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:

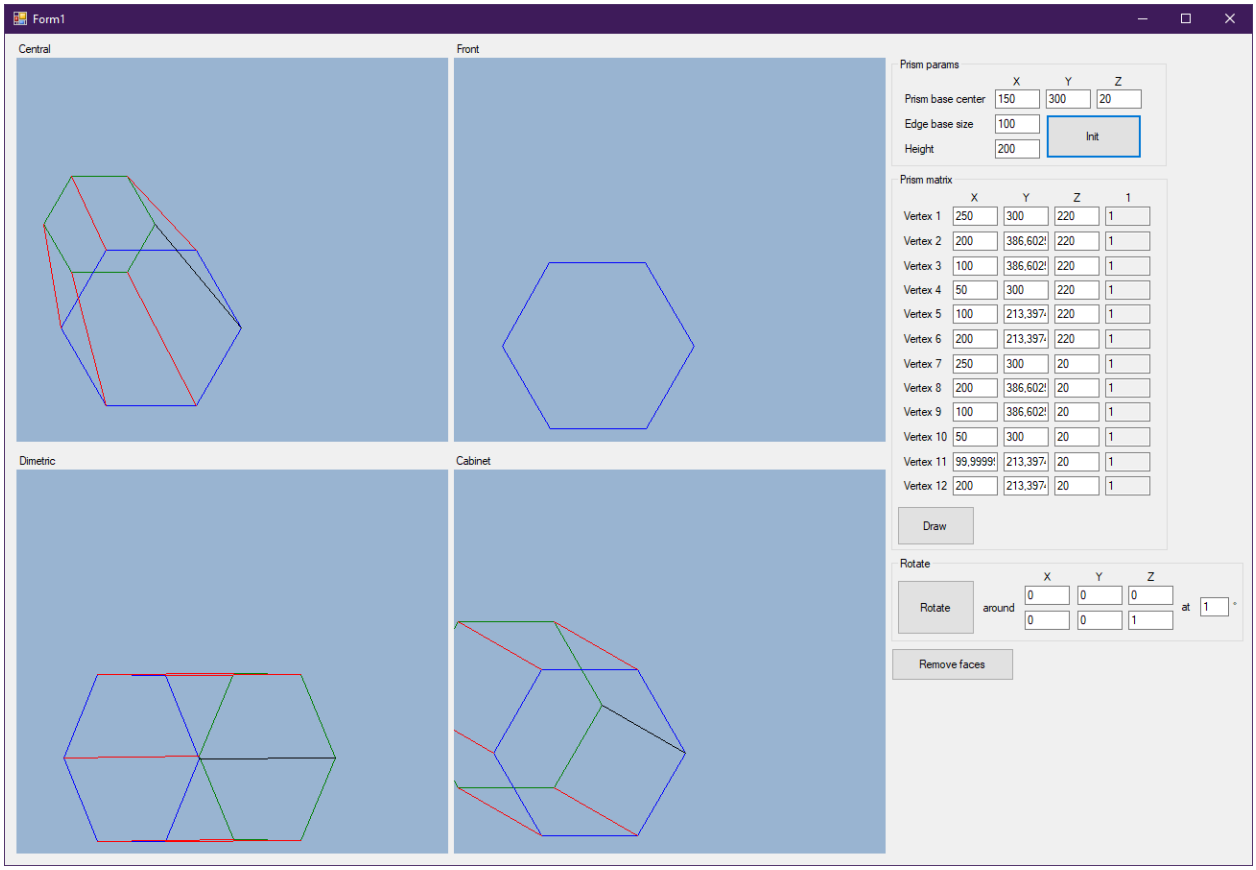

Рис 1. Исходный вид формы.

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.

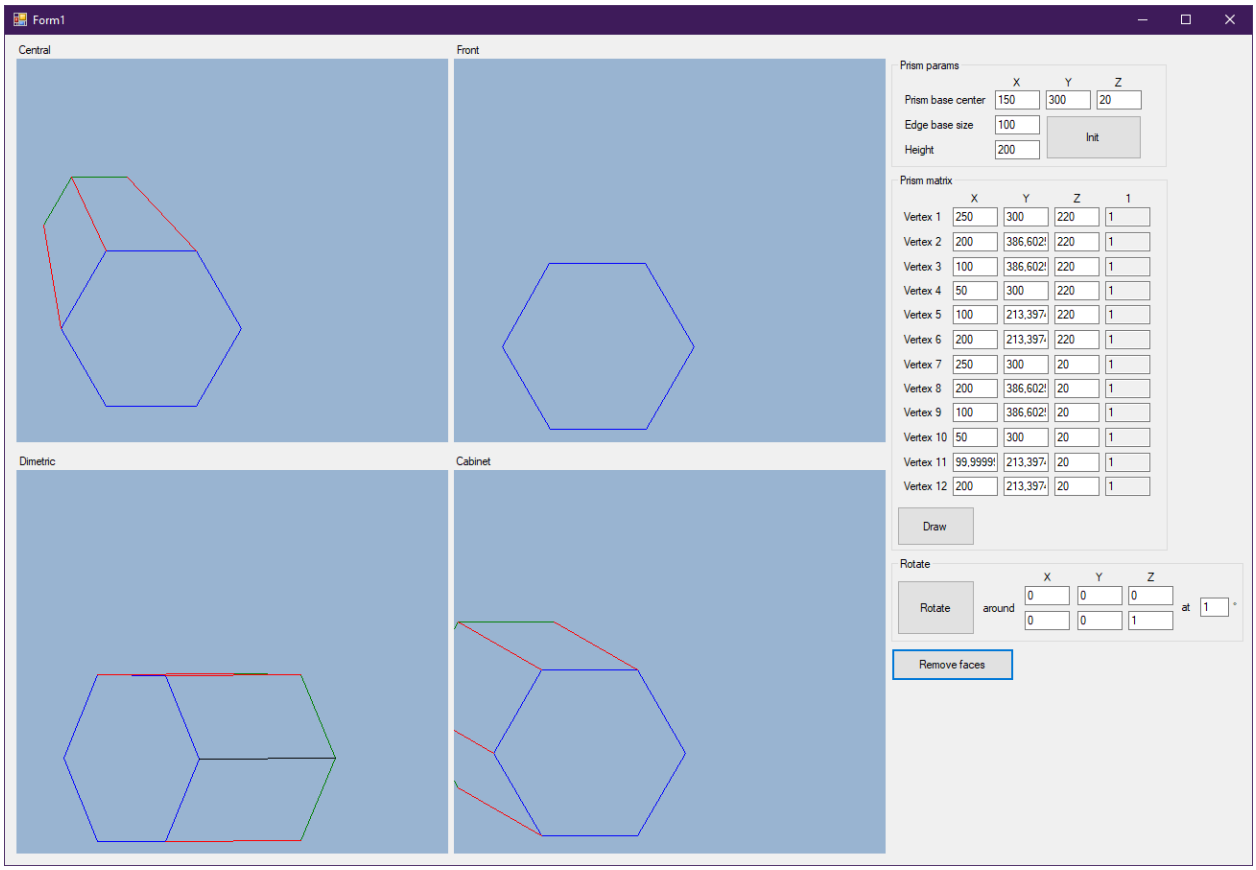

Рис. 2. Удаление нелицевых рёбер.

2. 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.

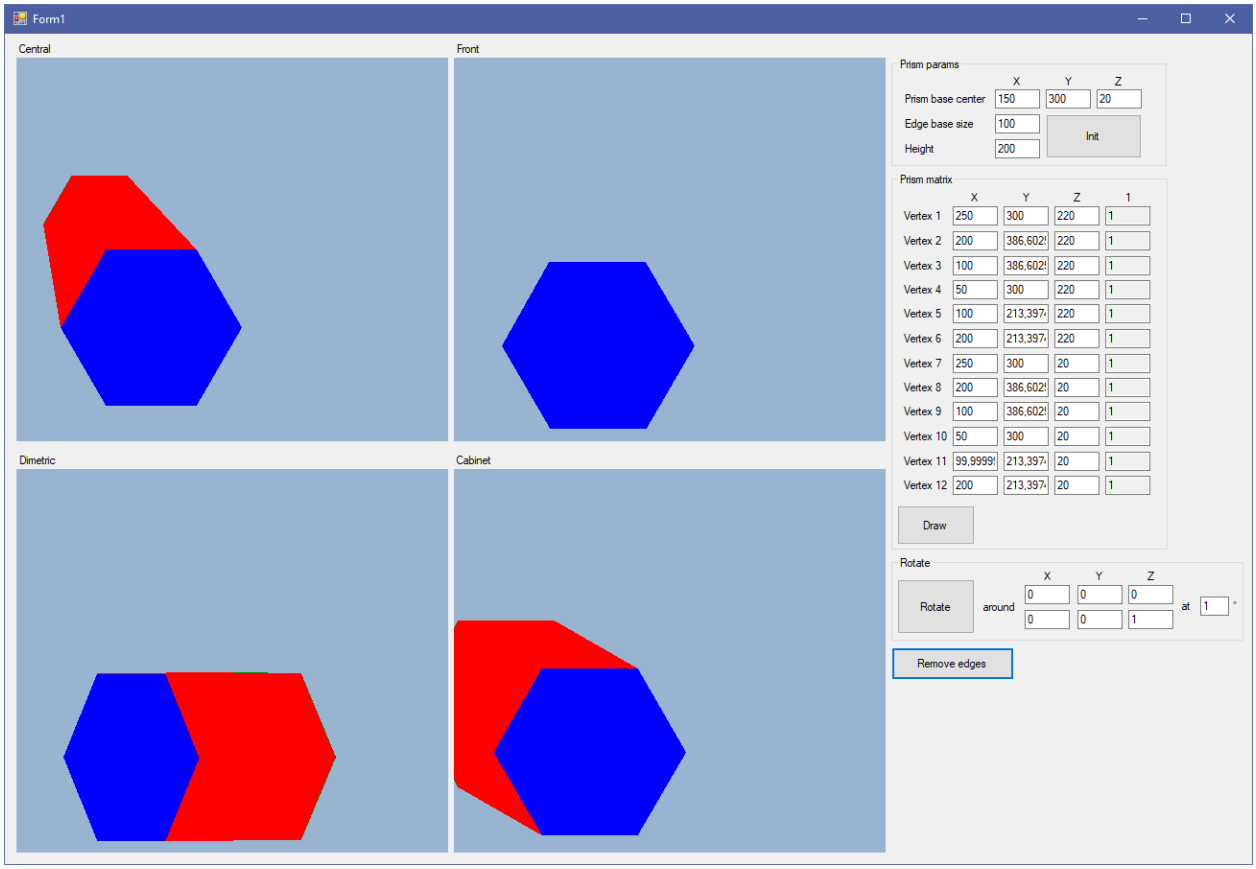

Рис. 3. Закраска граней.

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.