++i)
            {
                // Если нашли строку, где в том же столбце
                // имеется ненулевой элемент
                if (fabs(copy[i][k]) > 1e-8)
                {
                    // Меняем найденную и исходную строки местами
                    // как в исходной матрице, так и в единичной
                    std::swap(copy[k],   copy[i]);
                    std::swap(result[k], result[i]);

                    // Взводим ключ - сообщаем о произведённом обмене строк
                    changed = true;

                    break;
                }
            }
```

```
            // Если обмен строк произведён не был - матрица не может быть
            // обращена
            if (!changed)
            {
                // Чистим память
                for (int i = 0; i < size; ++i)
                    delete [] copy[i];

                delete [] copy;

                // Сообщаем о неудаче обращения
                return false;
            }
        }

        // Запоминаем делитель - диагональный элемент
        double div = copy[k][k];

        // Все элементы исходной строки делим на диагональный
        // элемент как в исходной матрице, так и в единичной
        for (int j = 0; j < size; ++j)
        {
            copy[k][j]   /= div;
            result[k][j] /= div;
        }

        // Идём по строкам, которые расположены ниже исходной
        for (int i = k + 1; i < size; ++i)
        {
            // Запоминаем множитель - элемент очередной строки,
            // расположенный под диагональным элементом исходной
            // строки
            double multi = copy[i][k];

            // Отнимаем от очередной строки исходную, умноженную
            // на сохранённый ранее множитель как в исходной,
            // так и в единичной матрице
            for (int j = 0; j < size; ++j)
            {
                copy[i][j]   -= multi * copy[k][j];
                result[i][j] -= multi * result[k][j];
            }
        }
    }

    // Проходим по вернхней треугольной матрице, полученной
    // на прямом ходе, снизу вверх
    // На данном этапе происходит обратный ход, и из исходной
    // матрицы окончательно формируется единичная, а из единичной -
    // обратная
    for (int k = size - 1; k > 0; --k)
    {
        // Идём по строкам, которые расположены выше исходной
        for (int i = k - 1; i + 1 > 0; --i)
        {
            // Запоминаем множитель - элемент очередной строки,
            // расположенный над диагональным элементом исходной
            // строки
            double multi = copy[i][k];

            // Отнимаем от очередной строки исходную, умноженную
            // на сохранённый ранее множитель как в исходной,
```

```
            // так и в единичной матрице
            for (int j = 0; j < size; ++j)
            {
                copy[i][j]   -= multi * copy[k][j];
                result[i][j] -= multi * result[k][j];
            }
        }
    }

    // Чистим память
    for (int i = 0; i < size; ++i)
        delete [] copy[i];

    delete [] copy;

    // Сообщаем об успехе обращения
    return true;
}

// https://bytefreaks.net/programming-2/c/c-undefined-reference-to-templated-class-function

template class Matrix<double>;
      //template class Matrix<int>;
```

Файл Camera.cpp

```
#include <cmath>
#include <string>

#include "./include/Point.h"
#include "./include/Camera.h"

Camera::Camera() : O(0.0, 0.0, 0.0), F(0.0, 1.0, 0.0), U(0.0, 0.0, 1.0), R(1.0, 0.0, 0.0)
{

}

Point Camera::o() const
{
    return O;
}

Point Camera::f() const
{
    return F;
}

Point Camera::u() const
{
    return U;
}

Point Camera::r() const
{
    return R;
}

Point Camera::vf() const
{
```

```
    Point res(F);
    res.sub(O);
    return res;
}

Point Camera::vu() const
{
    Point res(U);
    res.sub(O);
    return res;
}

Point Camera::vr() const
{
    Point res(R);
    res.sub(O);
    return res;
}

void Camera::move(double x, double y, double z)
{
    O.setX(O.x() + x);
    O.setY(O.y() + y);
    O.setZ(O.z() + z);

    F.setX(F.x() + x);
    F.setY(F.y() + y);
    F.setZ(F.z() + z);

    U.setX(U.x() + x);
    U.setY(U.y() + y);
    U.setZ(U.z() + z);

    R.setX(R.x() + x);
    R.setY(R.y() + y);
    R.setZ(R.z() + z);
}

void Camera::rotateOX(double a)
{
    rotate(a, 1);
}

void Camera::rotateOY(double a)
{
    rotate(a, 2);
}

void Camera::rotateOZ(double a)
{
    rotate(a, 3);
}

/*mode: 1-OX, 2-OY, 3-OZ*/
void Camera::rotate(double alpha, int mode)
{
    //https://i.imgur.com/WQtfkPW.png

    double o_x_buff, o_y_buff, o_z_buff;

    if(mode == 1)
    {
        o_x_buff = R.x();
```

```
    o_y_buff = R.y();
    o_z_buff = R.z();
}
else if(mode == 2)
{
    o_x_buff = F.x();
    o_y_buff = F.y();
    o_z_buff = F.z();
}
else /*if(mode == 3)*/
{
    o_x_buff = U.x();
    o_y_buff = U.y();
    o_z_buff = U.z();
}

const double o_x = o_x_buff-O.x();
const double o_y = o_y_buff-O.y();
const double o_z = o_z_buff-O.z();

double x, y, z;
double x_, y_, z_;

// F

x_ = F.x()-O.x(); y_ = F.y()-O.y(), z_ = F.z()-O.z();

x = x_ * ( o_x*o_x*c1(alpha) + c(alpha) );
x += y_ * ( o_x*o_y*c1(alpha) - o_z*s(alpha) );
x += z_ * ( o_x*o_z*c1(alpha) + o_y*s(alpha) );

y = x_ * ( o_x*o_y*c1(alpha) + o_z*s(alpha) );
y += y_ * ( o_y*o_y*c1(alpha)  + c(alpha) );
y += z_ * ( o_y*o_z*c1(alpha) - o_x*s(alpha) );

z = x_ * ( o_x*o_z*c1(alpha) - o_y*s(alpha) );
//z += y_ * ( o_x*o_z*c1(alpha)  + o_x*s(alpha) );
z += y_ * ( o_y*o_z*c1(alpha)  + o_x*s(alpha) );
z += z_ * ( o_z*o_z*c1(alpha) + c(alpha) );

F.setX(x+O.x()); F.setY(y+O.y()); F.setZ(z+O.z());

// U

x_ = U.x()-O.x(); y_ = U.y()-O.y(), z_ = U.z()-O.z();

x = x_ * ( o_x*o_x*c1(alpha) + c(alpha) );
x += y_ * ( o_x*o_y*c1(alpha) - o_z*s(alpha) );
x += z_ * ( o_x*o_z*c1(alpha) + o_y*s(alpha) );

y = x_ * ( o_x*o_y*c1(alpha) + o_z*s(alpha) );
y += y_ * ( o_y*o_y*c1(alpha)  + c(alpha) );
y += z_ * ( o_y*o_z*c1(alpha) - o_x*s(alpha) );

z = x_ * ( o_x*o_z*c1(alpha) - o_y*s(alpha) );
//z += y_ * ( o_x*o_z*c1(alpha)  + o_x*s(alpha) );
z += y_ * ( o_y*o_z*c1(alpha)  + o_x*s(alpha) );
z += z_ * ( o_z*o_z*c1(alpha) + c(alpha) );

U.setX(x+O.x()); U.setY(y+O.y()); U.setZ(z+O.z());

// R
```

```
    x_ = R.x()-O.x(); y_ = R.y()-O.y(), z_ = R.z()-O.z();

    x = x_ * ( o_x*o_x*c1(alpha) + c(alpha) );
    x += y_ * ( o_x*o_y*c1(alpha) - o_z*s(alpha) );
    x += z_ * ( o_x*o_z*c1(alpha) + o_y*s(alpha) );

    y = x_ * ( o_x*o_y*c1(alpha) + o_z*s(alpha) );
    y += y_ * ( o_y*o_y*c1(alpha)  + c(alpha) );
    y += z_ * ( o_y*o_z*c1(alpha) - o_x*s(alpha) );

    z = x_ * ( o_x*o_z*c1(alpha) - o_y*s(alpha) );
    //z += y_ * ( o_x*o_z*c1(alpha)  + o_x*s(alpha) );
    z += y_ * ( o_y*o_z*c1(alpha)  + o_x*s(alpha) );
    z += z_ * ( o_z*o_z*c1(alpha) + c(alpha) );

    R.setX(x+O.x()); R.setY(y+O.y()); R.setZ(z+O.z());
}

double Camera::c(double a)
{
        double a_rad = (M_PI*a) / 180.0;
        double res = cos(a_rad);
        return res;
}

double Camera::s(double a)
{
        double a_rad = (M_PI*a) / 180.0;
        double res = sin(a_rad);
        return res;
}

double Camera::c1(double a)
{
        double a_rad = (M_PI*a) / 180.0;
        double res = cos(a_rad);
        return 1.0 - res;
}

std::string Camera::print() const
{
    Point vf(F); vf.sub(O);
    Point vu(U); vu.sub(O);
    Point vr(R); vr.sub(O);
    std::string res = O.print("O: ") + "    " + vf.print("F: ") + "=" +
std::to_string(vf.vector_len()) + "    " + vu.print("U: ") + "=" +
std::to_string(vu.vector_len()) + "    " + vr.print("R: ") + "=" +
std::to_string(vr.vector_len());
    return res;
}

void Camera::moveForward(double s)
{
    Point v = vf();
    v.mul(s);
    move(v.x(), v.y(), v.z());
}

void Camera::moveBack(double s)
{
    moveForward(-s);
}
```

```
void Camera::moveRight(double s)
{
    Point v = vr();
    v.mul(s);
    move(v.x(), v.y(), v.z());
}

void Camera::moveLeft(double s)
{
    moveRight(-s);
}

void Camera::moveUp(double s)
{
    Point v = vu();
    v.mul(s);
    move(v.x(), v.y(), v.z());
}

void Camera::moveDown(double s)
{
    moveUp(-s);
      }
```

Файл Triangle.cpp

```
#include <QWidget>
#include <QPainter>

#include <iostream>

#include "./include/Point.h"
#include "./include/Triangle.h"

bool eq(double a, double b)
{
    double d = a - b;
    d = d<0?-d:d;
    return d < 0.0001?true:false;
}

Triangle::Triangle()
: P1(), P2(), P3(), A(), B(), C(), D() {}

Triangle::Triangle(const Point &p1, const Point &p2, const Point &p3)
: P1(p1), P2(p2), P3(p3)
{
    ABCD(P1, P2, P3, &A, &B, &C, &D);
    //std::cout << A << " " << B << " " << C << " " << D << std::endl;
}

Triangle::Triangle(const Triangle& toCopied)
: P1(toCopied.P1), P2(toCopied.P2), P3(toCopied.P3),
A(toCopied.A), B(toCopied.B), C(toCopied.D), D(toCopied.D)
{}

Point Triangle::p1() const {return P1;}

Point Triangle::p2() const {return P2;}

Point Triangle::p3() const {return P3;}
```

```
double Triangle::a() const {return A;}

double Triangle::b() const {return B;}

double Triangle::c() const {return C;}

double Triangle::d() const {return D;}

Point Triangle::n() const
{
    return Point(A, B, C);
}

void Triangle::setP(const Point &new_p1, const Point &new_p2, const Point
&new_p3)
{
    P1.copy(new_p1);
    P2.copy(new_p2);
    P3.copy(new_p3);
    ABCD(P1, P2, P3, &A, &B, &C, &D);
}

Point Triangle::crossLine(const Point &p_lineBegin, const Point &p_lineEnd)
const
{
    const Point p = Point(p_lineEnd.x() - p_lineBegin.x(),
                          p_lineEnd.y() - p_lineBegin.y(),
                          p_lineEnd.z() - p_lineBegin.z());

    const Point M = Point(p_lineBegin.x(), p_lineBegin.y(), p_lineBegin.z());

    double t = -((D + A*M.x() + B*M.y() + C*M.z()) / (A*p.x() + B*p.y() +
C*p.z()));

    return Point(t*p.x() + M.x(), t*p.y() + M.y(), t*p.z() + M.z());
}

void Triangle::copy(const Triangle& other)
{
    this->P1.copy(other.P1);
    this->P2.copy(other.P2);
    this->P3.copy(other.P3);
    this->A = other.A;
    this->B = other.B;
    this->C = other.C;
    this->D = other.D;

}

void Triangle::ABCD(const Point &p0, const Point &p1, const Point &p2, double
*A, double *B, double *C, double *D)
{
    // https://i.imgur.com/LWliYUK.png

    double A00 = (p1.y() - p0.y()) * (p2.z() - p0.z()) - (p2.y() - p0.y()) *
(p1.z() - p0.z());
    double A10 = (p1.x() - p0.x()) * (p2.z() - p0.z()) - (p2.x() - p0.x()) *
(p1.z() - p0.z());
    A10 = -A10;
    double A20 = (p1.x() - p0.x()) * (p2.y() - p0.y()) - (p2.x() - p0.x()) *
(p1.y() - p0.y());

    // A00*(x-p0.x) + A10*(y-p0.y) + A20*(z-p0.z) = 0
```

```
    *A = A00;
    *B = A10;
    *C = A20;

    *D = -(     A00*p0.x() + A10*p0.y() + A20*p0.z()     );
}

bool Triangle::crossLine2(const Point &p_lineBegin, const Point &p_lineEnd,
Point &res) const
{

    Point O = crossLine(p_lineBegin, p_lineEnd);

    double x_AO =  O.x() - P1.x(), y_AO =  O.y() - P1.y(); // P1 is A
    double x_AB = P2.x() - P1.x(), y_AB = P2.y() - P1.y(); // P2 is B
    double x_AC = P3.x() - P1.x(), y_AC = P3.y() - P1.y(); // P3 is C


    double c = (x_AO*y_AB - y_AO*x_AB) / (x_AC*y_AB - y_AC*x_AB);
    double b = (x_AO - c*x_AC) / (x_AB);

    res.copy(O);

    if(b >= 0 && c >= 0 && b+c <= 1)
        return true;
    else
        return false;
}

bool Triangle::equals(const Triangle& other) const
{
    if(this == &other)
        return true;
    if(
        eq(this->A, other.A)
        && eq(this->B, other.B)
        && eq(this->C, other.C)
        && eq(this->D, other.D)
    )
        return true;
    else
        return false;
}
```

Файл main.cpp

```
#include <QApplication>
#include <QIcon>

#include "./include/mainWindow.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    MainWindow window;

    window.setWindowTitle("lab6");
    window.resize(700, 700);
    window.setWindowIcon(QIcon("./ico.png"));

    window.show();
```

```
    return app.exec();
}
```