**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ТВ**

**отчЁт**

**по лабораторной работе № 3**

**по дисциплине «Компьютерный синтез трехмерных изображений»**

**Тема: АЛГОРИТМЫ УДАЛЕНИЯ НЕВИДИМЫХ РЕБЕР И ГРАНЕЙ. ПРОСТЕЙШИЕ АЛГОРИТМЫ ЗАКРАСКИ**

**Вариант 6**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9105 |  | Шаривзянов Д. Р. |
| Преподаватель |  | Сирый Р. С. |

Санкт-Петербург

2024

ИССЛЕДОВАНИЕ ГЕОМЕТРИЧЕСКИХ ПРЕОБРАЗОВАНИЙ В ТРЕХМЕРНОМ ПРОСТРАНСТВЕ (3D ПРЕОБРАЗОВАНИЙ) И МЕХАНИЗМОВ ПРОЕЦИРОВАНИЯ

**Цель работы:**

Целью лабораторной работы является знакомство с преобразованиями в 3D пространстве и механизмами проецирования.

**Исходные данные:**

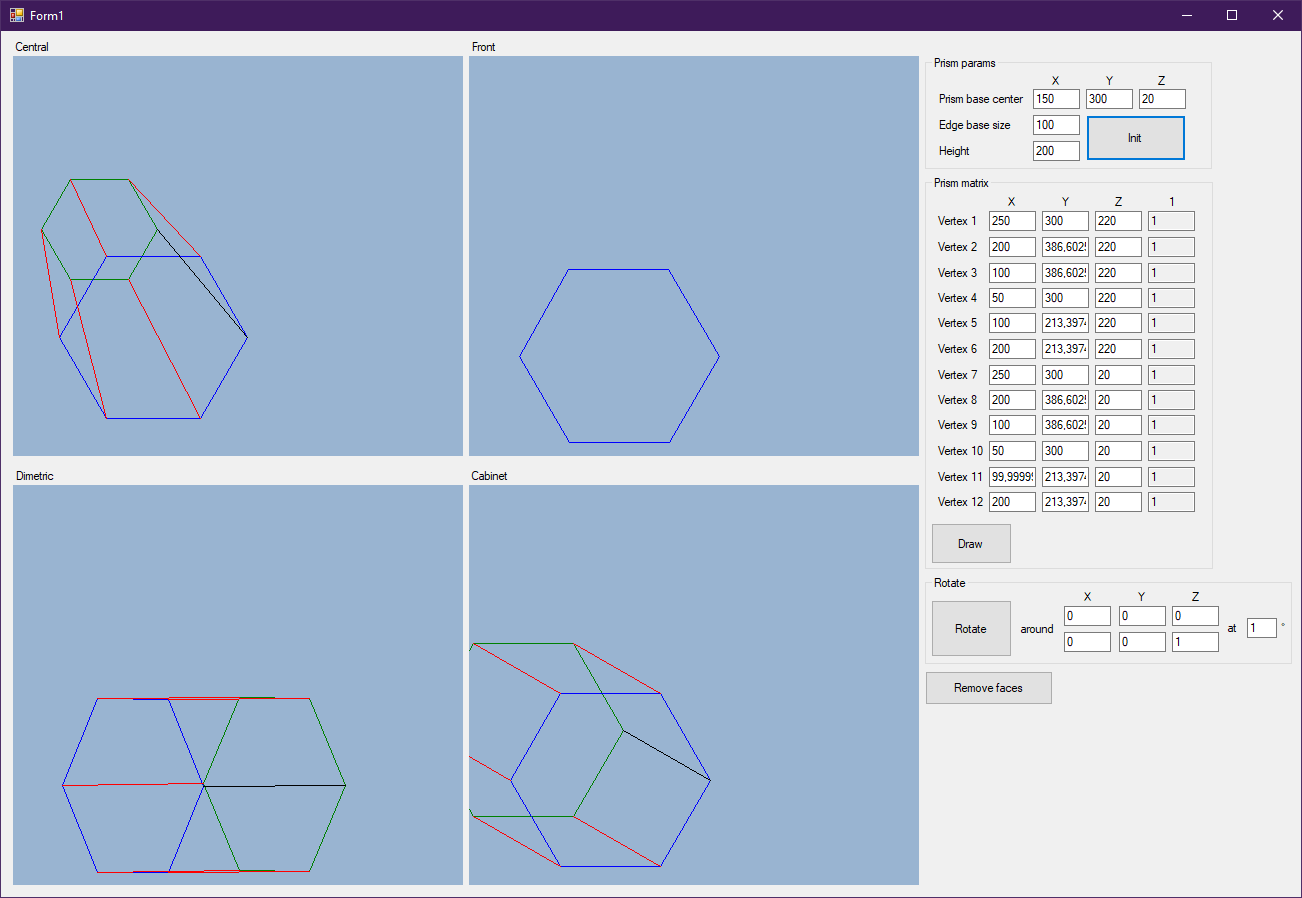


Рис 1. Исходный вид формы.

**Код программы:**

int Roberts[8] = { 0 };

private: System::Int32 defineAngle(int vertex\_0, int vertex\_1, int vertex\_2, int V[][dimension - 1]) {

int Nrm[3], R[3];

// вычисление вектора нормали по трем точкам, принадлежащим грани

Nrm[0] = (((V[vertex\_1][1] - V[vertex\_0][1]) \* (V[vertex\_2][2] - V[vertex\_1][2])) - ((V[vertex\_2][1] - V[vertex\_1][1]) \* (V[vertex\_1][2] - V[vertex\_0][2])));

Nrm[1] = (((V[vertex\_1][2] - V[vertex\_0][2]) \* (V[vertex\_2][0] - V[vertex\_1][0])) - ((V[vertex\_2][2] - V[vertex\_1][2]) \* (V[vertex\_1][0] - V[vertex\_0][0])));

Nrm[2] = (((V[vertex\_1][0] - V[vertex\_0][0]) \* (V[vertex\_2][1] - V[vertex\_1][1])) - ((V[vertex\_2][0] - V[vertex\_1][0]) \* (V[vertex\_1][1] - V[vertex\_0][1])));

R[0] = V[vertex\_0][0] - 0.0; // вектор, направленный на наблюдателя

R[1] = V[vertex\_0][1] - 0.0;

R[2] = V[vertex\_0][2] + d;

return (Nrm[0] \* R[0]) + (Nrm[1] \* R[1]) + (Nrm[2] \* R[2]); // скалярное произведение векторов

}

private: System::Void removeFaces() {

Roberts[0] = defineAngle(0, 1, 2, dek\_p); // 1 2 3 заднее основание

Roberts[1] = defineAngle(11, 6, 0, dek\_p); // 12 7 1 справа сверху

Roberts[2] = defineAngle(6, 7, 1, dek\_p); // 7 8 2 справа снизу

Roberts[3] = defineAngle(7, 8, 2, dek\_p); // 8 9 3 снизу

Roberts[4] = defineAngle(8, 9, 3, dek\_p); // 9 10 4 слева снизу

Roberts[5] = defineAngle(9, 10, 4, dek\_p); // 10 11 5 слева сверху

Roberts[6] = defineAngle(10, 11, 5, dek\_p); // 11 12 6 сверху

Roberts[7] = defineAngle(11, 10, 9, dek\_p); // 12 11 10 переднее основание

}

Обработчики событий изменения блоков рисования идентичны:

private: System::Void pictureBox\_front\_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e) {

int x1 = dek\_p[0][0];

int y1 = dek\_p[0][1];

int z1 = dek\_p[0][2];

int x2 = dek\_p[1][0];

int y2 = dek\_p[1][1];

int z2 = dek\_p[1][2];

int x3 = dek\_p[2][0];

int y3 = dek\_p[2][1];

int z3 = dek\_p[2][2];

int x4 = dek\_p[3][0];

int y4 = dek\_p[3][1];

int z4 = dek\_p[3][2];

int x5 = dek\_p[4][0];

int y5 = dek\_p[4][1];

int z5 = dek\_p[4][2];

int x6 = dek\_p[5][0];

int y6 = dek\_p[5][1];

int z6 = dek\_p[5][2];

int x7 = dek\_p[6][0];

int y7 = dek\_p[6][1];

int z7 = dek\_p[6][2];

int x8 = dek\_p[7][0];

int y8 = dek\_p[7][1];

int z8 = dek\_p[7][2];

int x9 = dek\_p[8][0];

int y9 = dek\_p[8][1];

int z9 = dek\_p[8][2];

int x10 = dek\_p[9][0];

int y10 = dek\_p[9][1];

int z10 = dek\_p[9][2];

int x11 = dek\_p[10][0];

int y11 = dek\_p[10][1];

int z11 = dek\_p[10][2];

int x12 = dek\_p[11][0];

int y12 = dek\_p[11][1];

int z12 = dek\_p[11][2];

//----------------------------------------------рисование ребёр-----------------------------------------------------

bool edges[18] = { 0 };

int countZeros = 0;

for (int i = 0; i < sizeof(Roberts); i++) {

if (Roberts[i] == 0) countZeros++;

}

//заднее основание

if ((Roberts[0] <= 0) || (Roberts[1] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Green, x6, y6, x1, y1); { if (countZeros != sizeof(Roberts)) edges[0] = 1; } }

if ((Roberts[0] <= 0) || (Roberts[2] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Green, x1, y1, x2, y2); { if (countZeros != sizeof(Roberts)) edges[1] = 1; } }

if ((Roberts[0] <= 0) || (Roberts[3] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Green, x2, y2, x3, y3); { if (countZeros != sizeof(Roberts)) edges[2] = 1; } }

if ((Roberts[0] <= 0) || (Roberts[4] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Green, x3, y3, x4, y4); { if (countZeros != sizeof(Roberts)) edges[3] = 1; } }

if ((Roberts[0] <= 0) || (Roberts[5] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Green, x4, y4, x5, y5); { if (countZeros != sizeof(Roberts)) edges[4] = 1; } }

if ((Roberts[0] <= 0) || (Roberts[6] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Green, x5, y5, x6, y6); { if (countZeros != sizeof(Roberts)) edges[5] = 1; } }

//переднее основание

if ((Roberts[7] <= 0) || (Roberts[6] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Blue, x7, y7, x8, y8); { if (countZeros != sizeof(Roberts)) edges[6] = 1; } }

if ((Roberts[7] <= 0) || (Roberts[1] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Blue, x8, y8, x9, y9); { if (countZeros != sizeof(Roberts)) edges[7] = 1; } }

if ((Roberts[7] <= 0) || (Roberts[2] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Blue, x9, y9, x10, y10); { if (countZeros != sizeof(Roberts)) edges[8] = 1; } }

if ((Roberts[7] <= 0) || (Roberts[3] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Blue, x10, y10, x11, y11); { if (countZeros != sizeof(Roberts)) edges[9] = 1; } }

if ((Roberts[7] <= 0) || (Roberts[4] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Blue, x11, y11, x12, y12); { if (countZeros != sizeof(Roberts)) edges[10] = 1; } }

if ((Roberts[7] <= 0) || (Roberts[5] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Blue, x12, y12, x7, y7); { if (countZeros != sizeof(Roberts)) edges[11] = 1; } }

//рёбры между основаниями (высоты)

if ((Roberts[1] <= 0) || (Roberts[2] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Black, x1, y1, x7, y7); { if (countZeros != sizeof(Roberts)) edges[12] = 1; } }

if ((Roberts[2] <= 0) || (Roberts[3] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Red, x2, y2, x8, y8); { if (countZeros != sizeof(Roberts)) edges[13] = 1; } }

if ((Roberts[3] <= 0) || (Roberts[4] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Red, x3, y3, x9, y9); { if (countZeros != sizeof(Roberts)) edges[14] = 1; } }

if ((Roberts[4] <= 0) || (Roberts[5] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Red, x4, y4, x10, y10); { if (countZeros != sizeof(Roberts)) edges[15] = 1; } }

if ((Roberts[5] <= 0) || (Roberts[6] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Red, x5, y5, x11, y11); { if (countZeros != sizeof(Roberts)) edges[16] = 1; } }

if ((Roberts[6] <= 0) || (Roberts[1] <= 0)) { e->Graphics->DrawLine(System::Drawing::Pens::Red, x6, y6, x12, y12); { if (countZeros != sizeof(Roberts)) edges[17] = 1; } }

//----------------------------------------------заливка граней-----------------------------------------------------

// Координаты вершин задней грани

array<System::Drawing::PointF>^ points\_backBase = gcnew array<System::Drawing::PointF>(6);

points\_backBase[0] = System::Drawing::PointF(x1, y1);

points\_backBase[1] = System::Drawing::PointF(x2, y2);

points\_backBase[2] = System::Drawing::PointF(x3, y3);

points\_backBase[3] = System::Drawing::PointF(x4, y4);

points\_backBase[4] = System::Drawing::PointF(x5, y5);

points\_backBase[5] = System::Drawing::PointF(x6, y6);

// Создаем кисть для заливки

System::Drawing::SolidBrush^ brush\_backBase = gcnew System::Drawing::SolidBrush(System::Drawing::Color::Green);

// Заливаем грань

if (edges[0] && edges[1] && edges[2] && edges[3] && edges[4] && edges[5]) e->Graphics->FillPolygon(brush\_backBase, points\_backBase);

// Координаты вершин передней грани

array<System::Drawing::PointF>^ points\_frontBase = gcnew array<System::Drawing::PointF>(6);

points\_frontBase[0] = System::Drawing::PointF(x7, y7);

points\_frontBase[1] = System::Drawing::PointF(x8, y8);

points\_frontBase[2] = System::Drawing::PointF(x9, y9);

points\_frontBase[3] = System::Drawing::PointF(x10, y10);

points\_frontBase[4] = System::Drawing::PointF(x11, y11);

points\_frontBase[5] = System::Drawing::PointF(x12, y12);

// Создаем кисть для заливки

System::Drawing::SolidBrush^ brush\_frontBase = gcnew System::Drawing::SolidBrush(System::Drawing::Color::Blue);

// Заливаем грань

if (edges[6] && edges[7] && edges[8] && edges[9] && edges[10] && edges[11]) e->Graphics->FillPolygon(brush\_frontBase, points\_frontBase);

// Создаем кисть для заливки боковых граней

System::Drawing::SolidBrush^ brush\_Base = gcnew System::Drawing::SolidBrush(System::Drawing::Color::Red);

// Координаты вершин грани 1

array<System::Drawing::PointF>^ points\_Base1 = gcnew array<System::Drawing::PointF>(4);

points\_Base1[0] = System::Drawing::PointF(x1, y1);

points\_Base1[1] = System::Drawing::PointF(x6, y6);

points\_Base1[2] = System::Drawing::PointF(x12, y12);

points\_Base1[3] = System::Drawing::PointF(x7, y7);

// Заливаем грань

if (edges[0] && edges[11] && edges[17] && edges[12]) e->Graphics->FillPolygon(brush\_Base, points\_Base1);

// Координаты вершин грани 2

array<System::Drawing::PointF>^ points\_Base2 = gcnew array<System::Drawing::PointF>(4);

points\_Base2[0] = System::Drawing::PointF(x1, y1);

points\_Base2[1] = System::Drawing::PointF(x7, y7);

points\_Base2[2] = System::Drawing::PointF(x8, y8);

points\_Base2[3] = System::Drawing::PointF(x2, y2);

// Заливаем грань

if (edges[1] && edges[6] && edges[12] && edges[13]) e->Graphics->FillPolygon(brush\_Base, points\_Base2);

// Координаты вершин грани 3

array<System::Drawing::PointF>^ points\_Base3 = gcnew array<System::Drawing::PointF>(4);

points\_Base3[0] = System::Drawing::PointF(x2, y2);

points\_Base3[1] = System::Drawing::PointF(x8, y8);

points\_Base3[2] = System::Drawing::PointF(x9, y9);

points\_Base3[3] = System::Drawing::PointF(x3, y3);

// Заливаем грань

if (edges[2] && edges[7] && edges[13] && edges[14]) e->Graphics->FillPolygon(brush\_Base, points\_Base3);

// Координаты вершин грани 4

array<System::Drawing::PointF>^ points\_Base4 = gcnew array<System::Drawing::PointF>(4);

points\_Base4[0] = System::Drawing::PointF(x3, y3);

points\_Base4[1] = System::Drawing::PointF(x9, y9);

points\_Base4[2] = System::Drawing::PointF(x10, y10);

points\_Base4[3] = System::Drawing::PointF(x4, y4);

// Заливаем грань

if (edges[3] && edges[8] && edges[14] && edges[15]) e->Graphics->FillPolygon(brush\_Base, points\_Base4);

// Координаты вершин грани 5

array<System::Drawing::PointF>^ points\_Base5 = gcnew array<System::Drawing::PointF>(4);

points\_Base5[0] = System::Drawing::PointF(x4, y4);

points\_Base5[1] = System::Drawing::PointF(x10, y10);

points\_Base5[2] = System::Drawing::PointF(x11, y11);

points\_Base5[3] = System::Drawing::PointF(x5, y5);

// Заливаем грань

if (edges[4] && edges[9] && edges[15] && edges[16]) e->Graphics->FillPolygon(brush\_Base, points\_Base5);

// Координаты вершин грани 6

array<System::Drawing::PointF>^ points\_Base6 = gcnew array<System::Drawing::PointF>(4);

points\_Base6[0] = System::Drawing::PointF(x5, y5);

points\_Base6[1] = System::Drawing::PointF(x11, y11);

points\_Base6[2] = System::Drawing::PointF(x12, y12);

points\_Base6[3] = System::Drawing::PointF(x6, y6);

// Заливаем грань

if (edges[5] && edges[10] && edges[16] && edges[17]) e->Graphics->FillPolygon(brush\_Base, points\_Base6);

//---------------------------------------------------------------------------------------------------------------

}

private: System::Void btn\_removeFaces\_Click(System::Object^ sender, System::EventArgs^ e) {

// служебный массив для хранения результата преобразования

double Result[vert\_num][dimension] = { 0 };

// проведение центрального проецирования

matrix\_mult(vert\_num, hmg\_p, matrix\_Center, Result);

hmg2dek(vert\_num, Result, dek\_p);

// заполнение массива результатами скалярного произведения нормали к // грани и вектора, направленного на наблюдателя.

removeFaces();

// вызов обработчика отрисовки

pictureBox\_central->Refresh();

// проведение проецирования сверху

matrix\_mult(vert\_num, hmg\_p, matrix\_Front, Result);

hmg2dek(vert\_num, Result, dek\_p);

// заполнение массива результатами скалярного произведения нормали к // грани и вектора, направленного на наблюдателя.

removeFaces();

// вызов обработчика отрисовки

pictureBox\_front->Refresh();

// проведение проецирования диметрии

matrix\_mult(vert\_num, hmg\_p, matrix\_Dimetry, Result);

hmg2dek(vert\_num, Result, dek\_p);

// заполнение массива результатами скалярного произведения нормали к // грани и вектора, направленного на наблюдателя.

removeFaces();

// вызов обработчика отрисовки

pictureBox\_dimetric->Refresh();

// проведение проецирования кабине

matrix\_mult(vert\_num, hmg\_p, matrix\_Cabinet, Result);

hmg2dek(vert\_num, Result, dek\_p);

// заполнение массива результатами скалярного произведения нормали к // грани и вектора, направленного на наблюдателя.

removeFaces();

// вызов обработчика отрисовки

pictureBox\_cabinet->Refresh();

}

Также были использованы переменные и функции из предыдущей лабораторной работы №2, а именно: psy, phi, alpha, d, vert\_num, dimension, hmg\_p, dek\_p, matrix\_Front, matrix\_Center, matrix\_Dimetry, matrix\_Cabinet, matrix\_mult, hmg2dek, btn\_draw\_Click, btn\_init\_Click, btn\_rotate\_Click

**Ход работы:**

1. Алгоритм удаления невидимых рёбер (Робертса)

Согласно алгоритму Робертса, для определения видимости грани нужно построить нормаль к поверхности грани и определить знак скалярного произведения этой нормали и вектора, направленного на наблюдателя. Для идентификации нормали и определения знака скалярного произведения предложена процедура, реализующая алгоритм, описанный в теоретической части пособия. В отличие от всех вышеописанных функций данная возвращает (return) значение типа integer, в котором, на самом деле, важен только знак.

Используя этот алгоритм (функцию defineAngle), перебираются все грани объекта и им присваивается значение (положительное – если они видимы, отрицательное – если нет). Далее, если две грани видны – то ребро между ними отображается, иначе – нет.

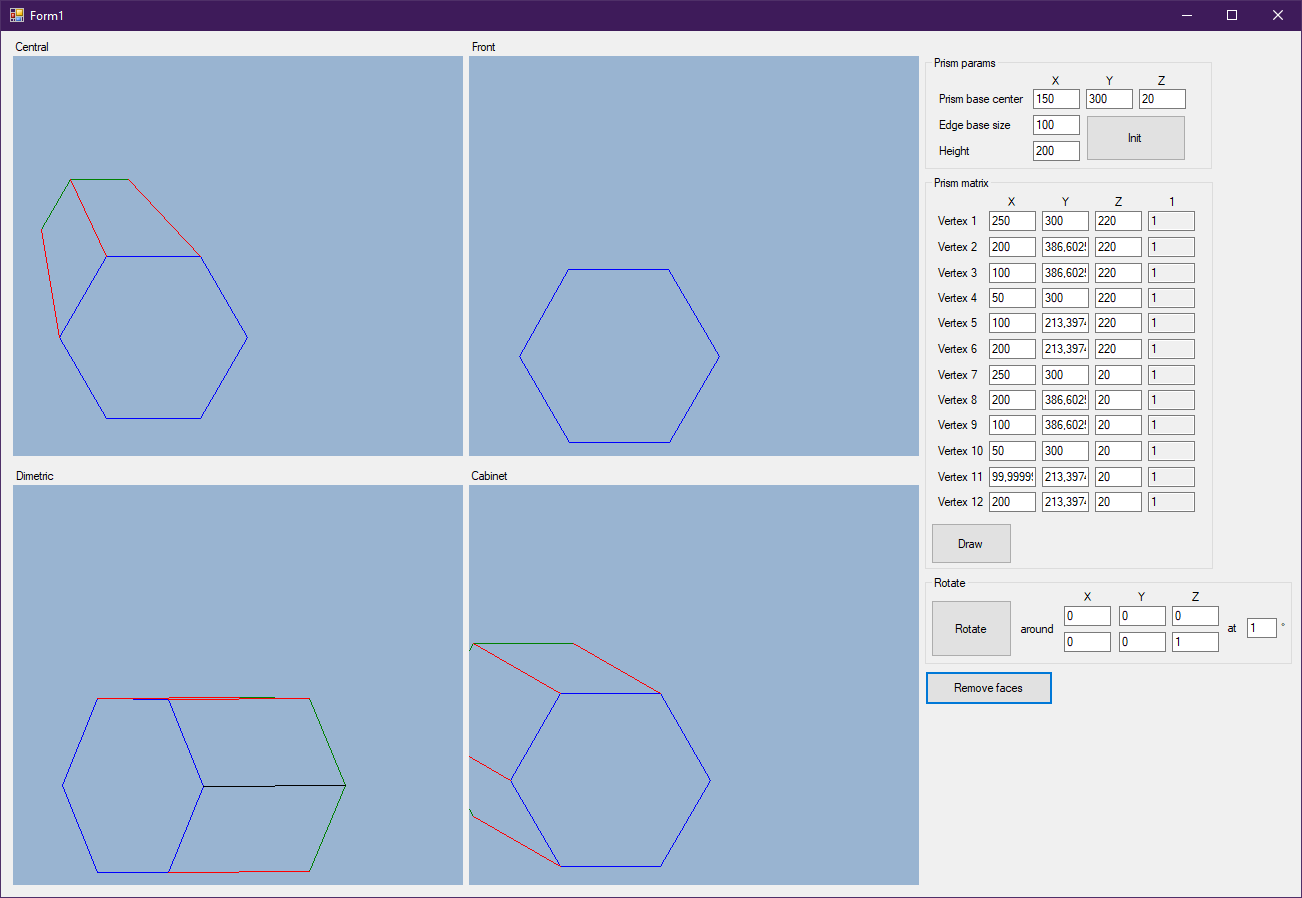


Рис. 2. Удаление нелицевых рёбер.

1. Алгоритм закраски видимых граней

На основе видимости рёбер можно судить о том, видна ли грань, описываемая этими рёбрами. Именно на основе этой логики основан алгоритм закраски грани.

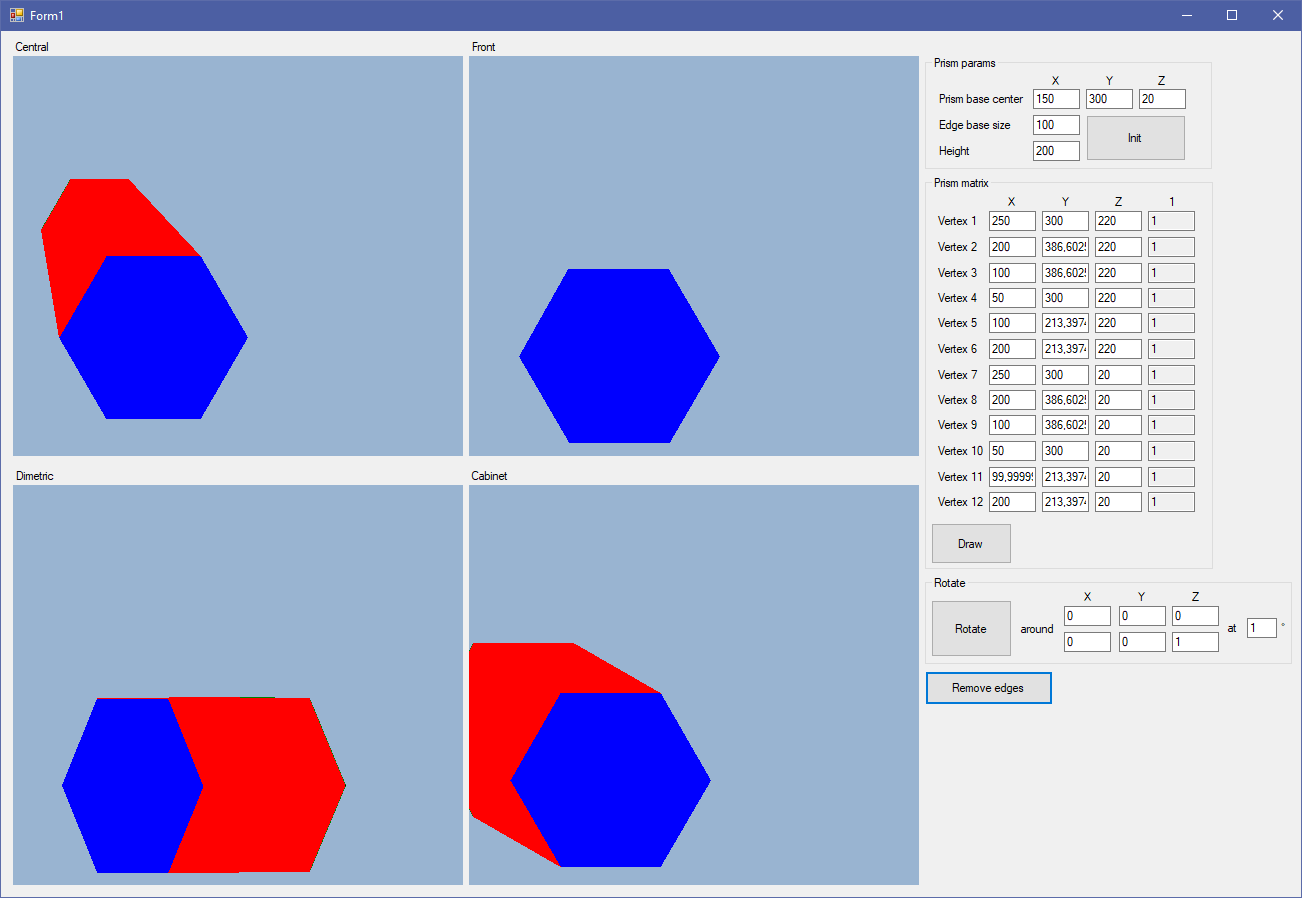


Рис. 3. Закраска граней.

**Выводы:**

Были изучены механизмы удаления невидимых рёбер (Робертса). Выяснилось, что при подборе номеров вершин следует помнить, что искомая нормаль к грани должна быть направлена наружу, относительно фигуры. Проверить это можно, например, по правилу буравчика для векторного произведения. Также была изучена и смоделирована логика закраски видимых граней.