|  |  |  |
| --- | --- | --- |
| Министерство образования и науки Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
| path817.png | | |
| Кафедра прикладной математики | | |
|  | | |
| Лабораторная работа № 3 | | |
| по дисциплине «Компьютерная графика» | | |
|  | | |
|  | | |
|  | | |
| эмблема_светлая.png | Факультет: | ПМИ |
| Группа: | ПМИ-62 |
| Бригада: | 4 |
| Студенты: | Мамонова Е. В., Ершов П. К. |
| Преподаватель: | Задорожный А. Г. |
|  | | |
| Новосибирск | | |
| 2019 | | |

1. Цель работы

Ознакомиться с методом тиражирования сечений (основным способом задания полигональных моделей) и средствами трехмерной визуализации (системы координат, источники света, свойства материалов).

1. **Задание**
   1. Считывать из файла (в зависимости от варианта): а) 2D-координаты вершин сечения (считающегося выпуклым); б) 3D-координаты траектории тиражирования; в) параметры изменения сечения.
   2. Построить фигуру в 3D по прочитанным данным.
   3. Включить режимы: а) буфера глубины; б) двойной буферизации; в) освещения и материалов.
   4. Предоставить возможность показа: а) каркаса объекта; б) нормалей (например, отрезками); в) текстур, «обернутых» вокруг фигуры.
   5. Предоставить возможность переключения между режимами ортографической и перспективной проекции.
   6. Обеспечить навигацию по сцене с помощью модельно-видовых преобразований, сохраняя положение источника света.
   7. Предоставить возможность включения / выключения режима сглаживания нормалей.
2. **Описание управления**

Выбор направленеи движения осуществляется с помощью мыши.

Управление клавишами.

W/w - вперёд.  
S/s - назад.  
A/a - поворот влево.  
D/d - поворот вправо.  
P/p - вкл/выкл режим проекции.  
T/t - включение/отключение текстур.  
R/r - переключение между текстурами.  
G/g - вкл/выкл сетку.  
B/b - сгладить нормали.  
L/l - вкл/выкл свет.  
1 - рассеянный свет (от пространства).  
2 - эмиссионный свет (от объекта).  
3 - точечный.  
4 - прожектор рассеянный.  
5 - ещё один прожектор, то уже.  
O/o - вернутся на исходную точку камеры (если улетишь далеко).  
M/m - каркасный режим.  
< - плюс к скорость.  
> - минус к скорость.  
N/n - показывает нормали.  
X/x - выбор режима текстурирования (с учётом цвета фигуры или без).  
- - отдаление от фигуры в режиме проекции.  
+ - приближение к фигуре в режиме проекции.

1. Текст программы

norm.h

#include<math.h>

class Vector3f

{

public:

float x, y, z;

Vector3f() {}; //конструктор по умолчанию

Vector3f(float \_x, float \_y, float \_z) //конструктор

{

x = \_x;

y = \_y;

z = \_z;

}

//Переопределение оператора +

Vector3f operator + (Vector3f \_Vector)

{

return Vector3f(\_Vector.x + x, \_Vector.y + y, \_Vector.z + z);

}

//Переопределение оператора -

Vector3f operator - (Vector3f \_Vector)

{

return Vector3f(x - \_Vector.x, y - \_Vector.y, z - \_Vector.z);

}

//Переопределение оператора \*

Vector3f operator \* (float num)

{

return Vector3f(x \* num, y \* num, z \* num);

}

//Переопределение оператора /

Vector3f operator / (float num)

{

return Vector3f(x / num, y / num, z / num);

}

};

//Вычисление нормали двух векторов

Vector3f Cross(Vector3f \_Vec1, Vector3f \_Vec2);

//Вычисление нормы вектора

float Norm(Vector3f \_Vec);

//Нормализация вектора

Vector3f Normalize(Vector3f \_Vec);

//Вычисление скалярного произведения

float scalar(Vector3f \_Vec1, Vector3f \_Vec2);

//Вектор между двумя точками

Vector3f Vector(Vector3f \_Point1, Vector3f \_Point2);

//Вычисление нормали полигона

Vector3f norm\_pol(Vector3f one, Vector3f two, Vector3f tr);

Camera.h

#pragma once

#include <windows.h>

#include <vector>

#include <gl\gl.h>

#include "norm.h"

class Camera

{

public:

Vector3f Pos; //позиция камеры/положение точки наблюдателя/

Vector3f View; //направление наблюдения

Vector3f rot\_vect; //вектор поворота сцены

Camera(); //конструктор

void pos\_camera(float posX, float posY, float posZ,

float viewX, float viewY, float viewZ,

float upX, float upY, float upZ); //установка позиции камеры

void set\_view\_mouse(GLint width, GLint height); //установка вида с помощью мыши

void move\_c(float speed); //передвижение камеры вперед/назад

void r\_view(float angle, float x, float y, float z); //вращение камеры вокруг заданной оси

void r\_obs(Vector3f vCenter, float angle, float x, float y, float z);//вращение камеры вокруг наблюдателя

};

sours.h

#pragma once

#define \_USE\_MATH\_DEFINES

#include <windows.h>

#include <gl\glut.h>

#include "glaux.h"

#include <vector>

#include <gl\gl.h>

#include <stdio.h>

#include "Camera.h"

using namespace std;

float kSpeed = 0.03f; //скорость перемещения камеры

#define NTEXT 8 //количество текстур

bool Mult = false; //тип наложения текстуры

float angle = 60; //угол в перспективной проекции

float factor = 10; //множитель в ортографической проекции

GLint width, height; //ширина и высота окна

Camera camera1; //камера

bool rot = false; //включить вращение камеры с помощью мыши

bool persp = true; //включить перспективную проекцию

bool wireframe = false; //каркасный режим

bool snormal = false; //отображение нормалей

bool smooth = false; //сгладить нормали

bool grid = true; //отображение сетки

bool g\_bLight = true; //включить освещение

bool textureMod = false;//отображение текстуры

int light\_sours = 1; //текущий источник света

float shift\_m[4][4]; //матрица сдвига

float rot\_m[4][4]; //матрица поворота

float scale\_m[4][4]; //матрица масштаба

unsigned int names\_tex[NTEXT];//имена текстур

int nt = 0; //номер текстуры

//структура для хранения треугольника

struct triangle

{

Vector3f one;

Vector3f two;

Vector3f tr;

};

vector <float> percent; //вектор процентных положений сечений тиражирования

vector <Vector3f> normals; //вектор нормалей/содержит нормаль каждого полигона/

vector <Vector3f> sm\_normals; //вектор сглаженных нормалей/содержит нормаль к каждой вершине

vector <Vector3f> traj; //траектория тиражирования

vector <Vector3f> scales; //масштабирование каждого сечения

vector <triangle> fig; //набор сечений треугольника

triangle temp; //шаблон треугольника

//умножение координаты на указанную матрицу

Vector3f multCoord(float \_matrix[4][4], Vector3f \_vec)

{

float vec[4] = { \_vec.x, \_vec.y, \_vec.z, 1.0f };

float resVec[4];

for (int i = 0; i < 4; i++)

{

double s = 0;

for (int j = 0; j < 4; j++)

s += \_matrix[i][j] \* vec[j];

resVec[i] = s;

}

return Vector3f::Vector3f(resVec[0], resVec[1], resVec[2]);

}

//определить матрицу сдвига

void makeTranslateMatrix(float dx, float dy, float dz)

{

shift\_m[0][0] = 1; shift\_m[0][1] = 0;

shift\_m[0][2] = 0; shift\_m[0][3] = dx;

shift\_m[1][0] = 0; shift\_m[1][1] = 1;

shift\_m[1][2] = 0; shift\_m[1][3] = dy;

shift\_m[2][0] = 0; shift\_m[2][1] = 0;

shift\_m[2][2] = 1; shift\_m[2][3] = dz;

shift\_m[3][0] = 0; shift\_m[3][1] = 0;

shift\_m[3][2] = 0; shift\_m[3][3] = 1;

}

//определить матрицу поворота/относительно произвольной оси/

void makeRotateMatrix(float angle, Vector3f Axis)

{

float c = (float)cos(angle\* M\_PI / 180.0);

float s = (float)sin(angle\* M\_PI / 180.0);

Vector3f Os = Normalize(Axis);

rot\_m[0][0] = c + (1 - c)\*Os.x\*Os.x;

rot\_m[0][1] = (1 - c)\*Os.x\*Os.y - s \* Os.z;

rot\_m[0][2] = (1 - c)\*Os.x\*Os.z + s \* Os.y;

rot\_m[0][3] = 0;

rot\_m[1][0] = (1 - c)\*Os.x\*Os.y + s \* Os.z;

rot\_m[1][1] = c + (1 - c)\*Os.y\*Os.y;

rot\_m[1][2] = (1 - c)\*Os.z\*Os.y - s \* Os.x;

rot\_m[1][3] = 0;

rot\_m[2][0] = (1 - c)\*Os.x\*Os.z - s \* Os.y;

rot\_m[2][1] = (1 - c)\*Os.z\*Os.y + s \* Os.x;

rot\_m[2][2] = c + (1 - c)\*Os.z\*Os.z;

rot\_m[2][3] = 0;

rot\_m[3][0] = 0;

rot\_m[3][1] = 0;

rot\_m[3][2] = 0;

rot\_m[3][3] = 1;

}

//определить матрицу масштаба

void makeScaleMatrix(float sx, float sy, float sz)

{

scale\_m[0][0] = sx; scale\_m[0][1] = 0;

scale\_m[0][2] = 0; scale\_m[0][3] = 0;

scale\_m[1][0] = 0; scale\_m[1][1] = sy;

scale\_m[1][2] = 0; scale\_m[1][3] = 0;

scale\_m[2][0] = 0; scale\_m[2][1] = 0;

scale\_m[2][2] = sz; scale\_m[2][3] = 0;

scale\_m[3][0] = 0; scale\_m[3][1] = 0;

scale\_m[3][2] = 0; scale\_m[3][3] = 1;

}

//трансформация нового треугольника

triangle transform(int num, float angle, Vector3f Axis, Vector3f shift)

{

triangle \_triangle;

\_triangle.one = temp.one;

\_triangle.two = temp.two;

\_triangle.tr = temp.tr;

//если угол не равен нулю - повернуть треугольник

if (angle != 0)

{

makeRotateMatrix(angle, Axis);

\_triangle.one = multCoord(rot\_m, \_triangle.one);

\_triangle.two = multCoord(rot\_m, \_triangle.two);

\_triangle.tr = multCoord(rot\_m, \_triangle.tr);

}

//масштабируем треугольник

makeScaleMatrix(scales[num].x, scales[num].y, scales[num].z);

\_triangle.one = multCoord(scale\_m, \_triangle.one);

\_triangle.two = multCoord(scale\_m, \_triangle.two);

\_triangle.tr = multCoord(scale\_m, \_triangle.tr);

//сдвинуть треугольник

makeTranslateMatrix(shift.x, shift.y, shift.z);

\_triangle.one = multCoord(shift\_m, \_triangle.one);

\_triangle.two = multCoord(shift\_m, \_triangle.two);

\_triangle.tr = multCoord(shift\_m, \_triangle.tr);

return \_triangle;

}

//тиражировать фигуру

void makeDuplication()

{

FILE \*in1;

FILE \*in2;

FILE \*in3;

in1 = fopen("triangle.txt", "r");

in2 = fopen("trajectory.txt", "r");

in3 = fopen("percent.txt", "r");

//считать координаты шаблонного треугольника

fscanf(in1, "%f%f%f", &temp.one.x, &temp.one.y, &temp.one.z);

fscanf(in1, "%f%f%f", &temp.two.x, &temp.two.y, &temp.two.z);

fscanf(in1, "%f%f%f", &temp.tr.x, &temp.tr.y, &temp.tr.z);

Vector3f coord;

Vector3f scale;

float ang;

//считать координаты траектории

while (!feof(in2))

{

fscanf(in2, "%f%f%f", &coord.x, &coord.y, &coord.z);

traj.push\_back(coord);

}

float per;

while (!feof(in3))

{

fscanf(in3, "%f", &per);

fscanf(in3, "%f%f%f", &scale.x, &scale.y, &scale.z);

percent.push\_back(per);

scales.push\_back(scale);

}

fclose(in1);

fclose(in2);

fclose(in3);

Vector3f Axis; //ось поворота

//предыдущее направление - нормаль шаблонного треугольника

Vector3f predPath = norm\_pol(temp.one, temp.two, temp.tr);

Vector3f curPath; //текущее направление

triangle \_triangle;

int i;

float trajLength = 0; //длина пути

//вычисление длины пути

for (i = 0; i < traj.size() - 1; i++)

{

curPath = traj[i + 1] - traj[i];

trajLength += Norm(curPath);

}

for (i = 0; i < percent.size(); i++)

{

//расстояние до точки тиражирования

float pointDistance = (trajLength\*percent[i]) / 100.;

//поиск отрезка траектории, которому принадлежит точка тиражирования

int j;

Vector3f curPathLength; //текущее направление вектора траектории

float curLength = 0; //текущая длина пути

float predLength = 0; //предыдущая длина пути

bool found = false; //найден отрезок траектории

int numPath; //индекс начала отрезка траектории

//если это последнее сечение

if (i == percent.size() - 1)

numPath = traj.size() - 2;

else

{

for (j = 0; j < traj.size() - 1, !found; j++)

{

curPathLength = traj[j + 1] - traj[j];

curLength += Norm(curPathLength);

//если точка тиражирования пренадлежит текущему отрезку

if (pointDistance >= predLength && pointDistance <= curLength)

found = true;

else

predLength = curLength;

}

numPath = j - 1;

}

//вычислить текущее направление

curPath = traj[numPath + 1] - traj[numPath];

//величина сдвига от начала отрезка, которому принадлежит точка тиражирования

float localPath = pointDistance - predLength;

if (i == percent.size() - 1)

localPath = Norm(curPath);

float normCurPath = Norm(curPath);

//вычисление координат сдвига

Vector3f shift = Vector3f(traj[numPath].x + (localPath \* curPath.x) / normCurPath,

traj[numPath].y + (localPath \* curPath.y) / normCurPath,

traj[numPath].z + (localPath \* curPath.z) / normCurPath);

//вычислить скалярное произведение

float scal = scalar(curPath, predPath);

//вычислить ось

Axis = Cross(curPath, predPath);

//вычислить угол поворота

ang = acos(scal / (Norm(curPath) \* Norm(predPath)))\* 180.0 / M\_PI;

//в зависимости от знака скалярного произведения изменить угол

if (scal < 0)

ang = 180 - ang;

else

ang = -(180 + ang);

if (ang == 180) ang = 0;

//определить координаты нового треугольника

\_triangle = transform(i, ang, Axis, shift);

//добавить треугольник в вектор

fig.push\_back(\_triangle);

}

}

//вычисление нормалей к плоскостям

void calcNormals()

{

Vector3f normal;

//отдельно вычисляются нормали к первому треугольнику

normal = norm\_pol(fig[0].one, fig[0].two, fig[0].tr);

normals.push\_back(normal);

for (int i = 0; i < fig.size() - 1; i++)

{

normal = norm\_pol(fig[i].two, fig[i].one, fig[i + 1].one);

normals.push\_back(normal);

normal = norm\_pol(fig[i].one, fig[i].tr, fig[i + 1].tr);

normals.push\_back(normal);

normal = norm\_pol(fig[i].tr, fig[i].two, fig[i + 1].two);

normals.push\_back(normal);

}

//и к последнему

int j = fig.size() - 1;

normal = norm\_pol(fig[j].one, fig[j].tr, fig[j].two);

normals.push\_back(normal);

}

//вычисление сглаженных нормалей

void smoothNormals()

{

Vector3f \_smoothNormal;

\_smoothNormal = (normals[0] + normals[1] + normals[2]) / 3.0;

sm\_normals.push\_back(\_smoothNormal);

\_smoothNormal = (normals[0] + normals[1] + normals[3]) / 3.0;

sm\_normals.push\_back(\_smoothNormal);

\_smoothNormal = (normals[0] + normals[2] + normals[3]) / 3.0;

sm\_normals.push\_back(\_smoothNormal);

int nN = 1;

for (int i = 1; i < fig.size() - 1; i++)

{

\_smoothNormal = (normals[nN] + normals[nN + 1] + normals[nN + 3] + normals[nN + 4]) / 4.0;

sm\_normals.push\_back(\_smoothNormal);

\_smoothNormal = (normals[nN] + normals[nN + 2] + normals[nN + 3] + normals[nN + 5]) / 4.0;

sm\_normals.push\_back(\_smoothNormal);

\_smoothNormal = (normals[nN + 1] + normals[nN + 2] + normals[nN + 4] + normals[nN + 5]) / 4.0;

sm\_normals.push\_back(\_smoothNormal);

nN += 3;

}

int j = normals.size() - 1;

\_smoothNormal = (normals[j] + normals[j - 2] + normals[j - 3]) / 3.0;

sm\_normals.push\_back(\_smoothNormal);

\_smoothNormal = (normals[j] + normals[j - 1] + normals[j - 3]) / 3.0;

sm\_normals.push\_back(\_smoothNormal);

\_smoothNormal = (normals[j] + normals[j - 1] + normals[j - 2]) / 3.0;

sm\_normals.push\_back(\_smoothNormal);

}

//\*/

void loadTexture()//загрузка текстуры

{

FILE \*F;

char str[20] = "1.bmp";

char num = '1';

auto widthT = width;

auto heightT = height;

glGenTextures(NTEXT, names\_tex); //создание имен текстур

for (int i = 0; i < NTEXT; i++)

{

str[0] = num;

num++;

if ((F = fopen(str, "rb")) != NULL)

{

/\*Перемещаемся в bmp-файле на нужную позицию, и считываем ширину и длинну \*/

fseek(F, 18, SEEK\_SET);

fread(&(widthT), 2, 1, F);

fseek(F, 2, SEEK\_CUR);

fread(&(heightT), 2, 1, F);

/\* Выделяем память под изображение. Если память не выделилась, закрываем файл и выходим с ошибкой \*/

auto pixels = new GLubyte[3 \* widthT \* heightT];

/\* Считываем изображение в память по 3 бита, то бишь RGB для каждого пикселя \*/

fseek(F, 30, SEEK\_CUR);

fread(pixels, 3, widthT \* heightT, F);

glBindTexture(GL\_TEXTURE\_2D, names\_tex[i]); //выбрать текущую текстуру

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

gluBuild2DMipmaps(GL\_TEXTURE\_2D, 3, widthT, heightT, GL\_BGR\_EXT, GL\_UNSIGNED\_BYTE, pixels);

free(pixels);

fclose(F);

}

}

}

void Reshape(GLint w, GLint h)

{

//изменить размеры окна

width = w;

height = h;

//вычислить соотношение между шириной и высотой

//предотвратить деление на 0

if (height == 0)

height = 1;

float ratio = 1. \* width / height;

//установить матрицу проекции/определяет объем сцены

glMatrixMode(GL\_PROJECTION);

//загрузить единичную матрицу

glLoadIdentity();

//определить окно просмотра

glViewport(0, 0, width, height);

//если стоит режим перспективы

if (persp)

gluPerspective(60, ratio, 0.1f, 10000.0f); //использовать перспективную проекциию

//иначе - ортографическую проекцию

else

if (width >= height)

glOrtho(-factor \* ratio, factor\*ratio, -factor, factor, -10000.0f, 10000.0f);

else

glOrtho(-factor, factor, -factor / ratio, factor / ratio, -10000.0f, 10000.0f);

//возврат к матрице модели

glMatrixMode(GL\_MODELVIEW);

}

void Draw3DSGrid()

{

glColor3ub(0, 0, 255);;

for (float i = -50; i <= 50; i += 1)

{

glBegin(GL\_LINES);

// Ось Х

glVertex3f(-50, 0, i);

glVertex3f(50, 0, i);

// Ось Z

glVertex3f(i, 0, -50);

glVertex3f(i, 0, 50);

glEnd();

}

}

//отрисовка тиражированной фигуры

void drawing\_f()

{

Vector3f normal;

int nN = 0;

//НЕ КАРКАСНЫЙ РЕЖИМ

if (!wireframe)

{

//НОРМАЛИ НЕ СГЛАЖЕНЫ

if (!smooth)

{

//ТЕКСТУРИРОВАНИЕ ОТКЛЮЧЕНО

if (!textureMod)

{

//устанавливается нормаль для каждой плоскости

glBegin(GL\_TRIANGLES);

glNormal3f(normals[nN].x, normals[nN].y, normals[nN].z); nN++;

glColor3ub(255, 0, 0);;

glVertex3f(fig[0].one.x, fig[0].one.y, fig[0].one.z);

glVertex3f(fig[0].two.x, fig[0].two.y, fig[0].two.z);

glVertex3f(fig[0].tr.x, fig[0].tr.y, fig[0].tr.z);

glEnd();

for (int i = 0; i < fig.size() - 1; i++)

{

glBegin(GL\_POLYGON);

glNormal3f(normals[nN].x, normals[nN].y, normals[nN].z); nN++;

glColor3ub(255, 0, 0);;

glVertex3f(fig[i].one.x, fig[i].one.y, fig[i].one.z);

glVertex3f(fig[i + 1].one.x, fig[i + 1].one.y, fig[i + 1].one.z);

glVertex3f(fig[i + 1].two.x, fig[i + 1].two.y, fig[i + 1].two.z);

glVertex3f(fig[i].two.x, fig[i].two.y, fig[i].two.z);

glEnd();

glBegin(GL\_POLYGON);

glNormal3f(normals[nN].x, normals[nN].y, normals[nN].z); nN++;

glColor3ub(255, 0, 0);;

glVertex3f(fig[i].tr.x, fig[i].tr.y, fig[i].tr.z);

glVertex3f(fig[i + 1].tr.x, fig[i + 1].tr.y, fig[i + 1].tr.z);

glVertex3f(fig[i + 1].one.x, fig[i + 1].one.y, fig[i + 1].one.z);

glVertex3f(fig[i].one.x, fig[i].one.y, fig[i].one.z);

glEnd();

glBegin(GL\_POLYGON);

glNormal3f(normals[nN].x, normals[nN].y, normals[nN].z); nN++;

glColor3ub(255, 0, 0);;

glVertex3f(fig[i].two.x, fig[i].two.y, fig[i].two.z);

glVertex3f(fig[i + 1].two.x, fig[i + 1].two.y, fig[i + 1].two.z);

glVertex3f(fig[i + 1].tr.x, fig[i + 1].tr.y, fig[i + 1].tr.z);

glVertex3f(fig[i].tr.x, fig[i].tr.y, fig[i].tr.z);

glEnd();

}

int j = fig.size() - 1;

glBegin(GL\_TRIANGLES);

glNormal3f(normals[nN].x, normals[nN].y, normals[nN].z); nN++;

glColor3ub(255, 0, 0);;

glVertex3f(fig[j].one.x, fig[j].one.y, fig[j].one.z);

glVertex3f(fig[j].two.x, fig[j].two.y, fig[j].two.z);

glVertex3f(fig[j].tr.x, fig[j].tr.y, fig[j].tr.z);

glEnd();

}

//ТЕКСТУРИРОВАНИЕ ВКЛЮЧЕНО

else

{

glEnable(GL\_TEXTURE\_2D); //разрешить текстурирование

glBindTexture(GL\_TEXTURE\_2D, names\_tex[nt]); //выбрать текстуру

if (!Mult)

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL); //умножение

else

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

//устанавливается нормаль для каждой плоскости

//и тестурные координаты

glBegin(GL\_TRIANGLES);

glNormal3f(normals[nN].x, normals[nN].y, normals[nN].z); nN++;

glColor3ub(255, 0, 0);;

glTexCoord2f(0, 0); glVertex3f(fig[0].one.x, fig[0].one.y, fig[0].one.z);

glTexCoord2f(0.5, 1); glVertex3f(fig[0].two.x, fig[0].two.y, fig[0].two.z);

glTexCoord2f(1, 0); glVertex3f(fig[0].tr.x, fig[0].tr.y, fig[0].tr.z);

glEnd();

for (int i = 0; i < fig.size() - 1; i++)

{

glBegin(GL\_POLYGON);

glNormal3f(normals[nN].x, normals[nN].y, normals[nN].z); nN++;

glColor3ub(255, 0, 0);;

//glTexCoord2f(0, 1);

glTexCoord2f(percent[i] / 100., 0.5);

glVertex3f(fig[i].one.x, fig[i].one.y, fig[i].one.z);

//glTexCoord2f(1, 1);

glTexCoord2f(percent[i + 1] / 100., 0.5);

glVertex3f(fig[i + 1].one.x, fig[i + 1].one.y, fig[i + 1].one.z);

//glTexCoord2f(1, 0);

glTexCoord2f(percent[i + 1] / 100., 0.25);

glVertex3f(fig[i + 1].two.x, fig[i + 1].two.y, fig[i + 1].two.z);

//glTexCoord2f(0, 0);

glTexCoord2f(percent[i] / 100., 0.25);

glVertex3f(fig[i].two.x, fig[i].two.y, fig[i].two.z);

glEnd();

glBegin(GL\_POLYGON);

glNormal3f(normals[nN].x, normals[nN].y, normals[nN].z); nN++;

glColor3ub(255, 0, 0);;

//glTexCoord2f(0, 1);

glTexCoord2f(percent[i] / 100., 0.75);

glVertex3f(fig[i].tr.x, fig[i].tr.y, fig[i].tr.z);

//glTexCoord2f(1, 1);

glTexCoord2f(percent[i + 1] / 100., 0.75);

glVertex3f(fig[i + 1].tr.x, fig[i + 1].tr.y, fig[i + 1].tr.z);

//glTexCoord2f(1, 0);

glTexCoord2f(percent[i + 1] / 100., 0.25);

glVertex3f(fig[i + 1].one.x, fig[i + 1].one.y, fig[i + 1].one.z);

//glTexCoord2f(0, 0);

glTexCoord2f(percent[i] / 100., 0.25);

glVertex3f(fig[i].one.x, fig[i].one.y, fig[i].one.z);

glEnd();

glBegin(GL\_POLYGON);

glNormal3f(normals[nN].x, normals[nN].y, normals[nN].z); nN++;

glColor3ub(255, 0, 0);;

//glTexCoord2f(0, 1);

glTexCoord2f(percent[i] / 100., 0.5);

glVertex3f(fig[i].two.x, fig[i].two.y, fig[i].two.z);

//glTexCoord2f(1, 1);

glTexCoord2f(percent[i + 1] / 100., 0.5);

glVertex3f(fig[i + 1].two.x, fig[i + 1].two.y, fig[i + 1].two.z);

//glTexCoord2f(1, 0);

glTexCoord2f(percent[i + 1] / 100., 0.25);

glVertex3f(fig[i + 1].tr.x, fig[i + 1].tr.y, fig[i + 1].tr.z);

//glTexCoord2f(0, 0);

glTexCoord2f(percent[i] / 100., 0.25);

glVertex3f(fig[i].tr.x, fig[i].tr.y, fig[i].tr.z);

glEnd();

}

int j = fig.size() - 1;

glBegin(GL\_TRIANGLES);

glNormal3f(normals[nN].x, normals[nN].y, normals[nN].z); nN++;

glColor3ub(255, 0, 0);;

glTexCoord2f(0, 0); glVertex3f(fig[j].one.x, fig[j].one.y, fig[j].one.z);

glTexCoord2f(0.5, 1); glVertex3f(fig[j].two.x, fig[j].two.y, fig[j].two.z);

glTexCoord2f(1, 0); glVertex3f(fig[j].tr.x, fig[j].tr.y, fig[j].tr.z);

glEnd();

//отключить текстурирование

glDisable(GL\_TEXTURE\_2D);

}

}

//НОРМАЛИ СГЛАЖЕНЫ

else

{

//ТЕКСТУРИРОВАНИЕ ОТКЛЮЧЕНО

if (!textureMod)

{

//устанавливается нормаль для каждой вершины

glBegin(GL\_TRIANGLES);

glColor3ub(255, 0, 0);;

glNormal3f(sm\_normals[0].x, sm\_normals[0].y, sm\_normals[0].z);

glVertex3f(fig[0].one.x, fig[0].one.y, fig[0].one.z);

glNormal3f(sm\_normals[1].x, sm\_normals[1].y, sm\_normals[1].z);

glVertex3f(fig[0].two.x, fig[0].two.y, fig[0].two.z);

glNormal3f(sm\_normals[2].x, sm\_normals[2].y, sm\_normals[2].z);

glVertex3f(fig[0].tr.x, fig[0].tr.y, fig[0].tr.z);

glEnd();

nN = 0;

for (int i = 0; i < fig.size() - 1; i++)

{

glBegin(GL\_POLYGON);

glColor3ub(255, 0, 0);;

glNormal3f(sm\_normals[nN].x, sm\_normals[nN].y, sm\_normals[nN].z);

glVertex3f(fig[i].one.x, fig[i].one.y, fig[i].one.z);

glNormal3f(sm\_normals[nN + 3].x, sm\_normals[nN + 3].y, sm\_normals[nN + 3].z);

glVertex3f(fig[i + 1].one.x, fig[i + 1].one.y, fig[i + 1].one.z);

glNormal3f(sm\_normals[nN + 4].x, sm\_normals[nN + 4].y, sm\_normals[nN + 4].z);

glVertex3f(fig[i + 1].two.x, fig[i + 1].two.y, fig[i + 1].two.z);

glNormal3f(sm\_normals[nN + 1].x, sm\_normals[nN + 1].y, sm\_normals[nN + 1].z);

glVertex3f(fig[i].two.x, fig[i].two.y, fig[i].two.z);

glEnd();

glBegin(GL\_POLYGON);

glColor3ub(255, 0, 0);;

glNormal3f(sm\_normals[nN + 2].x, sm\_normals[nN + 2].y, sm\_normals[nN + 2].z);

glVertex3f(fig[i].tr.x, fig[i].tr.y, fig[i].tr.z);

glNormal3f(sm\_normals[nN + 5].x, sm\_normals[nN + 5].y, sm\_normals[nN + 5].z);

glVertex3f(fig[i + 1].tr.x, fig[i + 1].tr.y, fig[i + 1].tr.z);

glNormal3f(sm\_normals[nN + 3].x, sm\_normals[nN + 3].y, sm\_normals[nN + 3].z);

glVertex3f(fig[i + 1].one.x, fig[i + 1].one.y, fig[i + 1].one.z);

glNormal3f(sm\_normals[nN].x, sm\_normals[nN].y, sm\_normals[nN].z);

glVertex3f(fig[i].one.x, fig[i].one.y, fig[i].one.z);

glEnd();

glBegin(GL\_POLYGON);

glColor3ub(255, 0, 0);;

glNormal3f(sm\_normals[nN + 1].x, sm\_normals[nN + 1].y, sm\_normals[nN + 1].z);

glVertex3f(fig[i].two.x, fig[i].two.y, fig[i].two.z);

glNormal3f(sm\_normals[nN + 4].x, sm\_normals[nN + 4].y, sm\_normals[nN + 4].z);

glVertex3f(fig[i + 1].two.x, fig[i + 1].two.y, fig[i + 1].two.z);

glNormal3f(sm\_normals[nN + 5].x, sm\_normals[nN + 5].y, sm\_normals[nN + 5].z);

glVertex3f(fig[i + 1].tr.x, fig[i + 1].tr.y, fig[i + 1].tr.z);

glNormal3f(sm\_normals[nN + 2].x, sm\_normals[nN + 2].y, sm\_normals[nN + 2].z);

glVertex3f(fig[i].tr.x, fig[i].tr.y, fig[i].tr.z);

glEnd();

nN += 3;

}

int j = fig.size() - 1;

glBegin(GL\_TRIANGLES);

glColor3ub(255, 0, 0);;

glNormal3f(sm\_normals[nN].x, sm\_normals[nN].y, sm\_normals[nN].z);

glVertex3f(fig[j].one.x, fig[j].one.y, fig[j].one.z);

glNormal3f(sm\_normals[nN + 1].x, sm\_normals[nN + 1].y, sm\_normals[nN + 1].z);

glVertex3f(fig[j].two.x, fig[j].two.y, fig[j].two.z);

glNormal3f(sm\_normals[nN + 2].x, sm\_normals[nN + 2].y, sm\_normals[nN + 2].z);

glVertex3f(fig[j].tr.x, fig[j].tr.y, fig[j].tr.z);

glEnd();

}

//ТЕКСТУРИРОВАНИЕ ВКЛЮЧЕНО

else

{

glEnable(GL\_TEXTURE\_2D); //разрешить текстурирование

glBindTexture(GL\_TEXTURE\_2D, names\_tex[nt]); //выбрать текстуру

if (!Mult)

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL); //умножение

else

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

//устанавливается нормаль для каждой вершины

//и текстурные коорднаты

glBegin(GL\_TRIANGLES);

glColor3ub(255, 0, 0);;

glNormal3f(sm\_normals[0].x, sm\_normals[0].y, sm\_normals[0].z);

glTexCoord2f(0, 0); glVertex3f(fig[0].one.x, fig[0].one.y, fig[0].one.z);

glNormal3f(sm\_normals[1].x, sm\_normals[1].y, sm\_normals[1].z);

glTexCoord2f(0.5, 1); glVertex3f(fig[0].two.x, fig[0].two.y, fig[0].two.z);

glNormal3f(sm\_normals[2].x, sm\_normals[2].y, sm\_normals[2].z);

glTexCoord2f(1, 0); glVertex3f(fig[0].tr.x, fig[0].tr.y, fig[0].tr.z);

glEnd();

nN = 0;

for (int i = 0; i < fig.size() - 1; i++)

{

glBegin(GL\_POLYGON);

glColor3ub(255, 0, 0);;

glNormal3f(sm\_normals[nN].x, sm\_normals[nN].y, sm\_normals[nN].z);

glTexCoord2f(0, 1); glVertex3f(fig[i].one.x, fig[i].one.y, fig[i].one.z);

glNormal3f(sm\_normals[nN + 3].x, sm\_normals[nN + 3].y, sm\_normals[nN + 3].z);

glTexCoord2f(1, 1); glVertex3f(fig[i + 1].one.x, fig[i + 1].one.y, fig[i + 1].one.z);

glNormal3f(sm\_normals[nN + 4].x, sm\_normals[nN + 4].y, sm\_normals[nN + 4].z);

glTexCoord2f(1, 0); glVertex3f(fig[i + 1].two.x, fig[i + 1].two.y, fig[i + 1].two.z);

glNormal3f(sm\_normals[nN + 1].x, sm\_normals[nN + 1].y, sm\_normals[nN + 1].z);

glTexCoord2f(0, 0); glVertex3f(fig[i].two.x, fig[i].two.y, fig[i].two.z);

glEnd();

glBegin(GL\_POLYGON);

glColor3ub(255, 0, 0);;

glNormal3f(sm\_normals[nN + 2].x, sm\_normals[nN + 2].y, sm\_normals[nN + 2].z);

glTexCoord2f(0, 1); glVertex3f(fig[i].tr.x, fig[i].tr.y, fig[i].tr.z);

glNormal3f(sm\_normals[nN + 5].x, sm\_normals[nN + 5].y, sm\_normals[nN + 5].z);

glTexCoord2f(1, 1); glVertex3f(fig[i + 1].tr.x, fig[i + 1].tr.y, fig[i + 1].tr.z);

glNormal3f(sm\_normals[nN + 3].x, sm\_normals[nN + 3].y, sm\_normals[nN + 3].z);

glTexCoord2f(1, 0); glVertex3f(fig[i + 1].one.x, fig[i + 1].one.y, fig[i + 1].one.z);

glNormal3f(sm\_normals[nN].x, sm\_normals[nN].y, sm\_normals[nN].z);

glTexCoord2f(0, 0); glVertex3f(fig[i].one.x, fig[i].one.y, fig[i].one.z);

glEnd();

glBegin(GL\_POLYGON);

glColor3ub(255, 0, 0);;

glNormal3f(sm\_normals[nN + 1].x, sm\_normals[nN + 1].y, sm\_normals[nN + 1].z);

glTexCoord2f(0, 1); glVertex3f(fig[i].two.x, fig[i].two.y, fig[i].two.z);

glNormal3f(sm\_normals[nN + 4].x, sm\_normals[nN + 4].y, sm\_normals[nN + 4].z);

glTexCoord2f(1, 1); glVertex3f(fig[i + 1].two.x, fig[i + 1].two.y, fig[i + 1].two.z);

glNormal3f(sm\_normals[nN + 5].x, sm\_normals[nN + 5].y, sm\_normals[nN + 5].z);

glTexCoord2f(1, 0); glVertex3f(fig[i + 1].tr.x, fig[i + 1].tr.y, fig[i + 1].tr.z);

glNormal3f(sm\_normals[nN + 2].x, sm\_normals[nN + 2].y, sm\_normals[nN + 2].z);

glTexCoord2f(0, 0); glVertex3f(fig[i].tr.x, fig[i].tr.y, fig[i].tr.z);

glEnd();

}

int j = fig.size() - 1;

glBegin(GL\_TRIANGLES);

glColor3ub(255, 0, 0);;

glNormal3f(sm\_normals[nN].x, sm\_normals[nN].y, sm\_normals[nN].z);

glTexCoord2f(0, 0); glVertex3f(fig[j].one.x, fig[j].one.y, fig[j].one.z);

glNormal3f(sm\_normals[nN + 1].x, sm\_normals[nN + 1].y, sm\_normals[nN + 1].z);

glTexCoord2f(0.5, 1); glVertex3f(fig[j].two.x, fig[j].two.y, fig[j].two.z);

glNormal3f(sm\_normals[nN + 2].x, sm\_normals[nN + 2].y, sm\_normals[nN + 2].z);

glTexCoord2f(1, 0); glVertex3f(fig[j].tr.x, fig[j].tr.y, fig[j].tr.z);

glEnd();

//отключить текстурирование

glDisable(GL\_TEXTURE\_2D);

}

}

}

//КАРКАСНЫЙ РЕЖИМ

else

{

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 0);

for (int i = 0; i < fig.size(); i++)

glVertex3f(fig[i].one.x, fig[i].one.y, fig[i].one.z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 0);

for (int i = 0; i < fig.size(); i++)

glVertex3f(fig[i].two.x, fig[i].two.y, fig[i].two.z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 0);

for (int i = 0; i < fig.size(); i++)

glVertex3f(fig[i].tr.x, fig[i].tr.y, fig[i].tr.z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 0);

for (int i = 0; i < traj.size(); i++)

glVertex3f(traj[i].x, traj[i].y, traj[i].z);

glEnd();

for (int i = 0; i < fig.size(); i++)

{

glBegin(GL\_LINE\_LOOP);

glColor3ub(0, 255, 0);

glVertex3f(fig[i].one.x, fig[i].one.y, fig[i].one.z);

glVertex3f(fig[i].two.x, fig[i].two.y, fig[i].two.z);

glVertex3f(fig[i].tr.x, fig[i].tr.y, fig[i].tr.z);

glEnd();

}

}

//ПОКАЗ НОРМАЛЕЙ

if (snormal)

{

//НОРМАЛИ НЕ СГЛАЖЕНЫ

if (!smooth)

{

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[0].one.x, fig[0].one.y, fig[0].one.z);

glVertex3f(fig[0].one.x + normals[0].x, fig[0].one.y + normals[0].y, fig[0].one.z + normals[0].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[0].two.x, fig[0].two.y, fig[0].two.z);

glVertex3f(fig[0].two.x + normals[0].x, fig[0].two.y + normals[0].y, fig[0].two.z + normals[0].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[0].tr.x, fig[0].tr.y, fig[0].tr.z);

glVertex3f(fig[0].tr.x + normals[0].x, fig[0].tr.y + normals[0].y, fig[0].tr.z + normals[0].z);

glEnd();

nN = 1;

for (int i = 0; i < fig.size() - 1; i++)

{

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i].one.x, fig[i].one.y, fig[i].one.z);

glVertex3f(fig[i].one.x + normals[nN].x, fig[i].one.y + normals[nN].y, fig[i].one.z + normals[nN].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i + 1].one.x, fig[i + 1].one.y, fig[i + 1].one.z);

glVertex3f(fig[i + 1].one.x + normals[nN].x, fig[i + 1].one.y + normals[nN].y, fig[i + 1].one.z + normals[nN].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i + 1].two.x, fig[i + 1].two.y, fig[i + 1].two.z);

glVertex3f(fig[i + 1].two.x + normals[nN].x, fig[i + 1].two.y + normals[nN].y, fig[i + 1].two.z + normals[nN].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i].two.x, fig[i].two.y, fig[i].two.z);

glVertex3f(fig[i].two.x + normals[nN].x, fig[i].two.y + normals[nN].y, fig[i].two.z + normals[nN].z);

glEnd();

nN++;

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i].tr.x, fig[i].tr.y, fig[i].tr.z);

glVertex3f(fig[i].tr.x + normals[nN].x, fig[i].tr.y + normals[nN].y, fig[i].tr.z + normals[nN].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i + 1].tr.x, fig[i + 1].tr.y, fig[i + 1].tr.z);

glVertex3f(fig[i + 1].tr.x + normals[nN].x, fig[i + 1].tr.y + normals[nN].y, fig[i + 1].tr.z + normals[nN].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i + 1].one.x, fig[i + 1].one.y, fig[i + 1].one.z);

glVertex3f(fig[i + 1].one.x + normals[nN].x, fig[i + 1].one.y + normals[nN].y, fig[i + 1].one.z + normals[nN].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i].one.x, fig[i].one.y, fig[i].one.z);

glVertex3f(fig[i].one.x + normals[nN].x, fig[i].one.y + normals[nN].y, fig[i].one.z + normals[nN].z);

glEnd();

nN++;

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i].two.x, fig[i].two.y, fig[i].two.z);

glVertex3f(fig[i].two.x + normals[nN].x, fig[i].two.y + normals[nN].y, fig[i].two.z + normals[nN].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i + 1].two.x, fig[i + 1].two.y, fig[i + 1].two.z);

glVertex3f(fig[i + 1].two.x + normals[nN].x, fig[i + 1].two.y + normals[nN].y, fig[i + 1].two.z + normals[nN].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i + 1].tr.x, fig[i + 1].tr.y, fig[i + 1].tr.z);

glVertex3f(fig[i + 1].tr.x + normals[nN].x, fig[i + 1].tr.y + normals[nN].y, fig[i + 1].tr.z + normals[nN].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i].tr.x, fig[i].tr.y, fig[i].tr.z);

glVertex3f(fig[i].tr.x + normals[nN].x, fig[i].tr.y + normals[nN].y, fig[i].tr.z + normals[nN].z);

glEnd();

nN++;

}

int j = fig.size() - 1;

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[j].one.x, fig[j].one.y, fig[j].one.z);

glVertex3f(fig[j].one.x + normals[nN].x, fig[j].one.y + normals[nN].y, fig[j].one.z + normals[nN].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[j].two.x, fig[j].two.y, fig[j].two.z);

glVertex3f(fig[j].two.x + normals[nN].x, fig[j].two.y + normals[nN].y, fig[j].two.z + normals[nN].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[j].tr.x, fig[j].tr.y, fig[j].tr.z);

glVertex3f(fig[j].tr.x + normals[nN].x, fig[j].tr.y + normals[nN].y, fig[j].tr.z + normals[nN].z);

glEnd();

}

//НОРМАЛИ СГЛАЖЕНЫ

else

{

nN = 0;

for (int i = 0; i < fig.size(); i++)

{

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i].one.x, fig[i].one.y, fig[i].one.z);

glVertex3f(fig[i].one.x + sm\_normals[nN].x, fig[i].one.y + sm\_normals[nN].y, fig[i].one.z + sm\_normals[nN].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i].two.x, fig[i].two.y, fig[i].two.z);

glVertex3f(fig[i].two.x + sm\_normals[nN + 1].x, fig[i].two.y + sm\_normals[nN + 1].y, fig[i].two.z + sm\_normals[nN + 1].z);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 255, 255);

glVertex3f(fig[i].tr.x, fig[i].tr.y, fig[i].tr.z);

glVertex3f(fig[i].tr.x + sm\_normals[nN + 2].x, fig[i].tr.y + sm\_normals[nN + 2].y, fig[i].tr.z + sm\_normals[nN + 2].z);

glEnd();

nN += 3;

}

}

}

}

//установка освещения

void Shine\_a\_Light()

{

GLfloat ambience[4] = { 0.1f, 0.1f, 0.1f, 1.0 }; //Цвет мирового света

GLfloat material\_diffuse[] = { 0.19, 0.19, 0.19, 1 };

glLightModelf(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_FALSE);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, material\_diffuse);

glEnable(GL\_COLOR\_MATERIAL);

if (light\_sours == 1)

{

//направленный источник света

//находится в бесконечности и свет от него распространяется в заданном направлении

GLfloat amb[4] = { 1, 1, 1, 1 };

GLfloat light0\_direction[] = { 0, 0, 0, 0 };

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, amb);

glLightfv(GL\_LIGHT0, GL\_POSITION, light0\_direction);

}

if (light\_sours == 2)

{

//точечный источник света

//убывание интенсивности с расстоянием

//отключено (по умолчанию)

GLfloat light1\_diffuse[] = { 0.4, 0.7, 0.2 };

GLfloat light1\_position[] = { 0, 0, 0, 0 };

glEnable(GL\_LIGHT1);

glLightfv(GL\_LIGHT1, GL\_AMBIENT, ambience);

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, light1\_diffuse);

glLightfv(GL\_LIGHT1, GL\_POSITION, light1\_position);

}

if (light\_sours == 3)

{

//точечный источник света

//убывание интенсивности с расстоянием

GLfloat ambi[4] = { 0, 0, 0, 0 };

GLfloat light2\_diffuse[] = { 1.6, 1.5, 0.2 };

GLfloat light2\_position[] = { -5.0, 8.0, -4.0, 1.0 };

glEnable(GL\_LIGHT2);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, ambi);

glLightfv(GL\_LIGHT2, GL\_AMBIENT, ambi);

glLightfv(GL\_LIGHT2, GL\_DIFFUSE, light2\_diffuse);

glLightfv(GL\_LIGHT2, GL\_POSITION, light2\_position);

glLightf(GL\_LIGHT2, GL\_CONSTANT\_ATTENUATION, 10);

glLightf(GL\_LIGHT2, GL\_LINEAR\_ATTENUATION, 0.04);

glLightf(GL\_LIGHT2, GL\_QUADRATIC\_ATTENUATION, 0.04);

}

if (light\_sours == 4)

{

//прожектор

//убывание интенсивности с расстоянием

//отключено (по умолчанию)

GLfloat ambi[4] = { -0.5, 0, 0, 0 };

GLfloat light3\_diffuse[] = { 0.8, 0.4, 0.2 };

GLfloat light3\_position[] = { -5.0, 8.0, -3.0, 1.0 };

GLfloat light3\_spot\_direction[] = { 1.0, -1.0, -1.0 };

glEnable(GL\_LIGHT3);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, ambi);

glLightfv(GL\_LIGHT3, GL\_DIFFUSE, light3\_diffuse);

glLightfv(GL\_LIGHT3, GL\_POSITION, light3\_position);

glLightf(GL\_LIGHT3, GL\_SPOT\_CUTOFF, 50);

glLightfv(GL\_LIGHT3, GL\_SPOT\_DIRECTION, light3\_spot\_direction);

}

if (light\_sours == 5)

{

//прожектор

//включен рассчет убывания интенсивности для прожектора

GLfloat ambi[4] = { -0.5, 0, 0, 0 };

GLfloat light4\_diffuse[] = { 0.9, 0.7, 0.2 };

GLfloat light4\_position[] = { -5.0, 8.0, -3.0, 1.0 };

GLfloat light4\_spot\_direction[] = { 1.0, -1.0, -1.0 };

glEnable(GL\_LIGHT4);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, ambi);

glLightfv(GL\_LIGHT4, GL\_DIFFUSE, light4\_diffuse);

glLightfv(GL\_LIGHT4, GL\_POSITION, light4\_position);

glLightf(GL\_LIGHT4, GL\_SPOT\_CUTOFF, 50);

glLightfv(GL\_LIGHT4, GL\_SPOT\_DIRECTION, light4\_spot\_direction);

glLightf(GL\_LIGHT4, GL\_SPOT\_EXPONENT, 20.0);

}

}

void shp()

{

GLUquadricObj \*quadObj1;

quadObj1 = gluNewQuadric();

glPushMatrix();

glTranslated(-5.0, 8.0, -3.0);

glColor4ub(0, 255, 0, 0);

gluQuadricDrawStyle(quadObj1, GLU\_FILL);

gluSphere(quadObj1, 0.5, 50, 50);

gluDeleteQuadric(quadObj1);

glPopMatrix();

}

void Display(void)

{

//очистка буфера цвета и глубины

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

//включить буфер глубины

glEnable(GL\_DEPTH\_TEST);

if (g\_bLight)

glEnable(GL\_LIGHTING); //включить свет

else

glDisable(GL\_LIGHTING); //выключить свет

//обнуление трансформации

glLoadIdentity();

//установить вид камеры

gluLookAt(camera1.Pos.x, camera1.Pos.y, camera1.Pos.z,

camera1.View.x, camera1.View.y, camera1.View.z,

camera1.rot\_vect.x, camera1.rot\_vect.y, camera1.rot\_vect.z);

//устаносить освещение

Shine\_a\_Light();

//нарисовать сетку

if (grid)

Draw3DSGrid();

//нарисовать тиражированную фигуру

drawing\_f();

shp();

//переместиться в точку взгляда

glTranslatef(camera1.View.x, 0, camera1.View.z);

//отключить все источники

glDisable(GL\_LIGHT0);

glDisable(GL\_LIGHT1);

glDisable(GL\_LIGHT2);

glDisable(GL\_LIGHT3);

glDisable(GL\_LIGHT4);

//двойная буферизация

glutSwapBuffers();

}

void Process\_Mouse\_Move(int x, int y)

{

camera1.set\_view\_mouse(width, height);

}

void Process\_Normal\_Keys(unsigned char key, int x, int y)

{

//приближение/отдаление в ортографической проекции

if (key == '+')

{

if (factor > 0.1 && !persp)

{

factor -= 0.05;

Reshape(width, height);

}

}

if (key == '-' && !persp)

{

if (factor < 10)

{

factor += 0.05;

Reshape(width, height);

}

}

//включение/выключение вращения камеры мышью

if (key == 'q' || key == 'Q')

{

rot = !rot;

ShowCursor(!rot);

}

//переключение между режимами перспективной и ортографической проекции

if (key == 'p' || key == 'P')

{

persp = !persp;

Reshape(width, height);

}

//включение/выключение каркасного режима

if (key == 'm' || key == 'M')

{

wireframe = !wireframe;

if (wireframe)

g\_bLight = false;

else

g\_bLight = true;

}

//включение/выключение отображения нормалей

if (key == 'n' || key == 'N')

{

snormal = !snormal;

}

if (key == 'r' || key == 'R')

{

if (nt < NTEXT)

nt++;

else

nt = 0;

}

//включение/выключение сглаженных нормалей

if (key == 'b' || key == 'B')

{

smooth = !smooth;

}

//включение/выключение показа сетки

if (key == 'g' || key == 'G')

{

grid = !grid;

}

//включение/выключение света

if (key == 'l' || key == 'L')

{

g\_bLight = !g\_bLight;

}

//включение/выключение текстуры

if (key == 't' || key == 'T')

{

textureMod = !textureMod;

}

//движение камеры

if (key == 'w' || key == 'W')

{

camera1.move\_c(kSpeed);

}

if (key == 's' || key == 'S')

{

camera1.move\_c(-kSpeed);

}

if (key == 'd' || key == 'D')

{

camera1.r\_obs(camera1.View, -kSpeed \* 2.0f, 0.0f, 1.0f, 0.0f);

}

if (key == 'a' || key == 'A')

{

camera1.r\_obs(camera1.View, kSpeed\*2.0f, 0.0f, 1.0f, 0.0f);

}

//увеличить/уменьшить скорость перемещения камеры

if (key == '<')

{

kSpeed += 0.01;

}

if (key == '>')

{

kSpeed -= 0.01;

}

//возврат на исходную позицию

if (key == 'O' || key == 'o')

{

camera1.pos\_camera(-18.0f, 0.5f, -3.0f, -12.0f, 0.5f, -3.0f, 0.0f, 1.0f, 0.0f);

}

//изменение режима наложения текстур

if (key == 'x' || key == 'X')

{

if (!Mult)

Mult = true;

else

Mult = false;

}

//выбор типа освещения

if (key == '1')

light\_sours = 1;//фоновый

if (key == '2')

light\_sours = 2;//эмиссионный

if (key == '3')

light\_sours = 3;//точечный

if (key == '4')

light\_sours = 4;//прожекторный

if (key == '5')

light\_sours = 5;//удалённый направленный

}

void Process\_Special\_Keys(int key, int x, int y)

{

//движение камеры

if (key == GLUT\_KEY\_UP)

{

camera1.move\_c(kSpeed);

}

if (key == GLUT\_KEY\_DOWN)

{

camera1.move\_c(-kSpeed);

}

if (key == GLUT\_KEY\_LEFT)

{

camera1.r\_obs(camera1.View, -kSpeed \* 2.0f, 0.0f, 1.0f, 0.0f);

}

if (key == GLUT\_KEY\_RIGHT)

{

camera1.r\_obs(camera1.View, kSpeed\*2.0f, 0.0f, 1.0f, 0.0f);

}

glutPostRedisplay();

}

void Initialize()

{

Shine\_a\_Light(); //установить освещение

camera1.pos\_camera(-18.0f, 0.5f, -3.0f, -12.0f, 0.5f, -3.0f, 0.0f, 1.0f, 0.0f); //установить начальную позицию камеры

makeDuplication(); //рассчитать координаты тиражированной фигуры

calcNormals(); //вычислить нормали

smoothNormals(); //вычислить сглаженные нормали

loadTexture(); //загрузить текстуру

}

norm.cpp

#include "norm.h"

//Вычисление нормали двух векторов

Vector3f Cross(Vector3f \_Vec1, Vector3f \_Vec2)

{

Vector3f \_Normal;

//вычисление векторного произведения

\_Normal.x = ((\_Vec1.y\*\_Vec2.z) - (\_Vec1.z\*\_Vec2.y));

\_Normal.y = ((\_Vec1.z\*\_Vec2.x) - (\_Vec1.x\*\_Vec2.z));

\_Normal.z = ((\_Vec1.x\*\_Vec2.y) - (\_Vec1.y\*\_Vec2.x));

return \_Normal;

}

//Вычисление нормы вектора

float Norm(Vector3f \_Vec)

{

return (float)sqrt((\_Vec.x\*\_Vec.x) + (\_Vec.y\*\_Vec.y) + (\_Vec.z\*\_Vec.z));

}

//Нормализация вектора

Vector3f Normalize(Vector3f \_Vec)

{

//Вычислить норму вектора

float norm = Norm(\_Vec);

//нормализовать вектор

\_Vec = \_Vec / norm;

return \_Vec;

}

//Вычисление скалярного произведения

float scalar(Vector3f \_Vec1, Vector3f \_Vec2)

{

return \_Vec1.x\*\_Vec2.x + \_Vec1.y\*\_Vec2.y + \_Vec1.z\*\_Vec2.z;

}

//Вектор между двумя точками

Vector3f Vector(Vector3f \_Point1, Vector3f \_Point2)

{

Vector3f \_Vector;

\_Vector.x = \_Point1.x - \_Point2.x;

\_Vector.y = \_Point1.y - \_Point2.y;

\_Vector.z = \_Point1.z - \_Point2.z;

return \_Vector;

}

//Вычисление нормали полигона

Vector3f norm\_pol(Vector3f one, Vector3f two, Vector3f tr)

{

Vector3f \_Vector1 = Vector(tr, two);

Vector3f \_Vector2 = Vector(two, one);

Vector3f \_Normal = Cross(\_Vector1, \_Vector2);

\_Normal = Normalize(\_Normal);

return \_Normal;

}

Camera.cpp

#include "Camera.h"

Camera::Camera()

{

}

void Camera::pos\_camera(float posX, float posY, float posZ,

float viewX, float viewY, float viewZ,

float upX, float upY, float upZ)

{

//установить позицию камеры

Vector3f \_Position = Vector3f(posX, posY, posZ);

Vector3f \_View = Vector3f(viewX, viewY, viewZ);

Vector3f \_rot\_vect = Vector3f(upX, upY, upZ);

Pos = \_Position;

View = \_View;

rot\_vect = \_rot\_vect;

}

void Camera::move\_c(float speed)

{

Vector3f \_View = View - Pos; //определить направление взгляда

Pos.x += \_View.x \* speed; //изменить положение

View.x += \_View.x \* speed;

Pos.z += \_View.z \* speed; //камеры

View.z += \_View.z \* speed;

Pos.y += \_View.y \* speed;

View.y += \_View.y \* speed;

}

void Camera::r\_view(float angle, float x, float y, float z)

{

Vector3f \_newView;

Vector3f \_View;

//определить направление взгляда

\_View = View - Pos;

//рассчитать синус и косинус переданного угла

float cosA = (float)cos(angle);

float sinA = (float)sin(angle);

//пересчитаем координаты по каким-то диким формулам

//новая координата X для вращаемой точки

\_newView.x = (cosA + (1 - cosA)\*x\*x) \* \_View.x;

\_newView.x += ((1 - cosA)\*x\*y - z \* sinA) \* \_View.y;

\_newView.x += ((1 - cosA)\*x\*z + y \* sinA) \* \_View.z;

//новая координата Y для вращаемой точки

\_newView.y = ((1 - cosA)\*x\*y + z \* sinA) \* \_View.x;

\_newView.y += (cosA + (1 - cosA)\*y\*y) \* \_View.y;

\_newView.y += ((1 - cosA)\*y\*z - x \* sinA) \* \_View.z;

//новая координата Z для вращаемой точки

\_newView.z = ((1 - cosA)\*x\*z - y \* sinA) \* \_View.x;

\_newView.z += ((1 - cosA)\*y\*z + x \* sinA) \* \_View.y;

\_newView.z += (cosA + (1 - cosA)\*z\*z) \* \_View.z;

//установить новый взгляд камеры

View.x = Pos.x + \_newView.x;

View.y = Pos.y + \_newView.y;

View.z = Pos.z + \_newView.z;

}

void Camera::set\_view\_mouse(GLint width, GLint height)

{

POINT mousePos; //позиция мыши

//вычислить координаты центра окна

int middleX = width / 2.0f;

int middleY = height / 2.0f;

float angleY = 0.0f; //Направление взгляда вверх/вниз

float angleZ = 0.0f; //Значение, необходимое для вращения влево-вправо (по оси Y)

static float currentRotX = 0.0f;

//получить текущие координаты мыши

GetCursorPos(&mousePos);

//если положение мыши не изменилось

//камеру вращать не нужно

if (mousePos.x == middleX && mousePos.y == middleY) return;

//вернуть координаты курсора в центр окна

SetCursorPos(middleX, middleY);

//определить, куда был сдвинут курсор

angleY = (float)((middleX - mousePos.x)) / 1000.0f;

angleZ = (float)((middleY - mousePos.y)) / 1000.0f;

static float lastRotX = 0.0f;

lastRotX = currentRotX; //Сохраняем последний угол вращения

//и используем заново currentRotX

//если поворот больше одного градуса, умешьшим его

//чтобы уменьшить скорость вращения

if (currentRotX > 1.0f)

{

currentRotX = 1.0f;

if (lastRotX != 1.0f)

{

//Чтобы найти ось, вокруг которой нужно совершать вращение вверх и вниз, нужно

//найти вектор, перпендикулярный вектору взгляда камеры и

//вертикальному вектору

Vector3f vAxis = Cross(View - Pos, rot\_vect);

//нормализуем ось

vAxis = Normalize(vAxis);

//вращаем камеру вокруг нашей оси на заданный угол

r\_view(1.0f - lastRotX, vAxis.x, vAxis.y, vAxis.z);

}

}

//Если угол меньше -1.0f

else if (currentRotX < -1.0f)

{

currentRotX = -1.0f;

if (lastRotX != -1.0f)

{

//вычисляем ось

Vector3f vAxis = Cross(View - Pos, rot\_vect);

//нормализуем ось

vAxis = Normalize(vAxis);

//вращаем

r\_view(-1.0f - lastRotX, vAxis.x, vAxis.y, vAxis.z);

}

}

//если в пределах 1.0f -1.0f - просто вращаем

else

{

Vector3f vAxis = Cross(View - Pos, rot\_vect);

vAxis = Normalize(vAxis);

r\_view(angleZ, vAxis.x, vAxis.y, vAxis.z);

}

//всегда вращаем камеру вокруг Y-оси

r\_view(angleY, 0, 1, 0);

}

void Camera::r\_obs(Vector3f \_Center, float angle, float x, float y, float z)

{

Vector3f \_NewPosition;

//получим центр, вокруг которого нужно вращатся

Vector3f vPos = Pos - \_Center;

//вычислим синус и косинус угла

float cosA = (float)cos(angle);

float sinA = (float)sin(angle);

//Найдем значение X точки вращения

\_NewPosition.x = (cosA + (1 - cosA)\*x\*x) \* vPos.x;

\_NewPosition.x += ((1 - cosA)\*x\*y - z \* sinA) \* vPos.y;

\_NewPosition.x += ((1 - cosA)\*x\*z + y \* sinA) \* vPos.z;

//значение Y

\_NewPosition.y = ((1 - cosA)\*x\*y + z \* sinA) \* vPos.x;

\_NewPosition.y += (cosA + (1 - cosA)\*y\*y) \* vPos.y;

\_NewPosition.y += ((1 - cosA)\*y\*z - x \* sinA) \* vPos.z;

//значение Z

\_NewPosition.z = ((1 - cosA)\*x\*z - y \* sinA) \* vPos.x;

\_NewPosition.z += ((1 - cosA)\*y\*z + x \* sinA) \* vPos.y;

\_NewPosition.z += (cosA + (1 - cosA)\*z\*z) \* vPos.z;

//установить новую позицию камеры

Pos = \_Center + \_NewPosition;

}

main.cpp

#include "sours.h"

int main(int argc, char \*argv[])

{

width = 1200;

height = 500;

glutInit(&argc, argv);

//включить буфер глубины/двойную буферизацию

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowSize(1200, 800);

glutCreateWindow("Тиражированный треугольник");

Initialize(); //начальные установки

glutKeyboardFunc(Process\_Normal\_Keys); //обработка клавиш с кодами ascii

glutSpecialFunc(Process\_Special\_Keys); //обработка не-ascii клавиш

glutDisplayFunc(Display);

glutReshapeFunc(Reshape);

glutIdleFunc(Display);

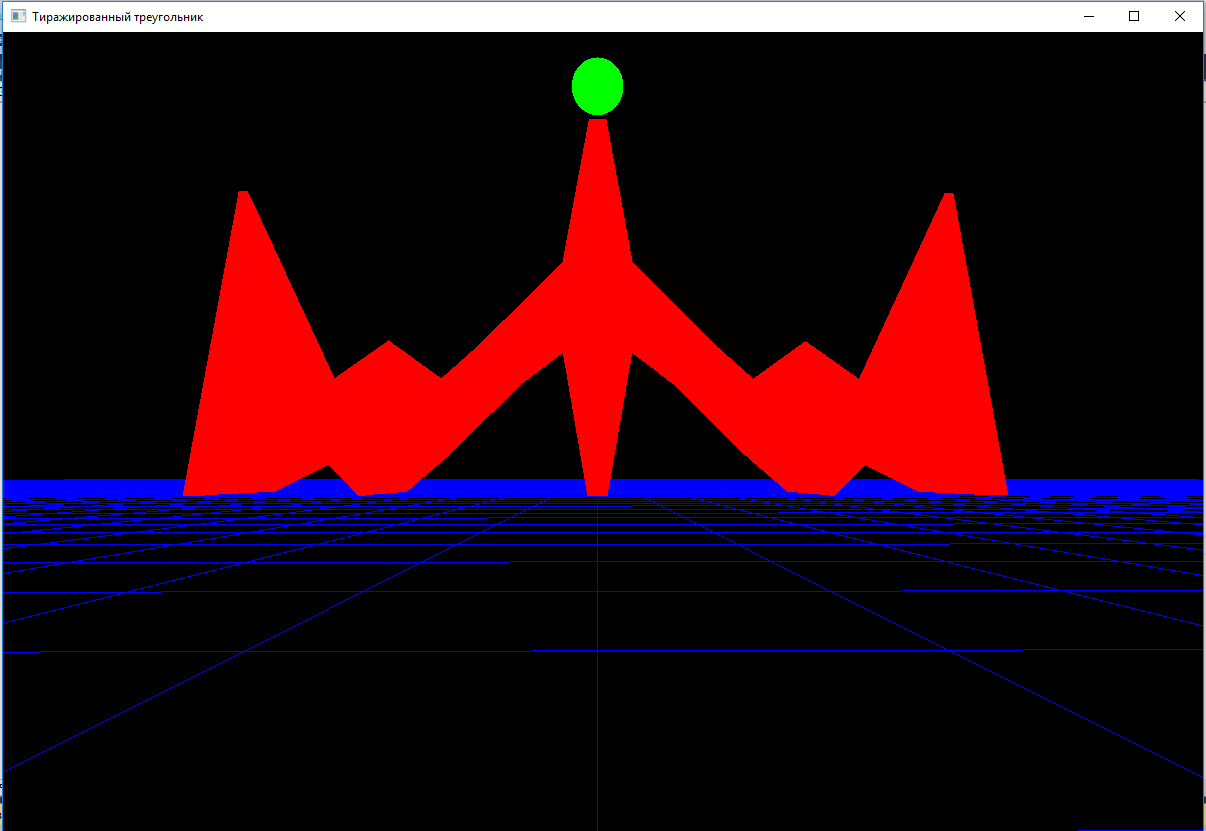
glutPassiveMotionFunc(Process\_Mouse\_Move);

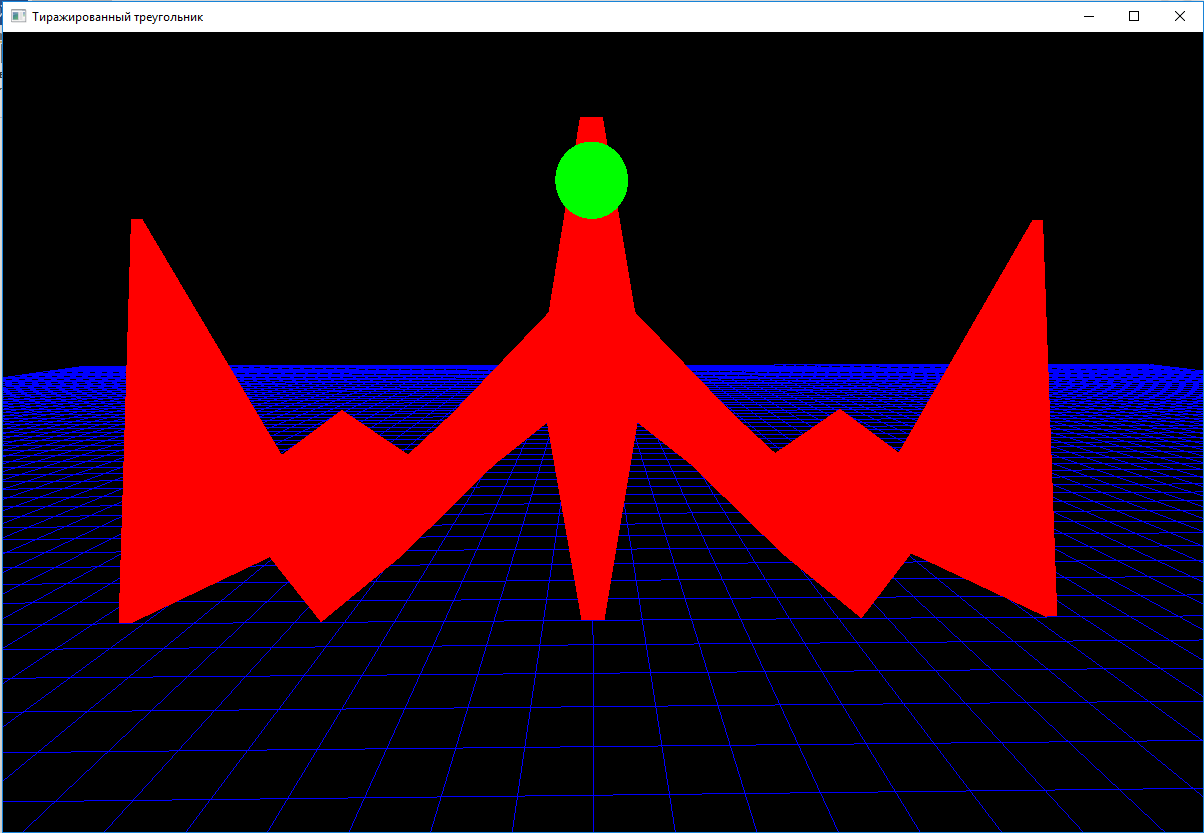
glutMainLoop();

}

1. Тесты

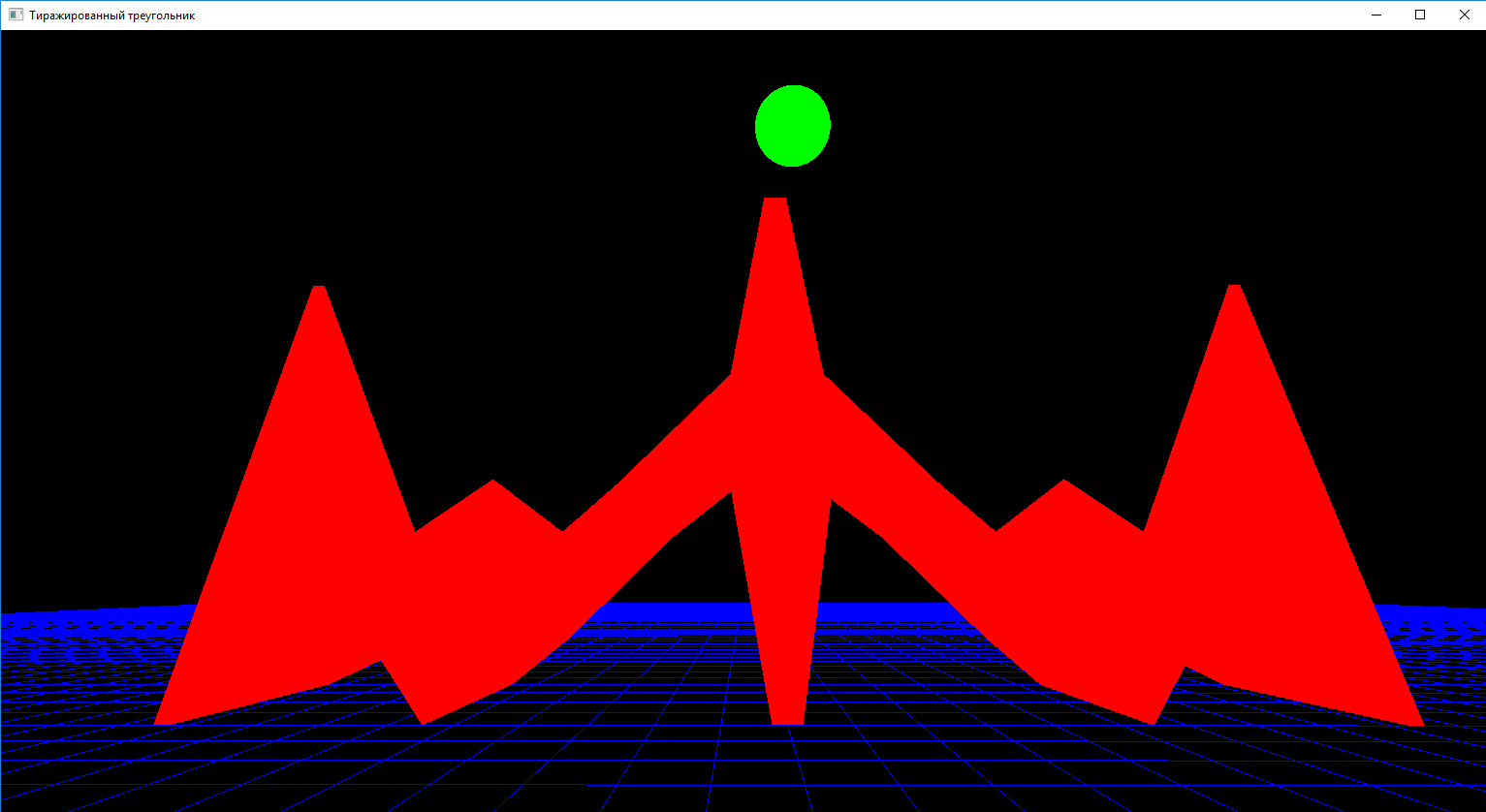
Тест 1. Перевемещение при момощи мыши и клавиатуры.



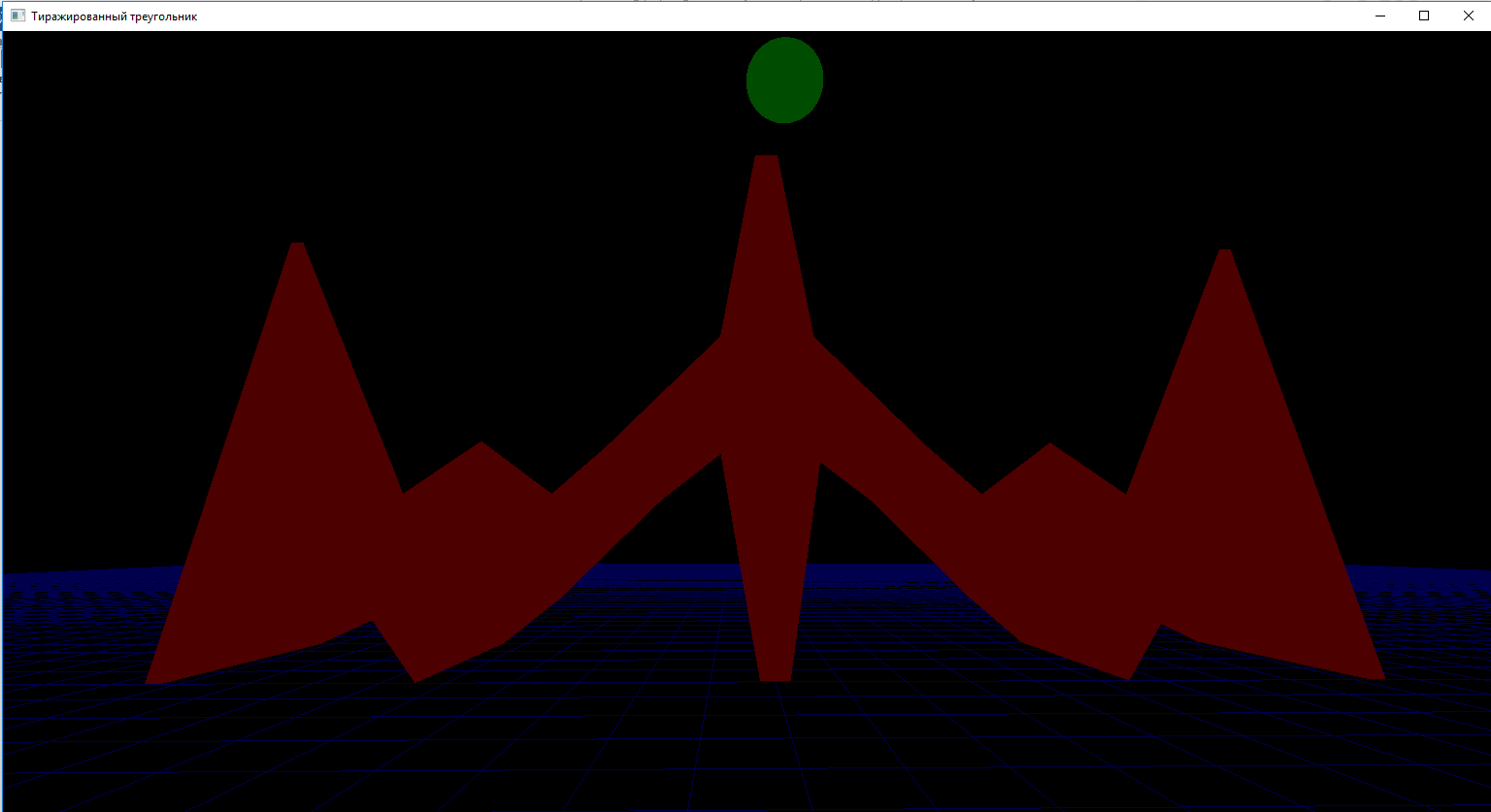


Тест 2. Выбор источника освещения.

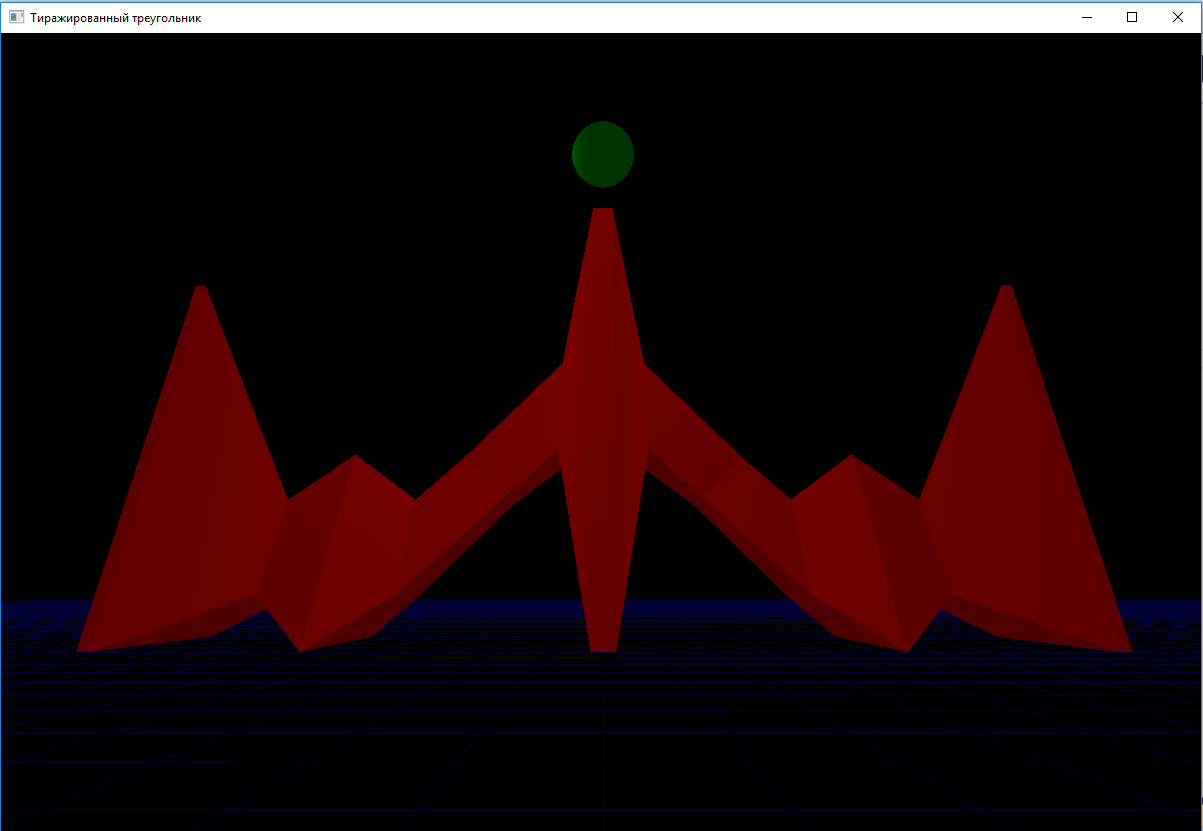
1. Фоновый.



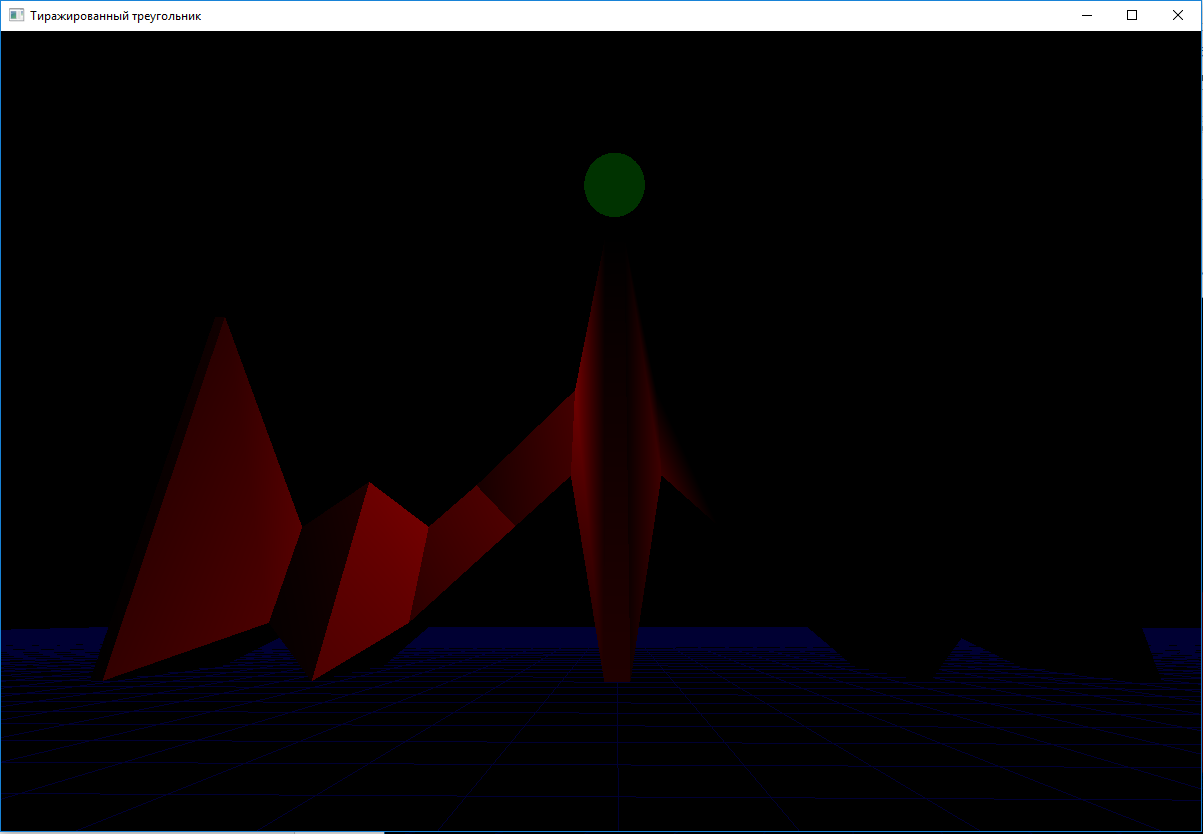
1. Эмиссионный.



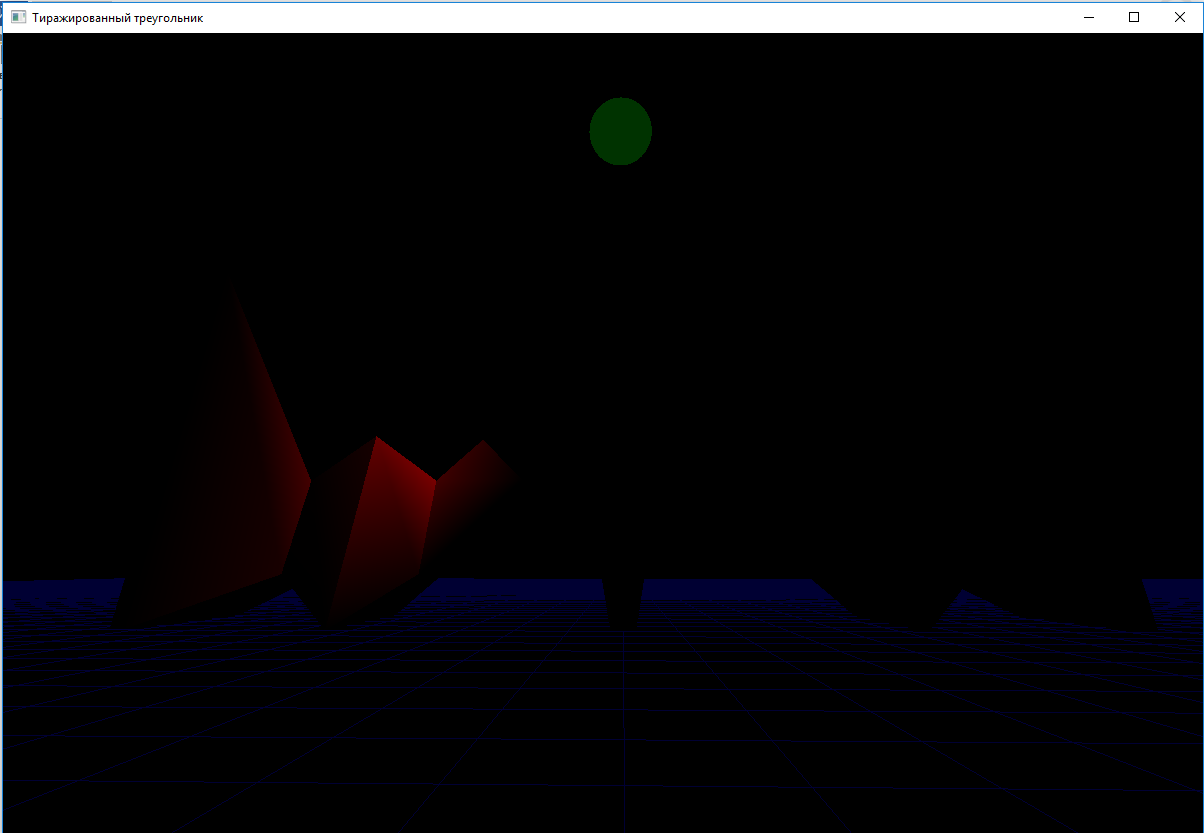
1. Точечный.



1. Прожекторный.

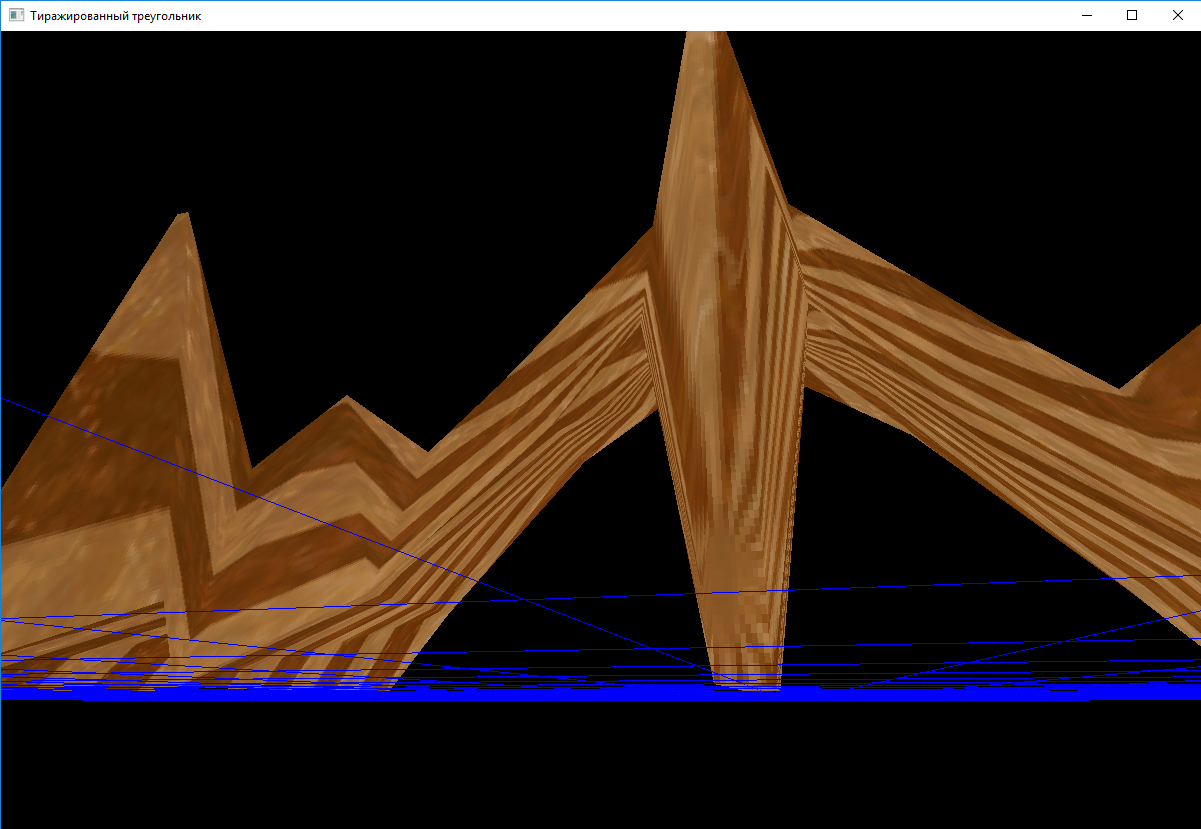


1. Удалённый. Направленный.

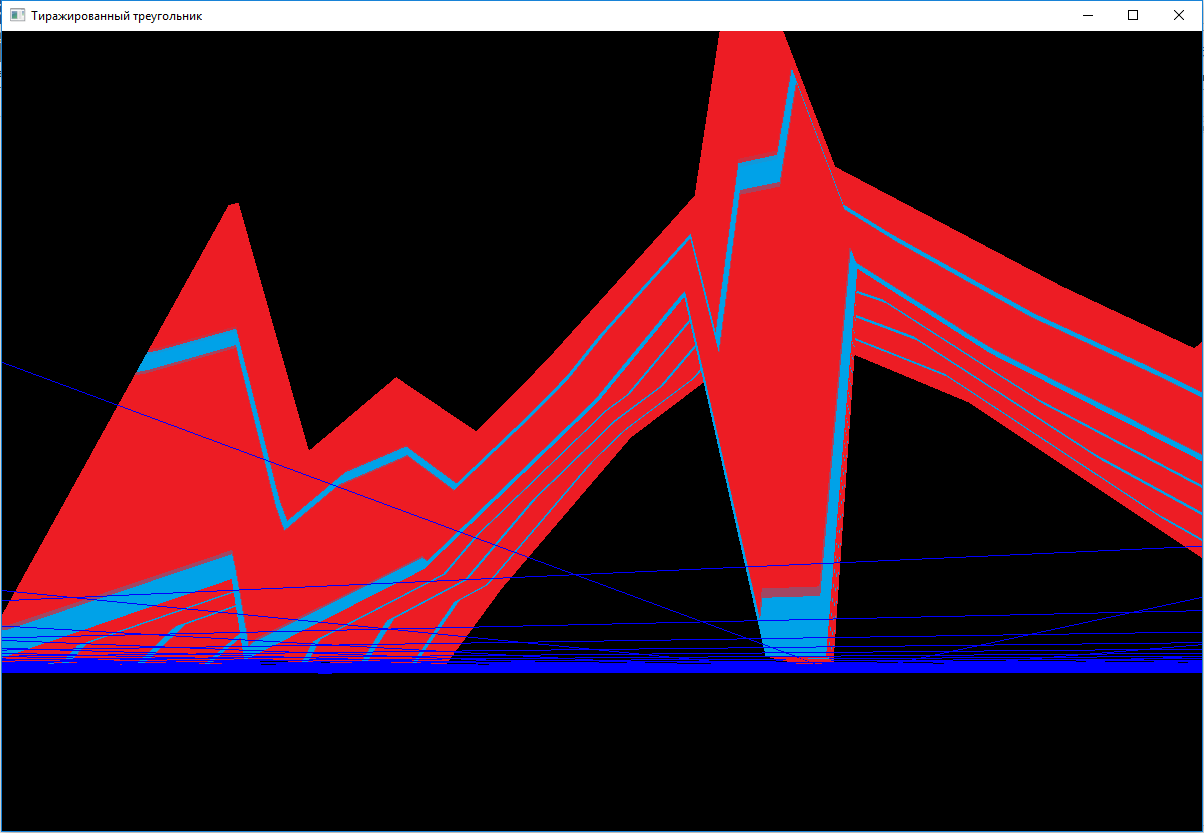


Тест 3. Текстурирование.

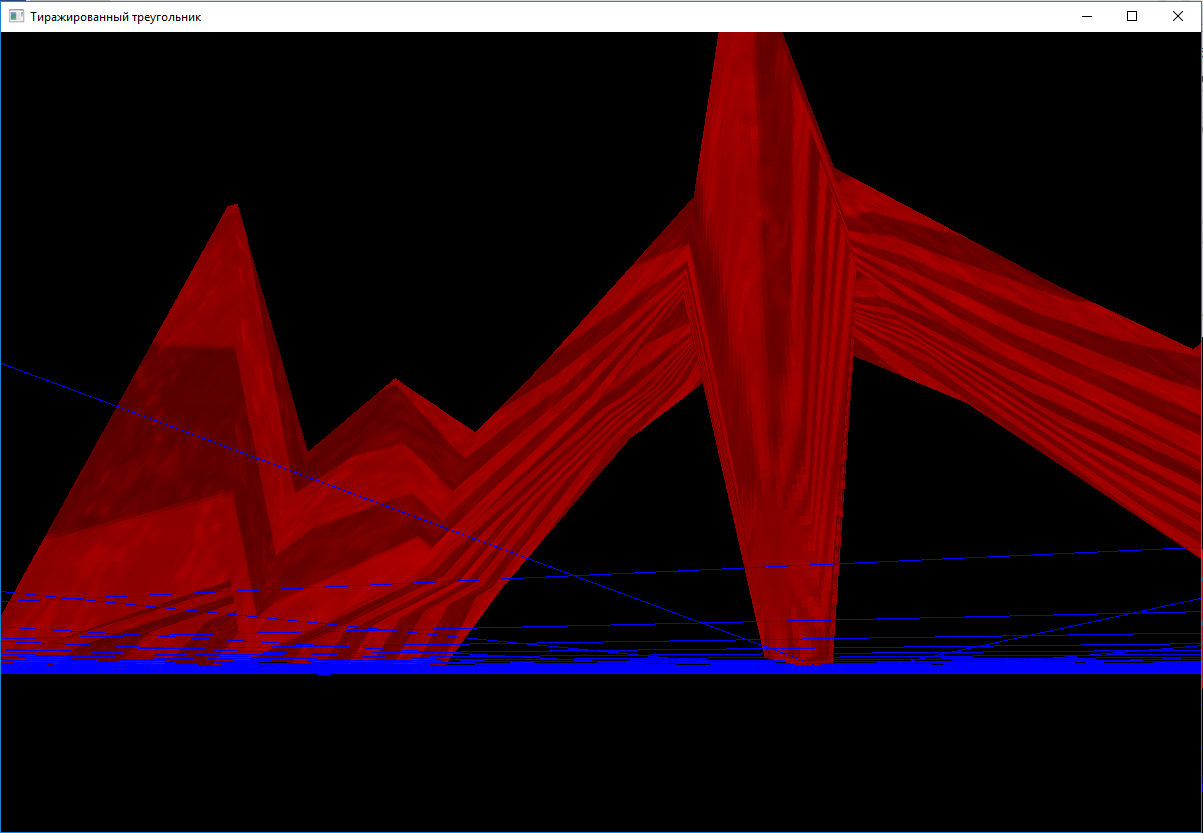
Текстура оборачивает всю модель.



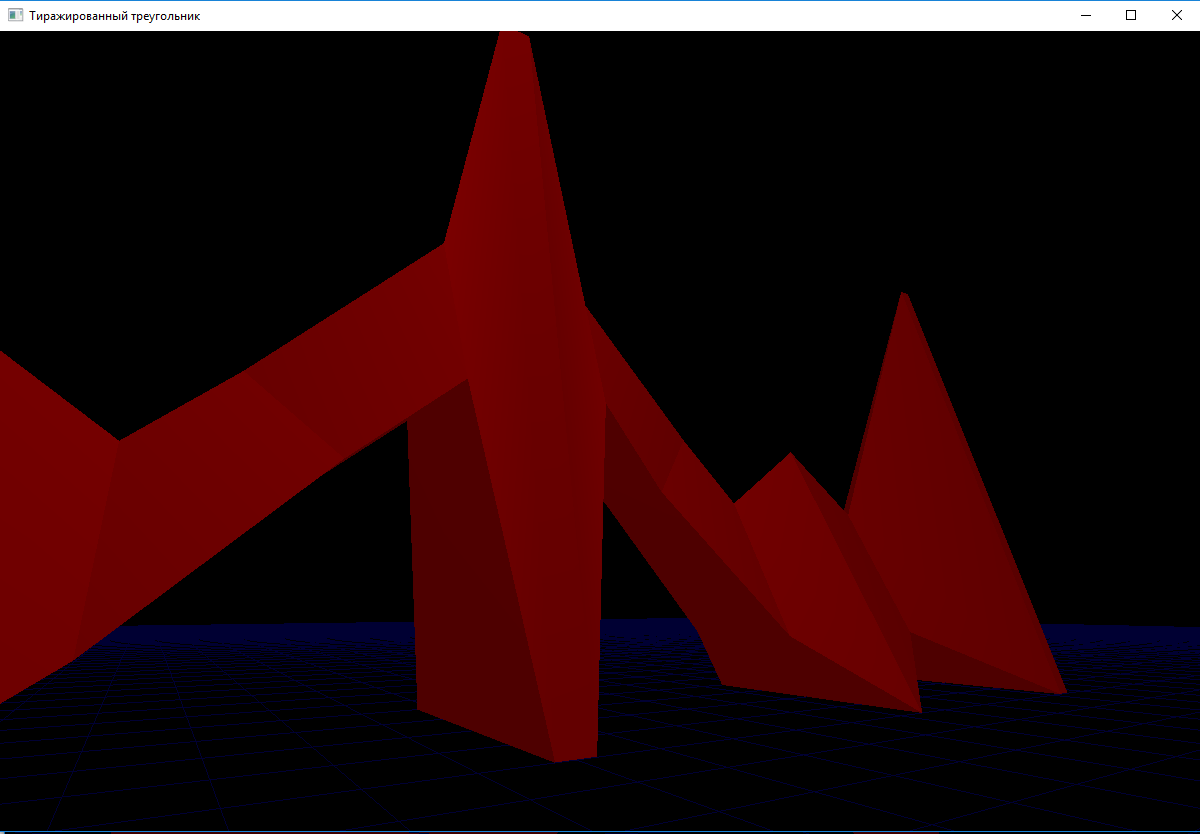
Выбор другой текстуры.

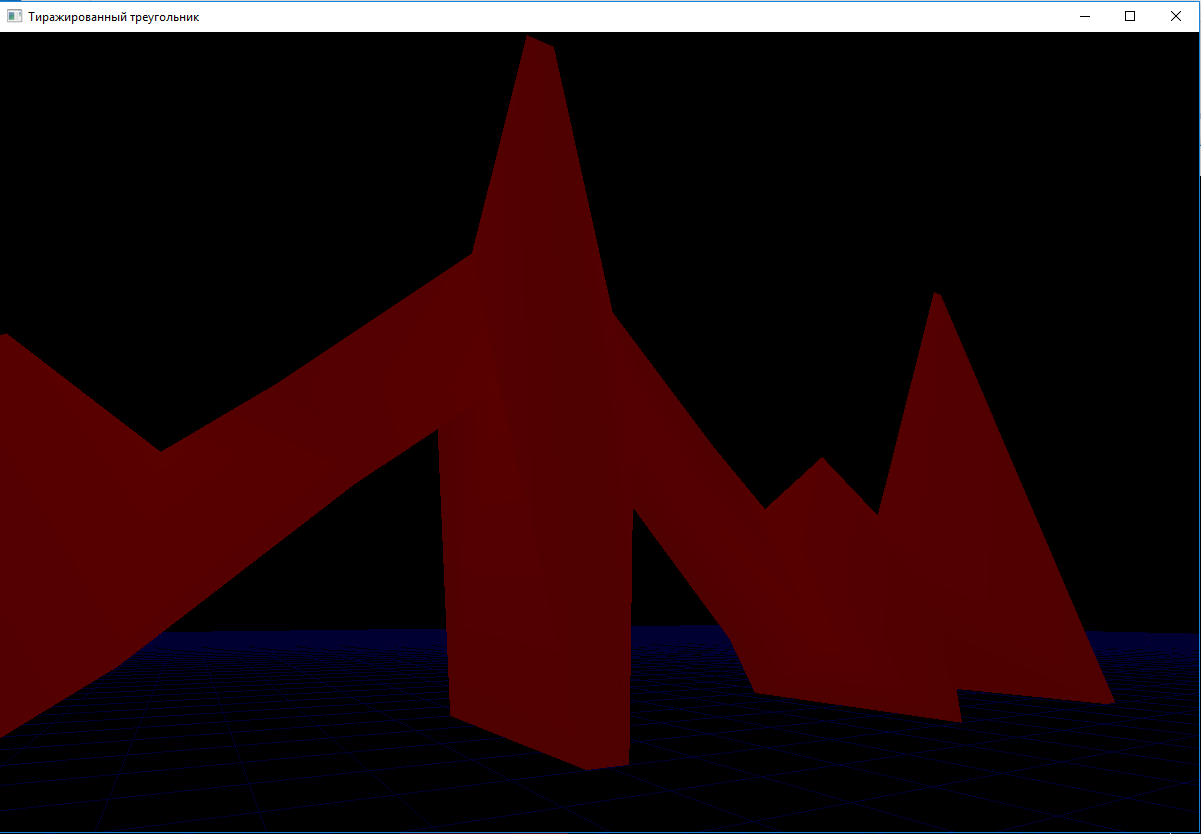


Наложение текстуры с учтом цвета модели.



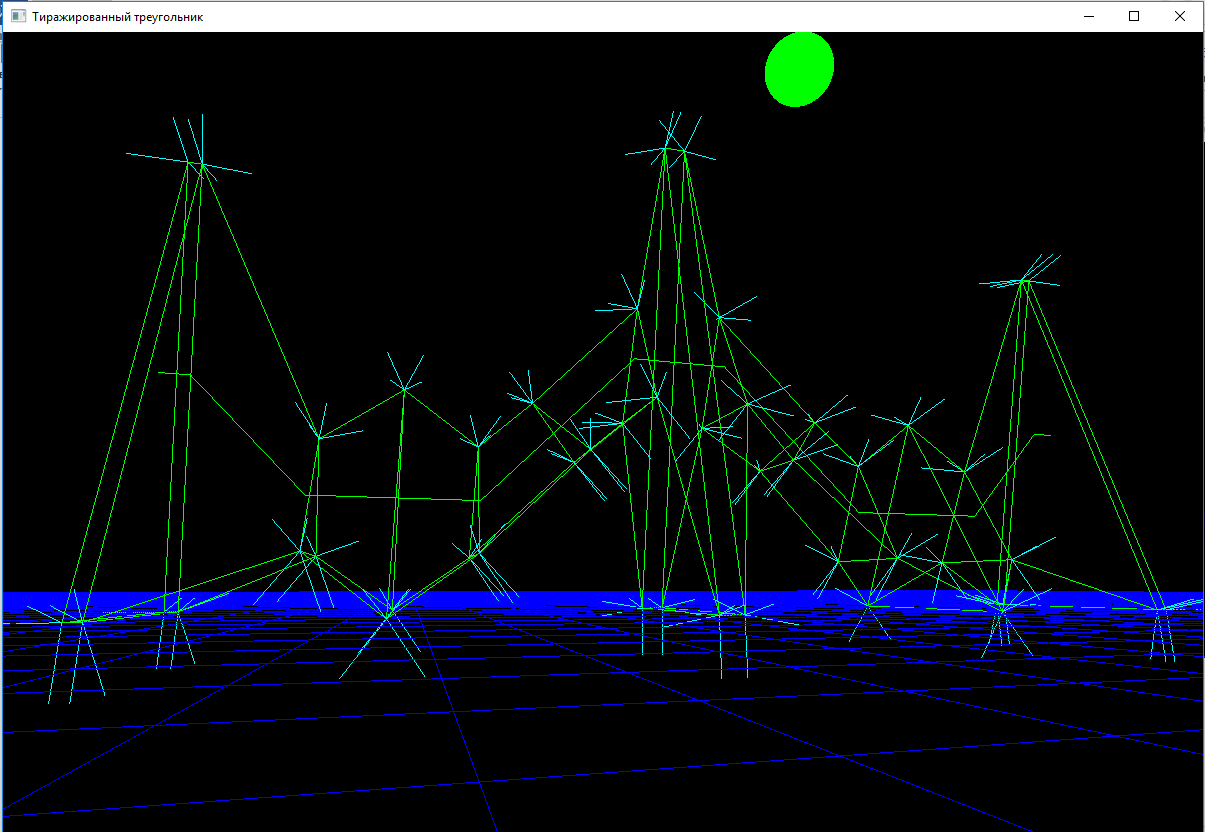
Тест 4. Скглаживание нормалей.



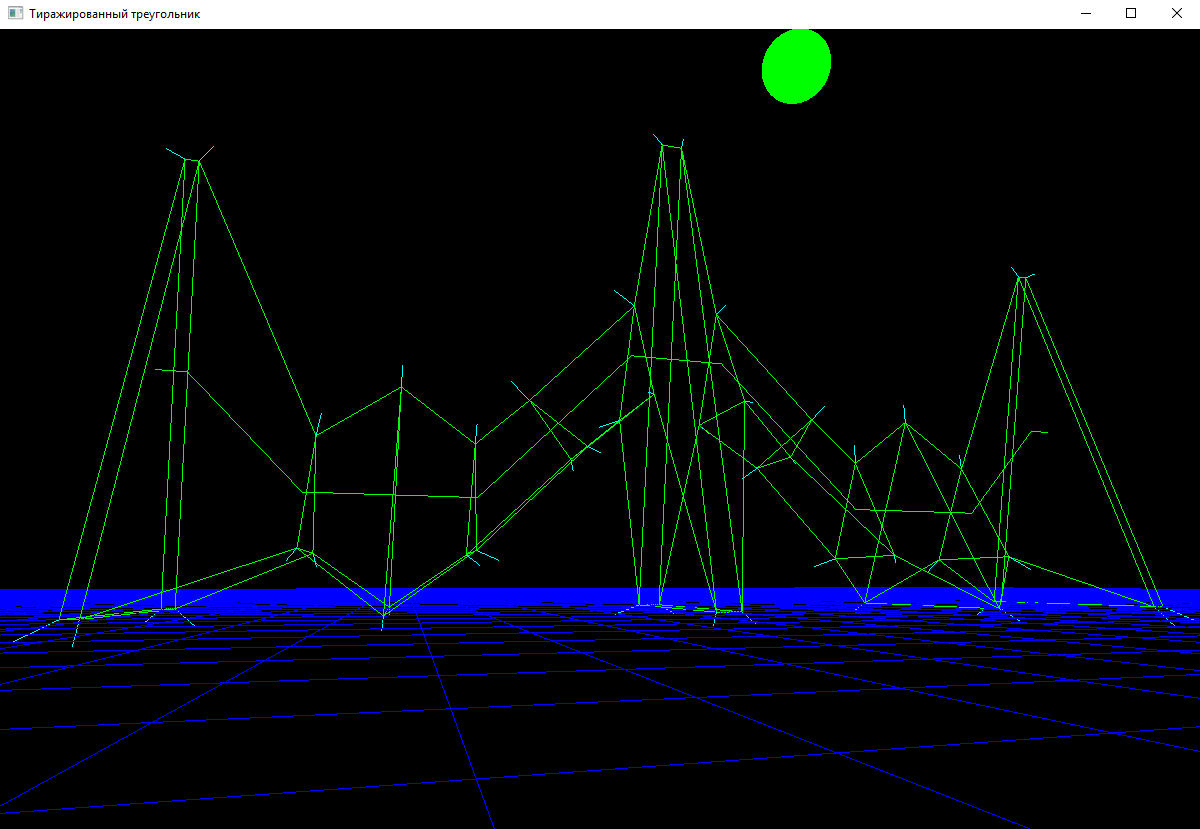


Тест 5. Каркасный режим и демострация нормалей.

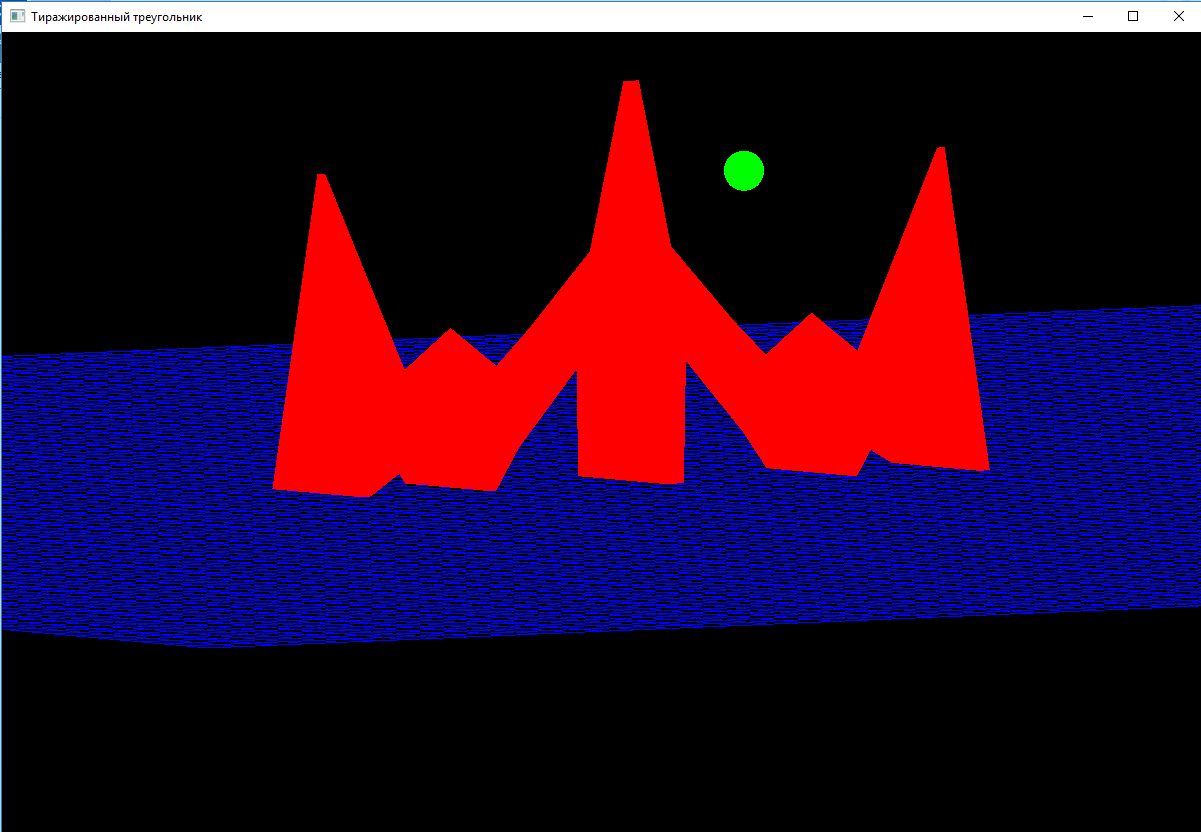
Для фигуры используется процентная траектория.



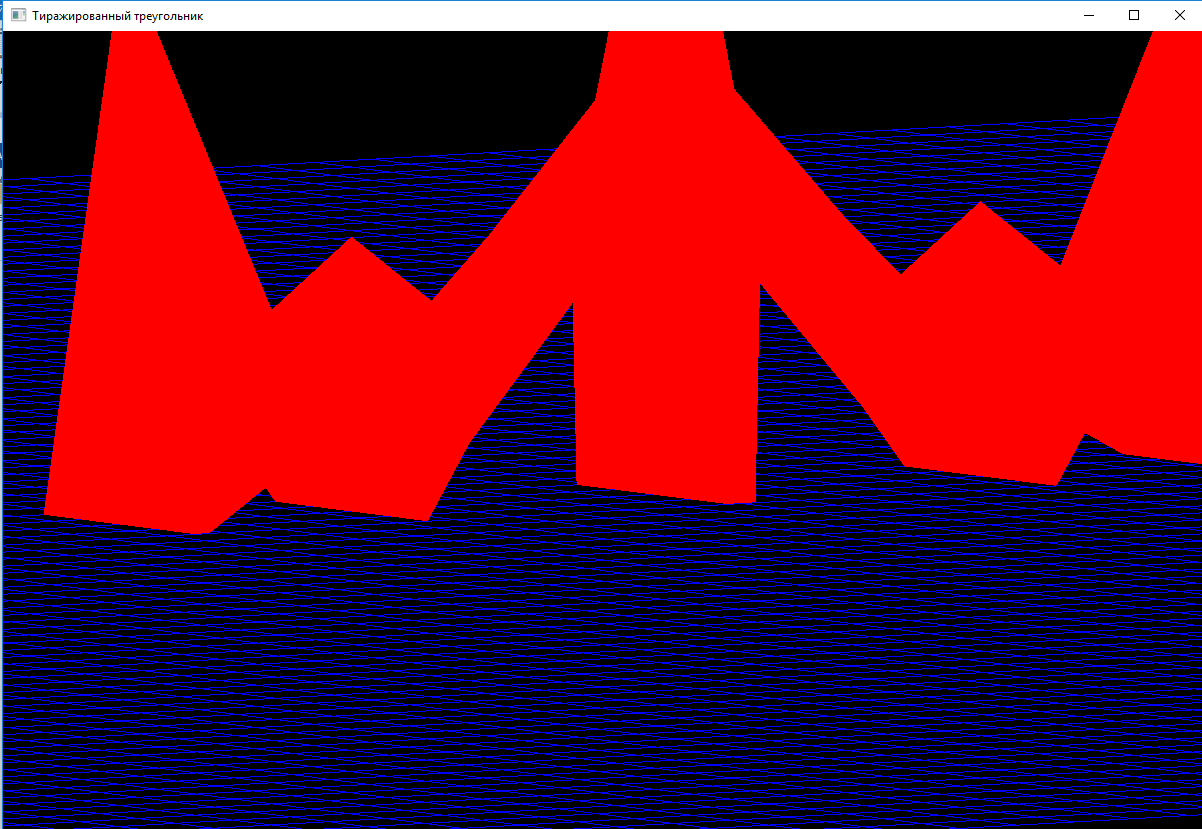
Нормали сглажены.



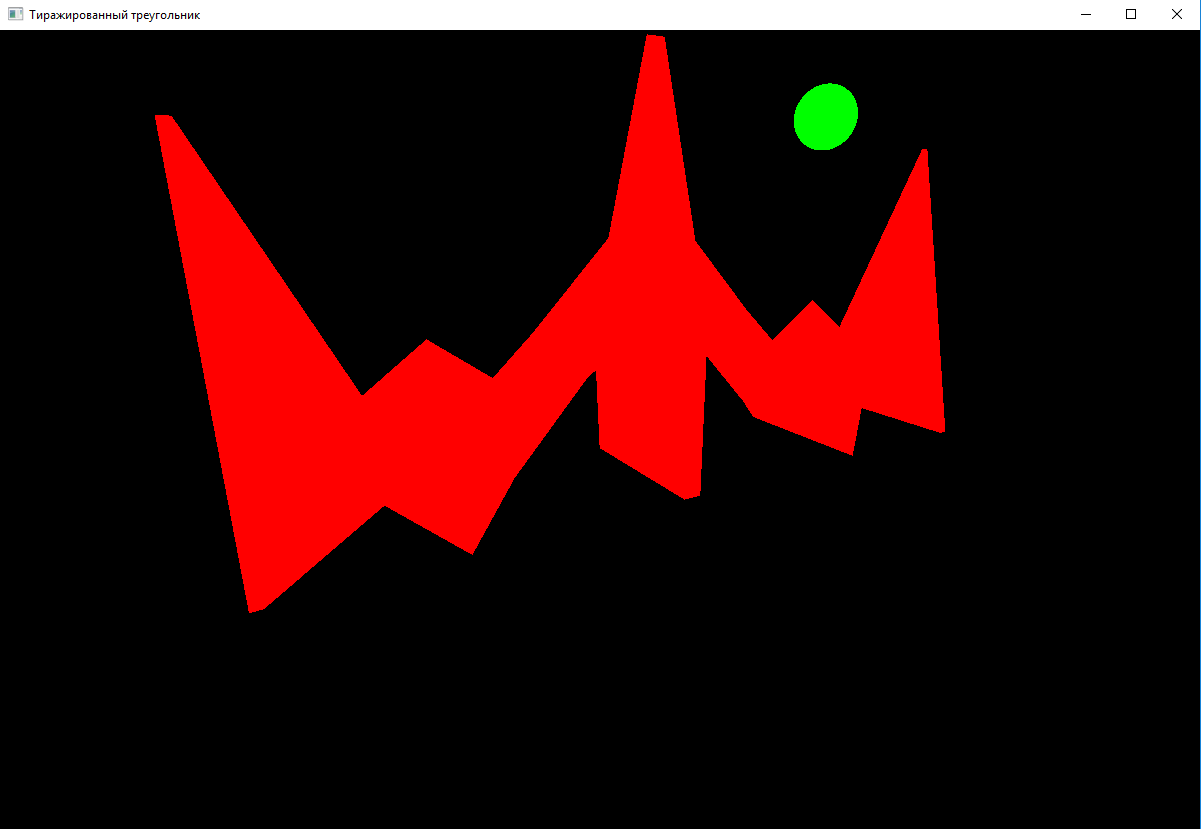
Тест 6. Проекционный режим.



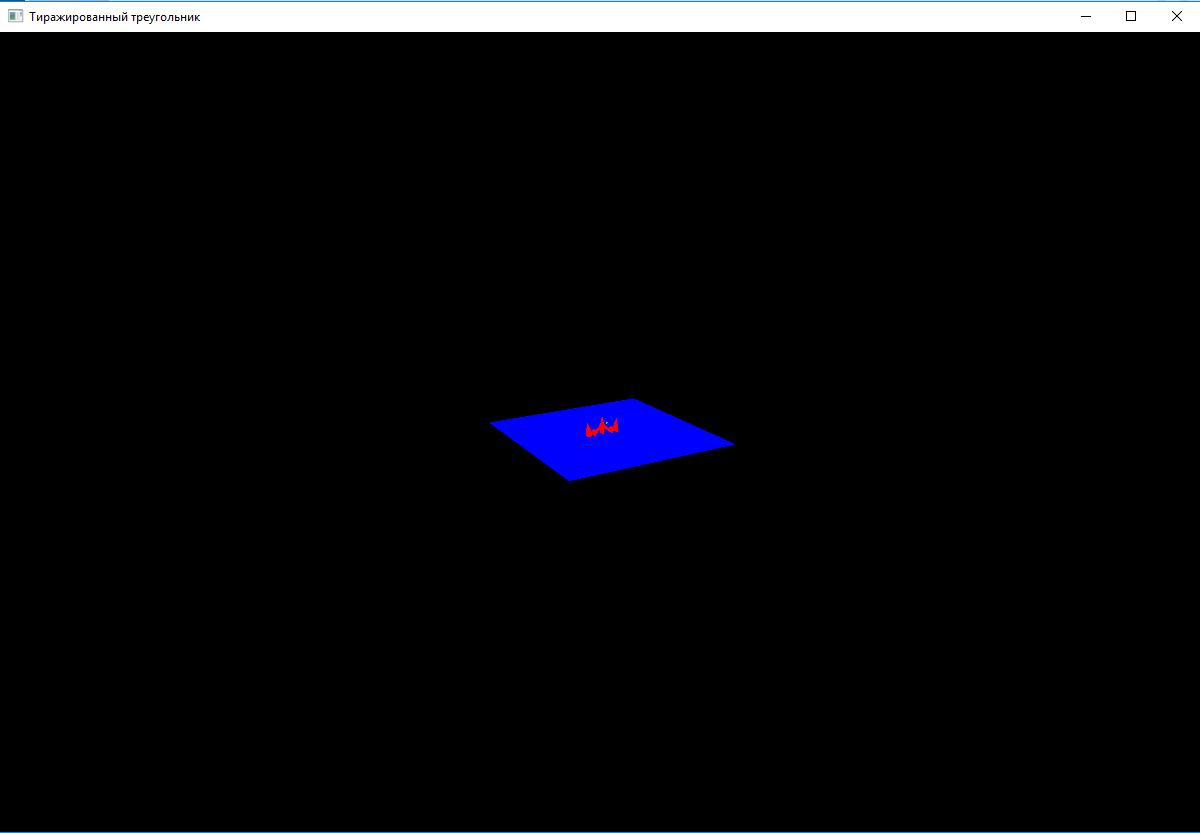
Приближение к модели.



Тест7. Отключение сетки.



Тест 8. Проверка куба видимости.



Постепеннео отдаление.