



МИНИСТЕРСТВО НАУКИ  
И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное  
образовательное учреждение высшего образования  
«Новосибирский государственный технический университет»



Новосибирский  
государственный  
технический университет  
**НЭТИ**

Кафедра прикладной математики  
Практическая работа №2  
по дисциплине «Компьютерная графика»



Факультет:	ПМИ
Группа:	ПМ-71
Студенты:	Востриков Вячеслав, Бурдуков Вадим, Баштовой Павел
Преподаватели:	Задорожный Александр Геннадьевич
Дата:	4.06.2020

Новосибирск  
2020

## 1) Цель работы

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

## 2) Текст задания

1. Считать из файла:
  - а. 2D – координаты вершин сечения (считающегося выпуклым);
  - б. 3D – координаты траектории тиражирования;
  - с. параметры изменения сечения.
2. Построить фигуру в 3D по прочитанным данным.
3. Включить режимы:
  - а. буфера глубины;
  - б. двойной буферизации;
  - с. освещения и материалов.
4. Предоставить возможность показа:
  - а. каркаса объекта;
  - б. нормалей;
  - с. текстур.
5. Предоставить возможность переключения между режимами ортогографической 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.
8. Построение фигуры осуществлять с использованием процентной траектории.
9. Использовать различные источники света.

## 3) Структура программы

Данная программа имела универсальную цель: умение построить 3д фигуры с разными количествами вершин. Поэтому программа состоит из двух главных этапов. Первый этап заключен в заголовочном файле «library.h». Второй этап реализован в main.cpp

В первом этапе происходит инициализация фигуры, а именно построение всевозможных поверхностей, по считанным точкам, объединение плоскостей,

которые параллельны и удаление лишних (внутренних плоскостей), подсчет векторов нормалей.

Второй этап заключается в том, чтобы эту фигуру отображать (текстурирование, свет, показ сглаженных/ несглаженных нормалей и т.д.)

#### 4) Код программы

Файл main.cpp:

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdlib.h>
#include <vector>
#include <fstream>
#include <stack>
#include <math.h>
#include "library.h"
#include "glut.h"
#include <soil.h> // Через Nuget Устанавливаем библиотеки для загрузки текстур

int Width = 800, Height = 800;

#define Pi 3.1415926

using namespace std;

object figure;
//point_3d spectator(0, 0, 0.1); // x - расстояние от объекта

bool buff_depth = 0;
bool double_buff = 1;
bool light = 0;
int type_source_light = 0;

bool carcass = 0;
bool show_normals = 0;

unsigned char* tex_2d = 0; // stat_load_texture ... Сама картинка
int width_text;
int height_text;
bool texture = 1;
bool stat_load_texture = 0;
FILE* file = fopen("texture.bmp", "rb");

bool ortho_or_perspective = 0; // 0 - ортографич, 1 - перспективная
float perspective_angle = 30;
float perspective_ratio = 1;

bool smooth_normal = 0;
#define Max_size 50
int matrix_smezhnost[Max_size][Max_size];

typedef struct system_sphere {
    type_coordinates Radius, Alpha, Theta;

    system_sphere(type_coordinates rad, type_coordinates alph, type_coordinates beth) {
        this->Radius = rad;
        this->Alpha = alph;
        this->Theta = beth;
    }
}Sphere_syst;
Sphere_syst SystemCoordinate(0, 0, 1);
```

```

void clear_matrix(){
    for (int i = 0; i < Max_size; i++)
        for (int j = 0; j < Max_size; j++)
            matrix_smezhnost[i][j] = 0;
}

void create_neib_matrix(section sk) {
    int in1 = 0, in2 = 0;
    for (int i = 0; i < sk.ribs.size(); i++)
    {
        for (int j = 0; j < sk.points.size(); j++)
            if (sk.ribs[i].a == sk.points[j])
                in1 = j;

        for (int j = 0; j < sk.points.size(); j++)
            if (sk.ribs[i].b == sk.points[j])
                in2 = j;

        matrix_smezhnost[in1][in2] = 1;
        matrix_smezhnost[in2][in1] = 1;
    }
}

void step(vector<int>& seq, section sk, int current_index, int prev) {
    for (int i = 0; i < sk.points.size(); i++)
    {
        if (matrix_smezhnost[current_index][i] == 1 && i == 0 && prev != 0)
            break;

        if (matrix_smezhnost[current_index][i] == 1 && i != prev && i != current_index)
        {
            seq.push_back(i);
            step(seq, sk, i, current_index);
            break;
        }
    }
}

void show_normals_func(object fig) {
    for (int i = 0; i < fig.platan.size(); i++)
    {
        point_3d center_of_platan, a, b;
        for (int j = 0; j < fig.platan[i].points.size(); j++)
            center_of_platan = center_of_platan + fig.platan[i].points[j];
        center_of_platan = center_of_platan / fig.platan[i].points.size(); // Нашли
середину поверхности
        a = center_of_platan + fig.platan[i].vec_normal / 5;
        b = center_of_platan - fig.platan[i].vec_normal / 5;

        glBegin(GL_LINES); // Просто линия
        glLineWidth(5000);
        glColor3f(0.0f, 0.0f, 0.0f); // Черный

        glVertex3d(a.x - figure.center_of_mass.x
            , a.y - figure.center_of_mass.y
            , a.z - figure.center_of_mass.z);
        glVertex3d(b.x - figure.center_of_mass.x
            , b.y - figure.center_of_mass.y
            , b.z - figure.center_of_mass.z);
        glEnd();
    }
}

```

```

}

void return_text_coord(section &sk, section z) // Пересчитываем текстурные координаты (проблема с переводом из 3-х мерного пространства)
{
    type_coordinates x_min, x_max, y_min, y_max;
    x_min = z.points[0].x;
    x_max = z.points[0].x;
    y_min = z.points[0].y;
    y_max = z.points[0].y;

    sk = z;
    for (int i = 0; i < sk.points.size(); i++)
    {
        if (sk.points[i].x < x_min)
            x_min = sk.points[i].x;
        else if (sk.points[i].x > x_max)
            x_max = sk.points[i].x;

        if (sk.points[i].y < y_min)
            y_min = sk.points[i].y;
        else if (sk.points[i].y > y_max)
            y_max = sk.points[i].y;
    }

    if (x_min < 0)
    {
        for (int i = 0; i < sk.points.size(); i++)
            sk.points[i].x = sk.points[i].x + abs(x_min);
    }
    else if (x_min > 0)
    {
        for (int i = 0; i < sk.points.size(); i++)
            sk.points[i].x = sk.points[i].x - x_min;
    }

    if (y_min < 0)
    {
        for (int i = 0; i < sk.points.size(); i++)
            sk.points[i].y = sk.points[i].y + abs(y_min);
    }
    else if (y_min > 0)
    {
        for (int i = 0; i < sk.points.size(); i++)
            sk.points[i].y = sk.points[i].y - y_min;
    }

    type_coordinates f = y_max - y_min,
        g = x_max - x_min;

    for (int i = 0; i < sk.points.size(); i++)
    {
        sk.points[i].x = sk.points[i].x / g;
        sk.points[i].y = sk.points[i].y / f;
    }
    return;
}

point_3d Sphere_to_Dekart(Sphere_syst SK)
{
    point_3d pos;
    pos.z = SK.Radius * cos(SK.Theta * Pi / 180);
    pos.x = SK.Radius * sin(SK.Theta * Pi / 180) * cos(SK.Alpha * Pi / 180);
    pos.y = SK.Radius * sin(SK.Theta * Pi / 180) * sin(SK.Alpha * Pi / 180);
}

```

```

        return pos;
    }

void Draw_smooth_normals() {
    Point3D k(0, 0, 0);
    for (int i = 0; i < figure.mass_point.size(); i++)
    {
        int count_normals_for_point = 0;
        for (int j = 0; j < figure.platan.size(); j++)
            if (have_platan_the_point(figure.platan[j], figure.mass_point[i]))
            {
                k = k + figure.platan[j].vec_normal;
                count_normals_for_point++;
            }

        k = k / count_normals_for_point;

        Point3D a(k.x + figure.mass_point[i].x,
                  k.y + figure.mass_point[i].y,
                  k.z + figure.mass_point[i].z);

        Point3D b(-k.x + figure.mass_point[i].x,
                  -k.y + figure.mass_point[i].y,
                  -k.z + figure.mass_point[i].z);

        glBegin(GL_LINES); // Просто линия
        glLineWidth(5000);
        glColor3f(0.0f, 0.0f, 0.0f); // Черный
        glVertex3d(a.x - figure.center_of_mass.x,
                  a.y - figure.center_of_mass.y,
                  a.z - figure.center_of_mass.z);
        glVertex3d(b.x - figure.center_of_mass.x,
                  b.y - figure.center_of_mass.y,
                  b.z - figure.center_of_mass.z);
        glEnd();
    }
}

void Display(void)
{
    glClearColor(0.2, 0.2, 0.2, 1); glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glEnable(GL_NORMALIZE);

    // Включение буфера
    if (buff_depth == 1) {
        glEnable(GL_DEPTH_TEST); // включили Тест глубины
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    }
    else glDisable(GL_DEPTH_TEST);

    // Включение двойного буфера
    if(double_buff == 1)
        glutInitDisplayMode(GLUT_DOUBLE);
    else
        glutInitDisplayMode(GLUT_SINGLE);

    // Текстурирование
    if (texture == 1)
    {
        unsigned char data54[54]; // служебный заголовок 54 байта

```

```

        if (file != NULL) { // Подгрузка текстуры, если файл открыт (после поток выво-
да из файла зануляется P.S. Все равно, что текстура уже загружена)
            fread(data54, 1, 54, file);
            int size = *(data54 + 10);

            int width = *(data54 + 18);
            int height = *(data54 + 22);
            unsigned char* Pixels = new unsigned char[Width * Height * 3];

            fread(Pixels, Width * Height * 3, 1, file);
            fclose(file);
            file = NULL;
            int Type = GL_BGR_EXT;
            GLuint tex;
            glEnable(GL_TEXTURE_2D); //Разрешение отображения

текстур
            glGenTextures(1, &tex); //Генерация мас-
сива номеров текстур
            glBindTexture(GL_TEXTURE_2D, tex); //Выбор текущей текстуры по
номеру
            gluBuild2DMipmaps(GL_TEXTURE_2D, 3, width, height, GL_BGR_EXT,
GL_UNSIGNED_BYTE, Pixels); //Задание режима выравнивания
            glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE); //Задание
режима учета параметров материала
        }

        for (int i = 0; i < figure.platan.size(); i++) {
            if (figure.platan[i].points.size() > 2)
            {
                glBegin(GL_POLYGON);
                clear_matrix();
                create_neib_matrix(figure.platan[i]);

                if (figure.platan[i].sequence.size() == 0) // Если не закэширована
"правильная" последовательность точек, то кэшируем
                {
                    figure.platan[i].sequence.push_back(0);
                    step(figure.platan[i].sequence, figure.platan[i], 0, 0);
                }

                int k;

                section help;
                return_text_coord(help, figure.platan[i]); // Считаем текстурные
координаты

                for (int z = 0; z < figure.platan[i].sequence.size(); z++)
                {
                    //glColor3f(1 - float(i) / figure.platan.size(), 1 -
float(i) / figure.platan.size(), 1); // Черный
                    glColor3f(0.7, 0.7, 0.7);
                    // Надо придумать перевод в двумерную точку
                    glTexCoord2f(help.points[z].x, help.points[z].y);
                    glVer-
tex3d(figure.platan[i].points[figure.platan[i].sequence[z]].x - figure.center_of_mass.x,
figure.platan[i].points[figure.platan[i].sequence[z]].y - figure.center_of_mass.y,
figure.platan[i].points[figure.platan[i].sequence[z]].z - figure.center_of_mass.z);
                }

                glEnd();
            }
        }
    }
}

```

```

    }
    else
    {
        // Просто вывод полигонов
        for (int i = 0; i < figure.platan.size(); i++) {

            if (figure.platan[i].points.size() > 2)
            {
                glBegin(GL_POLYGON);
                clear_matrix();
                create_neib_matrix(figure.platan[i]);

                if (figure.platan[i].sequence.size() == 0) // Если не закэширована
"правильная" последовательность точек, то кэшируем
                {
                    figure.platan[i].sequence.push_back(0);
                    step(figure.platan[i].sequence, figure.platan[i], 0, 0);
                }

                int k;

                for (int z = 0; z < figure.platan[i].sequence.size(); z++)
                {
                    glColor3f(1 - float(i) / figure.platan.size(), 1 -
float(i) / figure.platan.size(), 1); // Черный
                    glVertex3d(figure.platan[i].points[figure.platan[i].sequence[z]].x - figure.center_of_mass.x,
figure.platan[i].points[figure.platan[i].sequence[z]].y - figure.center_of_mass.y,
figure.platan[i].points[figure.platan[i].sequence[z]].z - figure.center_of_mass.z);
                }

                glEnd();
            }
            tex_2d = 0;
        }

        // Включение света
        if (light == 1)
        {
            glEnable(GL_NORMALIZE);
            glEnable(GL_LIGHTING);
            glEnable(GL_LIGHT0); // !!!! В зависимости от типа света нужно менять аргумент
        }
        else
        {
            glDisable(GL_LIGHT0); // !!!! В зависимости от типа света нужно менять аргу-
МЕНТ
            glDisable(GL_LIGHTING);
        }

        if (show_normals == 1)
            show_normals_func(figure);

        if (carcass == 1) {

            glDisable(GL_DEPTH_TEST);
            // Рисуем каркас
            for (int i = 0; i < figure.platan.size(); i++) {
                for (int k = 0; k < figure.platan[i].ribs.size(); k++) {

```

```

        glBegin(GL_LINES); // Просто линия
        glLineWidth(3000);
        glColor3f(0.0f, 0.0f, 0.0f); // Черный
        glVertex3d(figure.platan[i].ribs[k].a.x - figure.center_of_mass.x
            , figure.platan[i].ribs[k].a.y - figure.center_of_mass.y
            , figure.platan[i].ribs[k].a.z - figure.center_of_mass.z);
        glVertex3d(figure.platan[i].ribs[k].b.x - figure.center_of_mass.x
            , figure.platan[i].ribs[k].b.y - figure.center_of_mass.y
            , figure.platan[i].ribs[k].b.z - figure.center_of_mass.z);
        glEnd();
    }
}
//
if (smooth_normal)
    Draw_smooth_normals();

glFinish();

glutSwapBuffers(); // Для двойного буфера
}

void Reshape(GLint w, GLint h) //Функция изменения размеров окна
{
    Display();
    point_3d spect;
    Width = w; Height = h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, w, 0, h, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    Width = w;
    Height = h;

    //Вычисляем соотношение между шириной и высотой, предотвратив деление на 0
    if (Height == 0) Height = 1;
    float ratio = 1. * Width / Height;

    glViewport(0, 0, w, h); //Определяем окно просмотра
    glMatrixMode(GL_PROJECTION); //Устанавливаем матрицу проекции
    glLoadIdentity(); //Загружаем единичную матрицу

    //Перспективная проекция
    if (ortho_or_perspective == 1)
        gluPerspective(perspective_angle, perspective_ratio, 0.1f, 100.0f);
    //Ортографическая проекция
    else
    {
        double aspect = Width / double(Height);
        if (Width >= Height) {
            gluOrtho2D(-1. * aspect, 1. * aspect, -1., 1.);
        }
        else {
            gluOrtho2D(-1., 1., -1. / aspect, 1. / aspect);
        }
    }

    spect = Sphere_to_Dekart(SystemCoordinate);
    gluLookAt(spect.x, spect.y, spect.z,
        0, 0, 0, 0, 1, 0);
    glMatrixMode(GL_MODELVIEW); //Возврат к матрице модели

```

```

        glLoadIdentity();
    }

    enum keys
    {
        KeyW, KeyA, KeyS, KeyD,
        KeyZ, KeyX, KeyC, KeyV, KeyB, KeyN,
        KeyO, KeyP, Empty
    };

    void Keyboard(unsigned char key, int xx, int yy)
    {
        // Свет
        if (key == 'l') {
            light = !light;
        }

        //Каркасный режим
        if (key == 'z') {
            carcass = !carcass;
        }

        //Наложение текстуры
        if (key == 'x') {
            texture = !texture;
        }

        if (key == 'c') //Показ нормалей
            show_normals = !show_normals;

        if (key == 'v') //Сглаженные нормали
            smooth_normal = !smooth_normal;

        if (key == 'i') //Глубинный тест
            buff_depth = !buff_depth;

        if (key == 'p') //Перспективная проекция
        {
            ortho_or_perspective = 1;
        }

        if (key == 'o') //Ортогографическая проекция
        {
            ortho_or_perspective = 0;
        }

        if (key == 'w') {
            if (SystemCoordinate.Theta + 5 <= 180)
                SystemCoordinate.Theta += 5;
        }

        if (key == 's') {
            if (SystemCoordinate.Theta - 5 > 0)
                SystemCoordinate.Theta -= 5;
        }

        if (key == 'a') {
            if (SystemCoordinate.Alpha + 5 < 360)
                SystemCoordinate.Alpha += 5;
            else
                SystemCoordinate.Alpha = 0;
        }

        if (key == 'd') {

```

```

        if (SystemCoordinate.Alpha - 5 >= 0)
            SystemCoordinate.Alpha -= 5;
        else
            SystemCoordinate.Alpha = 360;
    }

    if (key == '-') {
        SystemCoordinate.Radius -= 0.01;
    }

    if (key == '=') {
        SystemCoordinate.Radius += 0.01;
    }

    Reshape(Width, Height);
}

void Menu(int pos)
{
    int key = (keys)pos;

    switch (key)
    {
        case KeyW: Keyboard('w', 0, 0); break;
        case KeyS: Keyboard('s', 0, 0); break;
        case KeyA: Keyboard('a', 0, 0); break;
        case KeyD: Keyboard('d', 0, 0); break;

        case KeyO: Keyboard('o', 0, 0); break;
        case KeyP: Keyboard('p', 0, 0); break;

        case KeyZ: Keyboard('z', 0, 0); break;
        case KeyX: Keyboard('x', 0, 0); break;
        case KeyC: Keyboard('c', 0, 0); break;
        case KeyV: Keyboard('v', 0, 0); break;
        case KeyB: Keyboard('b', 0, 0); break;

        default:
            int menu_projection = glutCreateMenu(Menu);
            glutAddMenuEntry("Перспективная проекция (P)", KeyP);
            glutAddMenuEntry("Ортографическая проекция (O)", KeyO);

            int menu_show = glutCreateMenu(Menu);
            glutAddMenuEntry("Каркасный режим (Z)", KeyZ);
            glutAddMenuEntry("Наложение текстуры (X)", KeyX);
            glutAddMenuEntry("Показ нормалей (C)", KeyC);
            glutAddMenuEntry("Показ сглаженных нормалей (V)", KeyV);

            glutAttachMenu(GLUT_RIGHT_BUTTON);
            Keyboard(Empty, 0, 0);
    }
}

int main(int argc, char* argv[])
{
    int choose_task;
    if (read_func(figure.mass_point, choose_task) != 1) {
        return -1;
    }
    init_figure(figure); // Надо улучшить вектор нормали?
}

```

```

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB);
    glutInitWindowSize(Width, Height);
    glutCreateWindow("Текущий цвет всех точек:");
    Menu(Empty);
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Keyboard);
    glutMainLoop();
    return 1;
}

```

## Файл library.h:

```

#include <iostream>
#include <vector>

#define eps 1e-7
using namespace std;
typedef float type_coordinates;

typedef struct sk {
    type_coordinates shift_x, shift_y, shift_z; // Сдвиги
    type_coordinates value_scale; // Масштабирование
    type_coordinates alpha, betha; // Вращение

    sk() {
        this->shift_x = 0;
        this->shift_y = 0;
        this->shift_z = 0;
        this->alpha = 0;
        this->betha = 0;

        this->value_scale = 1;
    }

} system_coord;

system_coord SystemCoordinates;

typedef struct Point2D
{
    type_coordinates x, y;

    Point2D(type_coordinates x_, type_coordinates y_) {
        this->x = x_;
        this->y = y_;
    }
} point_2d;

typedef struct Point3D //трехмерные точки
{
    type_coordinates x, y, z;

    Point3D() {
        this->x = 0;
        this->y = 0;
        this->z = 0;
    }
}

```

```

Point3D(type_coordinates x_, type_coordinates y_, type_coordinates z_) {
    this->x = x_;
    this->y = y_;
    this->z = z_;
}

template <typename T>
Point3D operator/(T size) {
    this->x /= size;
    this->y /= size;
    this->z /= size;
    return *this;
}

Point3D operator-(Point3D a) {
    Point3D k(this->x - a.x, this->y - a.y, this->z - a.z);
    return k;
}

Point3D operator+(Point3D z)
{
    Point3D k;
    k.x = this->x + z.x;
    k.y = this->y + z.y;
    k.z = this->z + z.z;
    return k;
}

bool operator==(Point3D z) {
    if (this->x == z.x && this->y == z.y)
        return true;
    return false;
}
} point_3d;

typedef struct carcass_obj {
    Point3D a, b;
    bool is_intersection;

    carcass_obj(Point3D z, Point3D f) {
        this->a = z;
        this->b = f;
        this->is_intersection = 0;
    }

    carcass_obj() {
        this->is_intersection = 0;
    }

    carcass_obj operator=(carcass_obj f) {
        this->a = f.a;
        this->b = f.b;
        this->is_intersection = f.is_intersection;
        return *this;
    }

    bool operator==(carcass_obj k) {
        if (this->a == k.a && this->b == k.b)
            return true;

        if (this->b == k.a && this->a == k.b)
            return true;

        return false;
    }
}

```

```

    }
} carcass_obj;

typedef struct section {
    vector<point_3d> points;
    vector<carcass_obj> ribs;
    point_3d vec_normal;
    vector<int> sequence;
} section;

typedef struct object
{
    vector<point_3d> mass_point;
    vector<section> platan;
    point_3d center_of_mass;
} Object;

typedef struct vector_ {
    type_coordinates x, y, z;

    vector_(point_3d b, point_3d a) {
        this->x = a.x - b.x;
        this->y = a.y - b.y;
        this->z = a.z - b.z;
    }
} vector_;

carcass_obj create_line(Point3D a, Point3D b)
{
    carcass_obj k(a, b);
    return k;
}

point_3d vec_mult(point_3d a, point_3d b)
{
    point_3d k(a.y * b.z - b.y * a.z, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x);
    return k;
}

point_3d vec_mult(vector_ a, vector_ b)
{
    point_3d k(a.y * b.z - b.y * a.z, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x);
    return k;
}

point_3d get_vec_normal(section& sc)
{
    point_3d sup, vec_1, vec_2;

    // Инициализация двух векторов плоскости
    if (sc.points.size() > 2)
    {
        vec_1 = sc.points[1] - sc.points[0];
        vec_2 = sc.points[2] - sc.points[0];
    }

    sup = vec_mult(vec_1, vec_2); // здесь надо продумать

    type_coordinates z = sqrt(pow(sup.x, 2) + pow(sup.y, 2) + pow(sup.z, 2));
    sup = sup / z;

    return sup;
}

void init_ribs(section& sk) {

```

```

    point_3d point_1, point_2;

    for (int j = 0; j < sk.points.size(); j++)
    {
        point_1 = sk.points[j];
        for (int k = j + 1; k < sk.points.size(); k++)
        {
            point_2 = sk.points[k];
            sk.ribs.push_back(create_line(point_1, point_2));
        }
    }
}

bool have_common_line(object fig, int i, int j, int& index_edge_i, int& index_edge_j)
{
    int count_i = fig.platan[i].ribs.size(), count_j = fig.platan[j].ribs.size();
    for (index_edge_i = 0; index_edge_i < count_i; index_edge_i++)
        for (index_edge_j = 0; index_edge_j < count_j; index_edge_j++)
            if (fig.platan[i].ribs[index_edge_i] ==
fig.platan[j].ribs[index_edge_j])
                return 1;

    return 0;
}

bool have_platan_the_point(section s, Point3D p) {
    for (int i = 0; i < s.points.size(); i++)
        if (p == s.points[i])
            return true;
    return false;
}

type_coordinates norma_vec(Point3D K) {
    return sqrt(pow(K.x, 2) + pow(K.y, 2) + pow(K.z, 2));
}

bool are_they_parallel(object fig, int i, int j) {
    if (norma_vec(fig.platan[i].vec_normal - fig.platan[j].vec_normal) < eps)
        return true;

    if (norma_vec(fig.platan[i].vec_normal + fig.platan[j].vec_normal) < eps)
        return true;

    return false;
}

bool same_sign(type_coordinates a, type_coordinates b)
{
    if (a > 0 && b > 0 || a < 0 && b < 0 || a == 0 && b == 0)
        return true;
    return false;
}

bool check_intersection(carcass_obj a, carcass_obj b) {
    if (a.a == b.a || a.a == b.b
        || a.b == b.a || a.b == b.b)
        return false; // либо исходят из одной точки, либо вообще совпадают

    point_3d A(a.a), B(a.b), C(b.a), D(b.b), f, g;
    vector_ AB(A, B), AC(A, C), AD(A, D);

    f = vec_mult(AB, AD);

```

```

    g = vec_mult(AC, AB);

    if (same_sign(f.x, g.x) && same_sign(f.y, g.y) && same_sign(f.z, g.z))
        return true;

    return false;
}

int init_figure(object& fig)
{
    // Ищем центр масс
    point_3d sup(0, 0, 0);
    int count_points = fig.mass_point.size();
    for (int i = 0; i < count_points; i++)
        sup = sup + fig.mass_point[i];
    sup = sup / count_points;
    fig.center_of_mass = sup;
    //

    // Считаем всевозможные сечения
    section plat;
    point_3d point_1, point_2, point_3;
    for (int i = 0; i < count_points; i++)
        for (int j = i + 1; j < count_points; j++)
            for (int k = j + 1; k < count_points; k++)
            {
                point_1 = fig.mass_point[i];
                point_2 = fig.mass_point[j];
                point_3 = fig.mass_point[k];

                // Инициализация Сечения
                plat.points.push_back(point_1);
                plat.points.push_back(point_2);
                plat.points.push_back(point_3);
                plat.vec_normal = get_vec_normal(plat);
                //

                fig.platan.push_back(plat);
                plat.points.clear();
            }

    //

    // Инициализируем ребра для плоскостей
    for (int i = 0; i < fig.platan.size(); i++) // выбираем грань
        init_ribs(fig.platan[i]);
    //

    // Ищем плоскости, которые имеют общую грань и параллельны

    int indx_1, indx_2; // Индексы для определения общих граней

    bool flag = false; // Если никаких изменений не произошло, тогда заканчиваем внешний
    цикл, так как ещё раз перебирать бесполезно
    point_3d point_not_general;
    for (int z = 0; z < fig.platan.size(); z++) {
        for (int i = 0; i < fig.platan.size(); i++) // Ходим по плоскостям N^3 раз
            (Чтобы наверняка)
                for (int j = i + 1; j < fig.platan.size(); j++) // Ходим по плоскостям
                    if (have_common_line(fig, i, j, indx_1, indx_2) &&
are_they_parallel(fig, i, j))
                        {

```

```

        // добавление точек в i сечение из j
        for (int i_1 = 0; i_1 < fig.platan[j].points.size();
i_1++)
            fig.platan[i].points.push_back(fig.platan[j].points[i_1]);

        // Удаляем дубликаты точек в i сечении
        for (int i_1 = 0; i_1 < fig.platan[i].points.size();
i_1++)
            for (int j_1 = i_1 + 1; j_1 <
fig.platan[i].points.size(); j_1++)
                if (fig.platan[i].points[i_1] ==
fig.platan[i].points[j_1])
                    fig.platan[i].points.erase(fig.platan[i].points.begin() + j_1);

        fig.platan[i].ribs.clear();
        init_ribs(fig.platan[i]);
        // Мы обязаны проверять, какая точка находится на прямой и
удалить её ???

        // Удаляем ненужную плоскости
        fig.platan.erase(fig.platan.begin() + j);
    }
}
//

stack<carcass_obj> Stack_;

// Помечаем для каждой плоскости пересечения граней
for (int i = 0; i < fig.platan.size(); i++)
    for (int j = 0; j < fig.platan[i].ribs.size(); j++)
        for (int k = j + 1; k < fig.platan[i].ribs.size(); k++)
            if (check_intersection(fig.platan[i].ribs[j],
fig.platan[i].ribs[k]))
                {
                    Stack_.push(fig.platan[i].ribs[j]);
                    Stack_.push(fig.platan[i].ribs[k]);
                }

carcass_obj Z;
while (Stack_.size() != 0) {
    Z = Stack_.top();
    Stack_.pop();

    for (int i = 0; i < fig.platan.size(); i++)
        for (int j = 0; j < fig.platan[i].ribs.size(); j++)
            if (fig.platan[i].ribs[j] == Z)
                fig.platan[i].ribs.erase(fig.platan[i].ribs.begin() + j);
}
//

// Теперь удаляем лишние плоскости
for (int i = 0; i < fig.platan.size(); i++)
    if (fig.platan[i].points.size() != fig.platan[i].ribs.size())
    {
        fig.platan.erase(fig.platan.begin() + i);
        i--;
    }
//
return 1;
}

```

```

int read_func(vector<point_3d>& mas_3d, int& choose)
{
    type_coordinates x_, y_, z_, shift;
    int count;
    ifstream in_pointer("info.txt");

    if (in_pointer.is_open() == false)
        return -1;

    in_pointer >> choose;
    in_pointer >> count;
    ifstream fin;

    switch (choose)
    {
    case 0: // 2D-координаты вершин сечения (считающегося выпуклым)

        fin.open("coordinates_2d.txt");

        if (fin.is_open() == true) {
            fin >> shift;

            for (int i = 0; i < count; i++) {

                fin >> x_ >> y_;
                point_3d k(x_, y_, 0);
                mas_3d.push_back(k);
                point_3d z(x_, y_, shift);
                mas_3d.push_back(z);

            }
        }
        else
            return -1;

        break;
    case 1: // 3D-координаты траектории тиражирования
        fin.open("coordinates_3d.txt");
        if (fin.is_open()) {
            for (int i = 0; i < count; i++) {
                fin >> x_ >> y_ >> z_;
                point_3d k(x_, y_, z_);
                mas_3d.push_back(k);

            }
        }
        else
            return -1;

        break;
    default:
        return -1;
    }

    return 1;
}

```

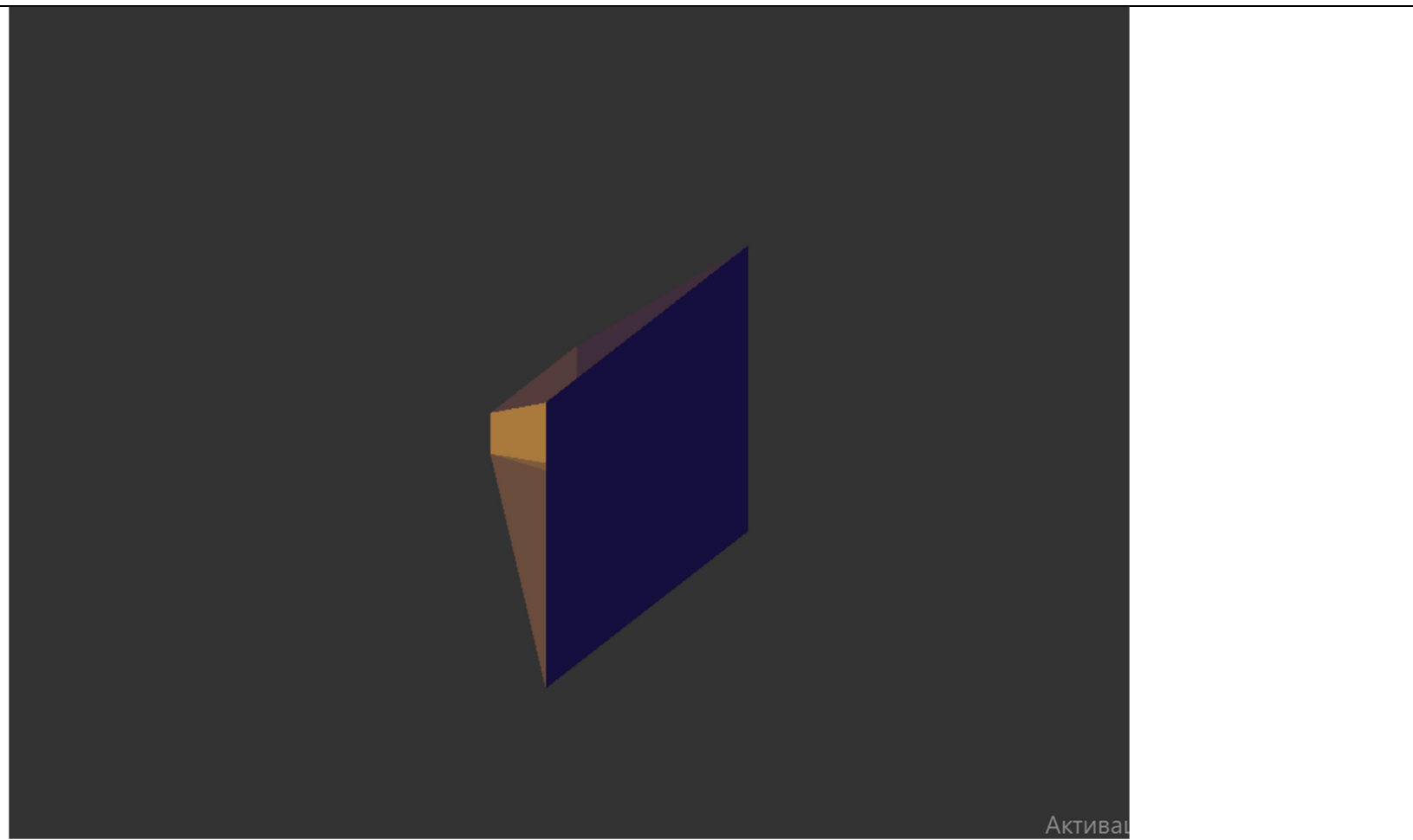
## 5) Руководство пользователя

Клавиша	Назначение
wasd	Управление камерой (наблюдателем)
«р» и «о»	Включение перспективной и ортогографической проекций соответственно
+	Приближение вида в ортогографической проекции
-	Отдаление вида в ортогографической проекции
z	Включение/выключение каркасного режима. По умолчанию выключен.
c	Включение/выключение демонстрации нормалей. По умолчанию выключена.
v	Включение/выключение сглаженных нормалей
l	Включение/выключение освещения. По умолчанию освещение отключено.
x	Включение/выключение текстурирования объекта. По умолчанию выключено.
i	Включение/выключение теста глубины

## 6) Тестирование программы

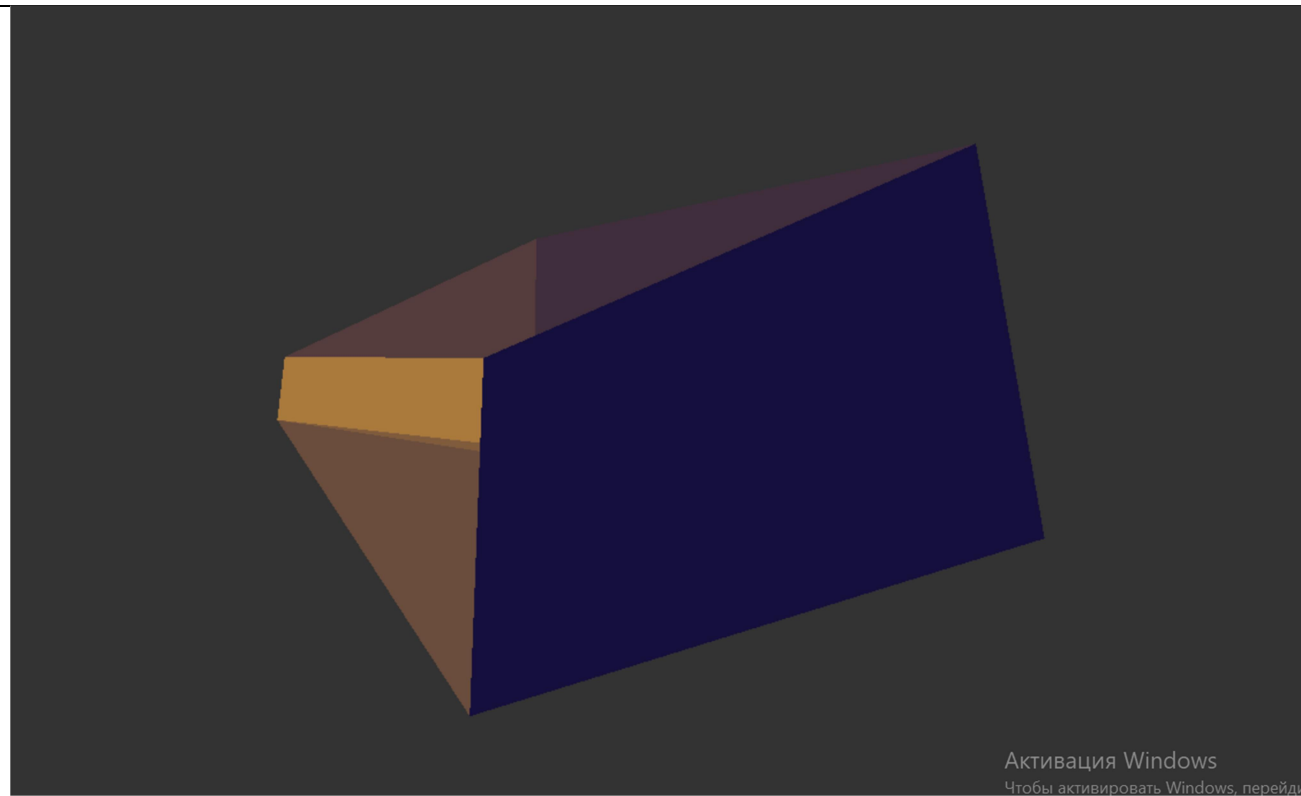
Для фигуры с 8 вершинами

Запуск программы

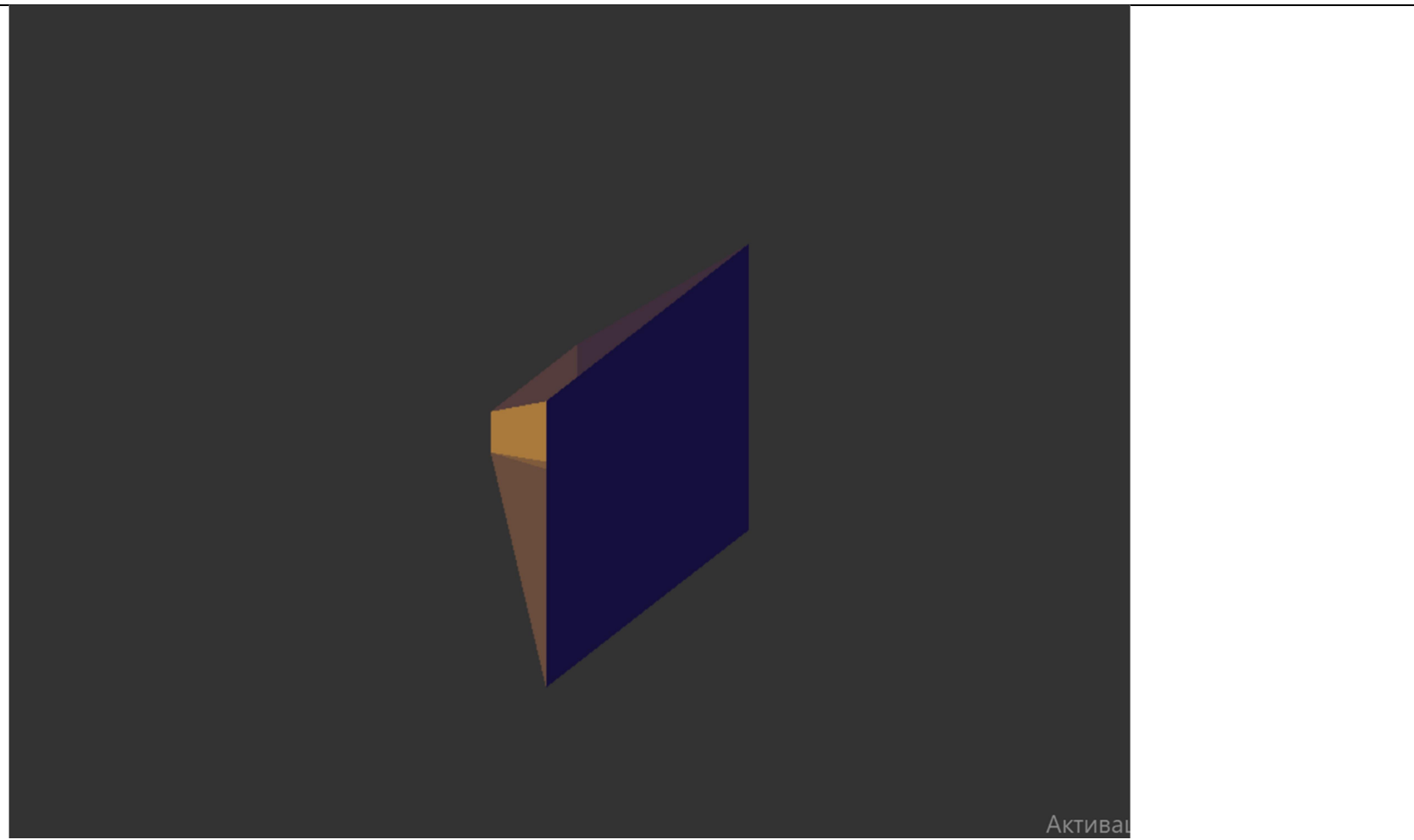


Активация

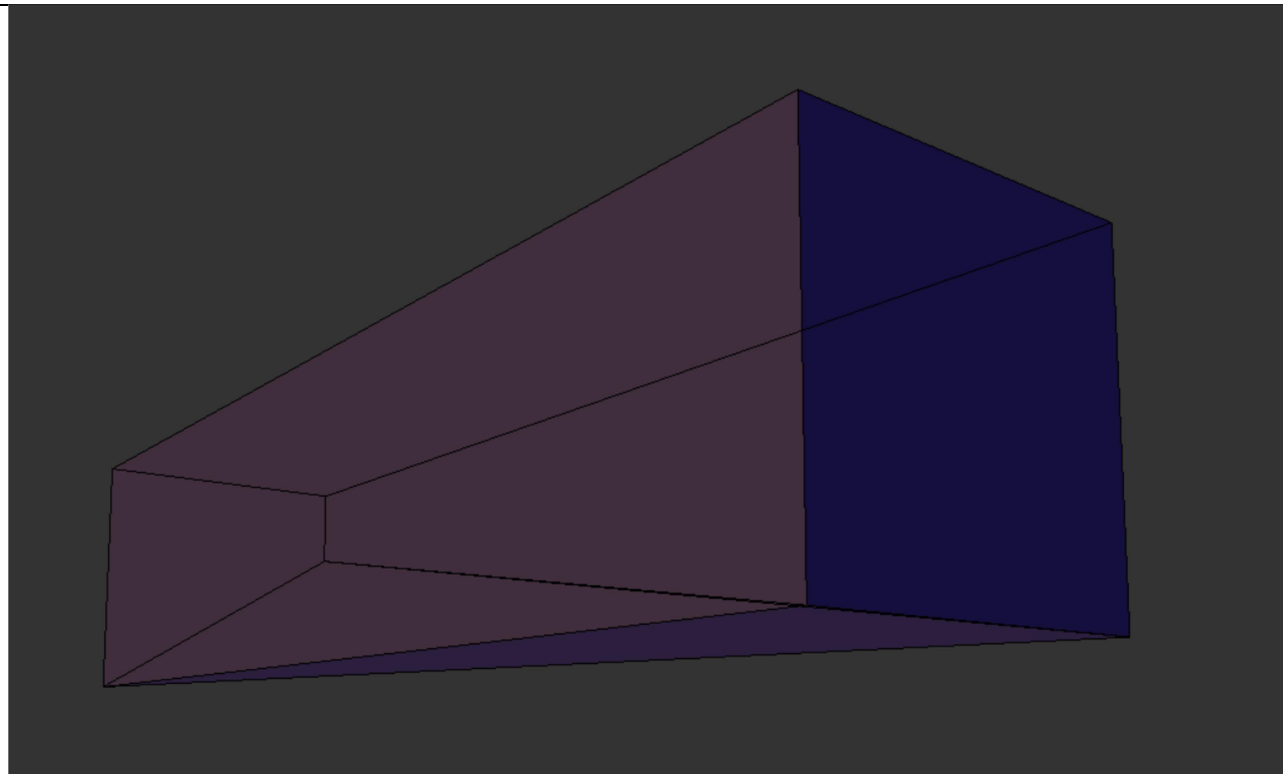
Фигура в перспективной  
проекции



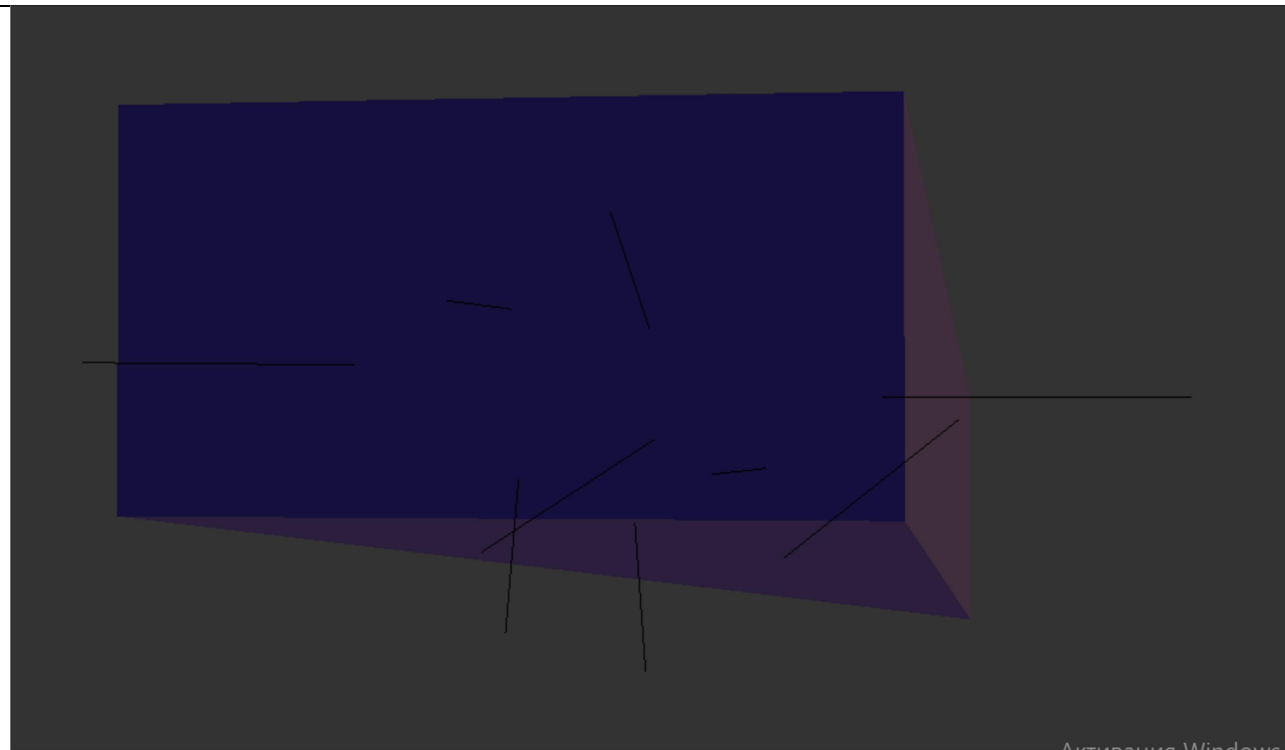
Фигура в ортографической проекции



Включить каркасный режим



Включить демонстрацию  
нормалей

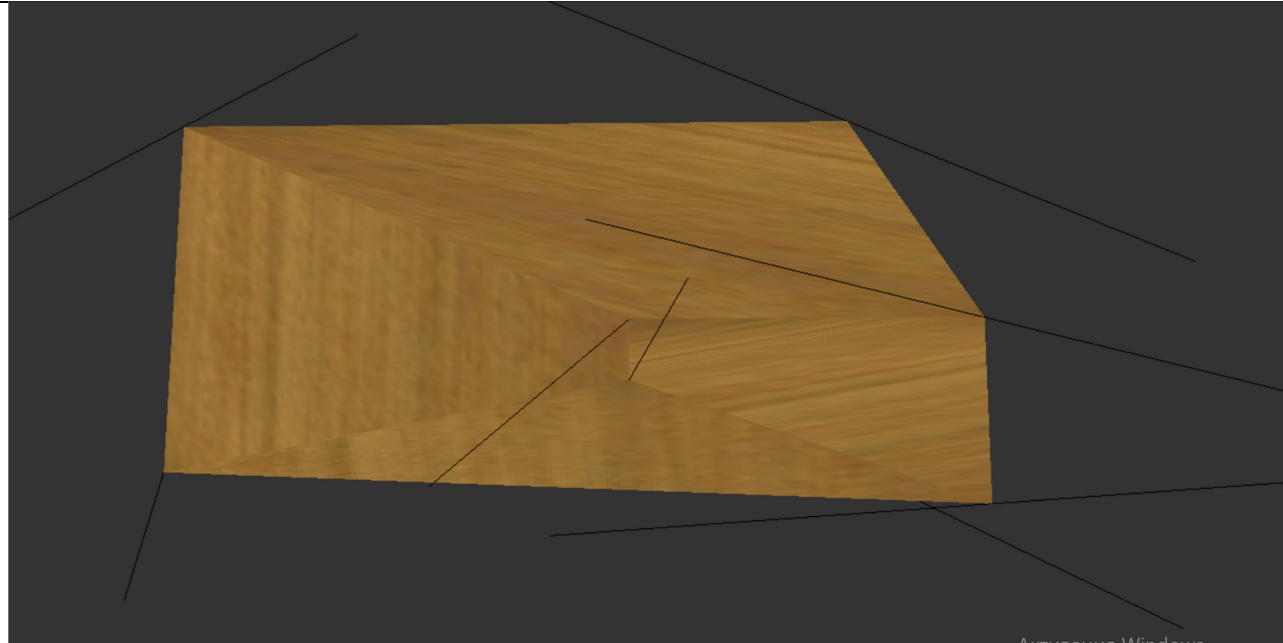


Отключить каркасный режим. Включить текстурирование. Включить z - буфер

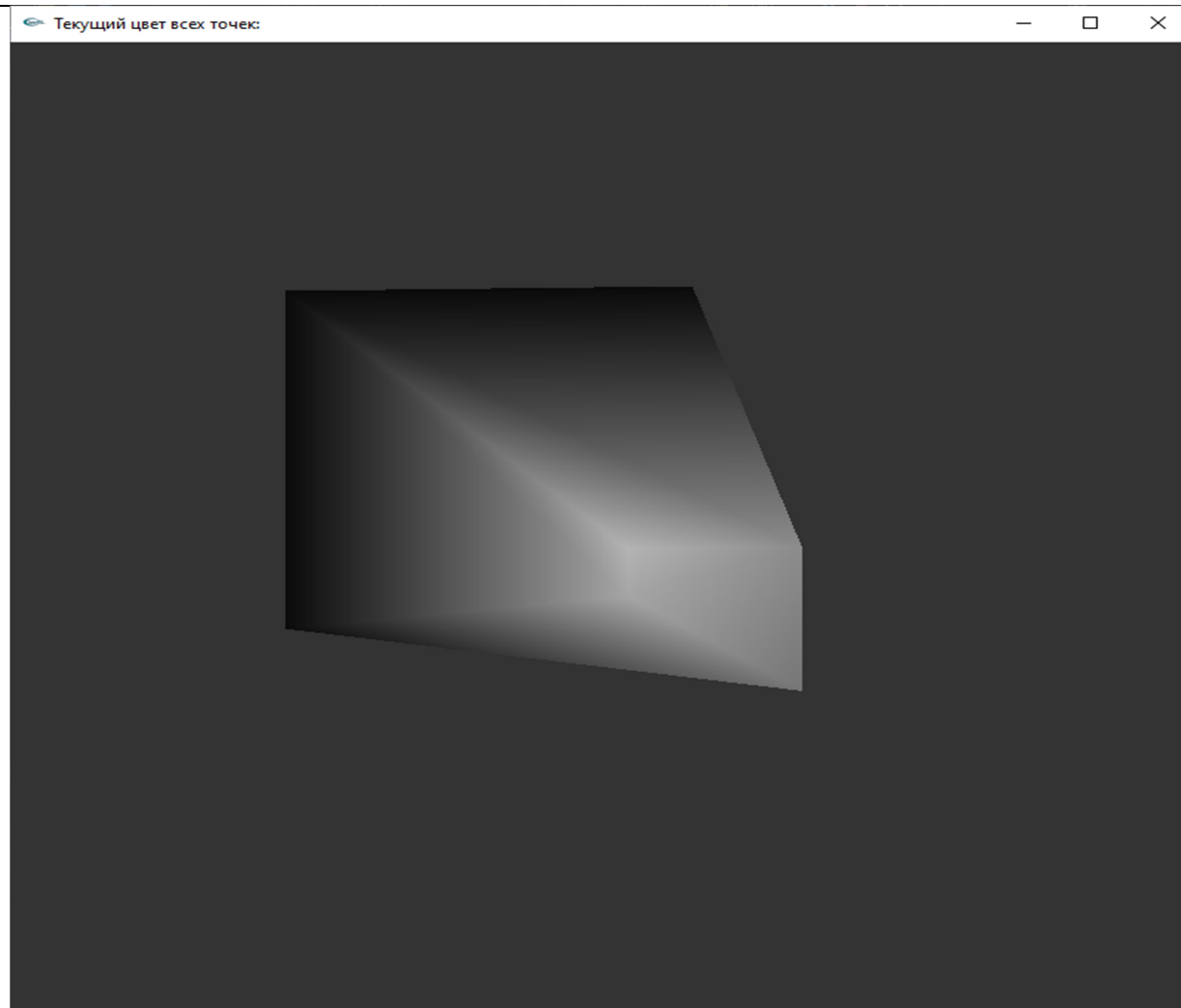


Включить сглаживание нормалей

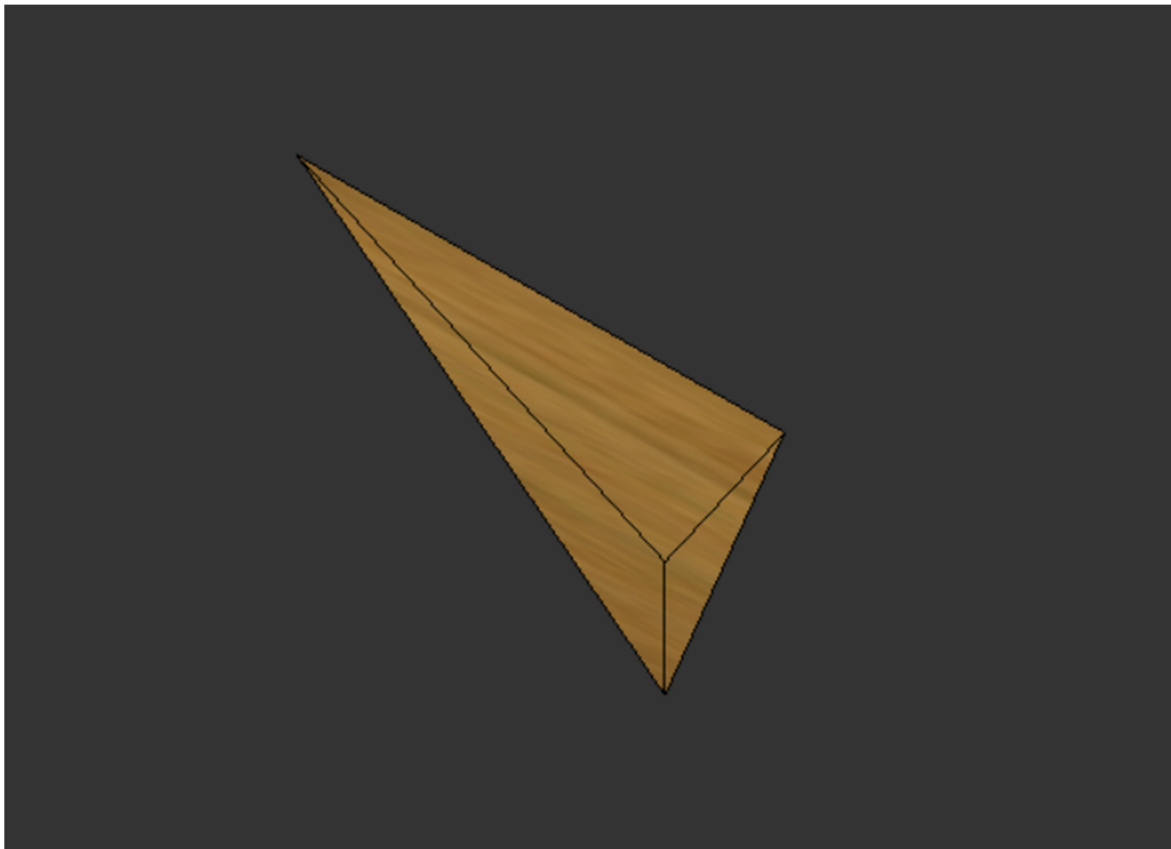
Замечание: здесь мы можем видеть одну из проблем, связанную с подсчетом нормали (нормаль есть векторное произведение векторов, которые образуют плоскость, и не получилось брать точки поверхностей в таком порядке, чтобы вектор был всегда направлен из фигуры, а не вовнутрь, как это было с несколькими поверхностями)



Включение света + от-  
ключение текстур



Фигура с 4 вершинами



Фигура с 5 вершинами

