МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ТВ

ОТЧЁТ

по лабораторной работе № 2

по дисциплине «Компьютерный синтез трехмерных изображений»

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

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:

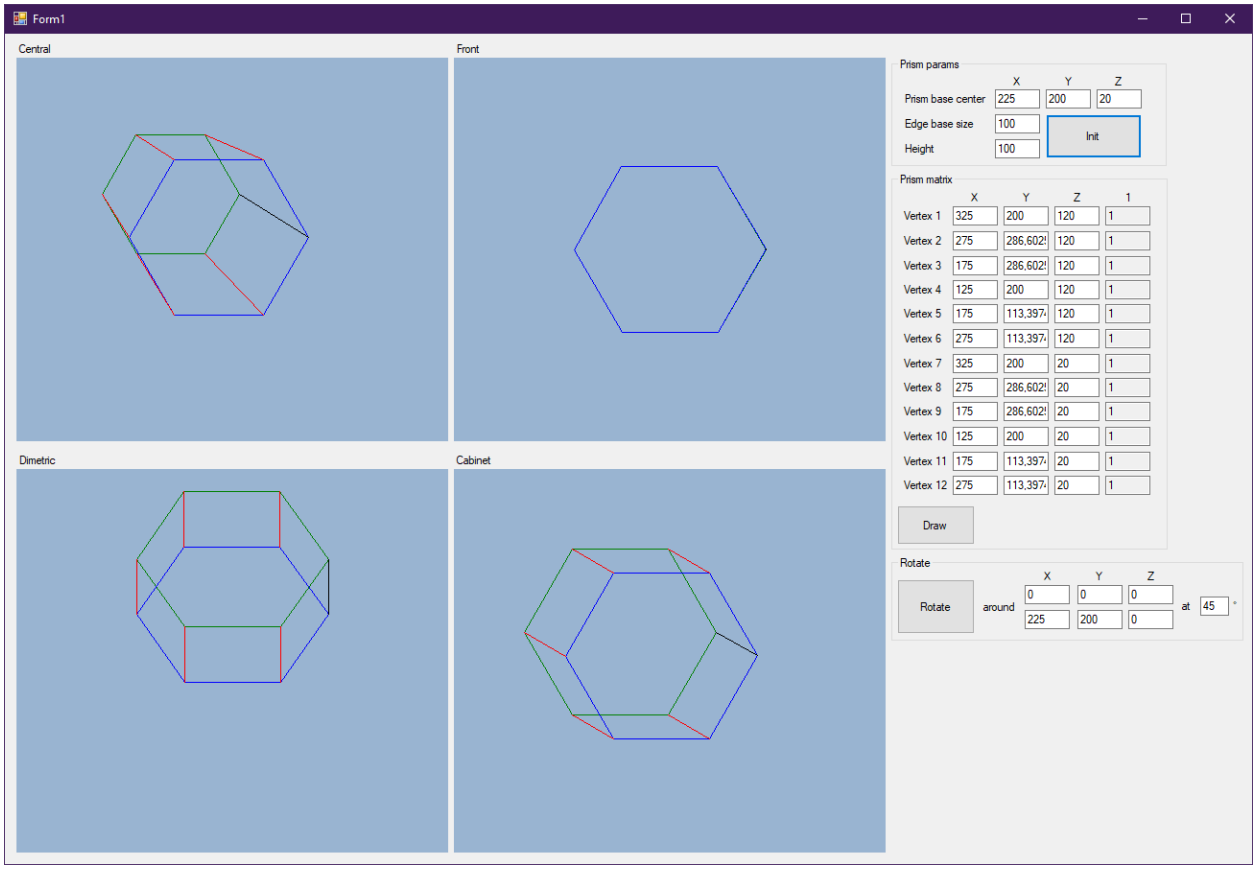
Целью лабораторной работы является знакомство с преобразованиями в 3D пространстве и механизмами проецирования.

Исходные данные:

Вариант	Вид многогранника	Типы проекции
6	Призма, в основании правильный шестиугольник	Центральная, спереди, диметрия, кабине

Вариант	Вид преобразования
6	Вращение многогранника вокруг произвольной прямой

Реализация формы программы:



Код программы:

```
#include <math.h>
#define deg2rad (acos(-1))/180

        // углы в радианах для проекций
        const double psy = 45.0 * deg2rad; // для димметрии
        const double phi = atan(sin(psy)) * deg2rad; // для димметрии
        const double alpha = 30.0 * deg2rad; // для Кавалье и Кабине

        // координата для центральной проекции
        const double d = 300;

        const int vert_num = 12; // количество вершин фигуры
        const int dimension = 4; // число столбцов матрицы однородных координат     //3-для двумерной графики, 4 для трехмерной

        //матрицы общие
        double hmg_p[vert_num][dimension]                    = { 0 }; // матрица однородных координат фигуры
        int          dek_p[vert_num][dimension - 1]          = { 0 }; // матрица экранных координат фигуры

        //матрицы проекций
        double matrix_Front[dimension][dimension] = // матрица вида спереди
        { 1, 0, 0, 0,
        0, 1, 0, 0,
        0, 0, 0, 0,
        0, 0, 0, 1 };

        double matrix_Center[dimension][dimension] = //центральная
        { 1, 0, 0, 0,
        0, 1, 0, 0,
        0, 0, 0, 1 / d,
        0, 0, 0, 1 };

        double matrix_Dimetry[dimension][dimension] =
        { cos(psy),                 sin(phi) * sin(psy),        0,          0,
          0,                        cos(phi),                   0,          0,
          sin(psy),                 -sin(phi) * cos(psy),       0,          0,
          0,                        0,                          0,          1 };

        double matrix_Cabinet[dimension][dimension] = {
        1,                  0,                  0,                  0,
        0,                  1,                  0,                  0,
        -0.5*cos(alpha),    -0.5*sin(alpha),    0,                  0,
        0,                  0,                  0,                  1 };
```

```
private: System::Void update_textBoxes(){
                textBox_x1->Text = Convert::ToString(hmg_p[0][0]);
                textBox_y1->Text = Convert::ToString(hmg_p[0][1]);
                textBox_z1->Text = Convert::ToString(hmg_p[0][2]);
                textBox_w1->Text = Convert::ToString(hmg_p[0][3]);

                textBox_x2->Text = Convert::ToString(hmg_p[1][0]);
                textBox_y2->Text = Convert::ToString(hmg_p[1][1]);
                textBox_z2->Text = Convert::ToString(hmg_p[1][2]);
                textBox_w2->Text = Convert::ToString(hmg_p[1][3]);

                textBox_x3->Text = Convert::ToString(hmg_p[2][0]);
                textBox_y3->Text = Convert::ToString(hmg_p[2][1]);
                textBox_z3->Text = Convert::ToString(hmg_p[2][2]);
                textBox_w3->Text = Convert::ToString(hmg_p[2][3]);
```

```
		textBox_x4->Text = Convert::ToString(hmg_p[3][0]);
		textBox_y4->Text = Convert::ToString(hmg_p[3][1]);
		textBox_z4->Text = Convert::ToString(hmg_p[3][2]);
		textBox_w4->Text = Convert::ToString(hmg_p[3][3]);

		textBox_x5->Text = Convert::ToString(hmg_p[4][0]);
		textBox_y5->Text = Convert::ToString(hmg_p[4][1]);
		textBox_z5->Text = Convert::ToString(hmg_p[4][2]);
		textBox_w5->Text = Convert::ToString(hmg_p[4][3]);

		textBox_x6->Text = Convert::ToString(hmg_p[5][0]);
		textBox_y6->Text = Convert::ToString(hmg_p[5][1]);
		textBox_z6->Text = Convert::ToString(hmg_p[5][2]);
		textBox_w6->Text = Convert::ToString(hmg_p[5][3]);

		textBox_x7->Text = Convert::ToString(hmg_p[6][0]);
		textBox_y7->Text = Convert::ToString(hmg_p[6][1]);
		textBox_z7->Text = Convert::ToString(hmg_p[6][2]);
		textBox_w7->Text = Convert::ToString(hmg_p[6][3]);

		textBox_x8->Text = Convert::ToString(hmg_p[7][0]);
		textBox_y8->Text = Convert::ToString(hmg_p[7][1]);
		textBox_z8->Text = Convert::ToString(hmg_p[7][2]);
		textBox_w8->Text = Convert::ToString(hmg_p[7][3]);

		textBox_x9->Text = Convert::ToString(hmg_p[8][0]);
		textBox_y9->Text = Convert::ToString(hmg_p[8][1]);
		textBox_z9->Text = Convert::ToString(hmg_p[8][2]);
		textBox_w9->Text = Convert::ToString(hmg_p[8][3]);

		textBox_x10->Text = Convert::ToString(hmg_p[9][0]);
		textBox_y10->Text = Convert::ToString(hmg_p[9][1]);
		textBox_z10->Text = Convert::ToString(hmg_p[9][2]);
		textBox_w10->Text = Convert::ToString(hmg_p[9][3]);

		textBox_x11->Text = Convert::ToString(hmg_p[10][0]);
		textBox_y11->Text = Convert::ToString(hmg_p[10][1]);
		textBox_z11->Text = Convert::ToString(hmg_p[10][2]);
		textBox_w11->Text = Convert::ToString(hmg_p[10][3]);

		textBox_x12->Text = Convert::ToString(hmg_p[11][0]);
		textBox_y12->Text = Convert::ToString(hmg_p[11][1]);
		textBox_z12->Text = Convert::ToString(hmg_p[11][2]);
		textBox_w12->Text = Convert::ToString(hmg_p[11][3]);
}
private: System::Void matrix_mult(int number_of_vertex_A, double A[][dimension], double B[][dimension], double C[][dimension]) {
		int i, j, k;
		for (i = 0; i < number_of_vertex_A; i++)
			for (j = 0; j < dimension; j++) {
				C[i][j] = 0;
				for (k = 0; k < dimension; k++)
					C[i][j] += (A[i][k] * B[k][j]);
			}
}
private: System::Void hmg2dek(int number_of_vertex_A, double HMG[][dimension], int DEK[][dimension - 1]) {
		for (int i = 0; i < number_of_vertex_A; i++) {
			double a = HMG[i][dimension - 1];
			if (a != 0) {
				DEK[i][0] = HMG[i][0] / a;
				DEK[i][1] = HMG[i][1] / a;
			}
			else {
```

```
                                        System::Windows::Forms::MessageBox::Show("Division by zero is impossible! Check the coordinates of the shape and the transformation matrix");
                                        break;
                                    }
                                }
                            }
```

```
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];

            //верхнее основание
            e->Graphics->DrawLine(System::Drawing::Pens::Green, x1, y1, x2, y2);
            e->Graphics->DrawLine(System::Drawing::Pens::Green, x2, y2, x3, y3);
            e->Graphics->DrawLine(System::Drawing::Pens::Green, x3, y3, x4, y4);
            e->Graphics->DrawLine(System::Drawing::Pens::Green, x4, y4, x5, y5);
            e->Graphics->DrawLine(System::Drawing::Pens::Green, x5, y5, x6, y6);
            e->Graphics->DrawLine(System::Drawing::Pens::Green, x6, y6, x1, y1);
```

```
            //нижнее основание
            e->Graphics->DrawLine(System::Drawing::Pens::Blue, x7, y7, x8, y8);
            e->Graphics->DrawLine(System::Drawing::Pens::Blue, x8, y8, x9, y9);
            e->Graphics->DrawLine(System::Drawing::Pens::Blue, x9, y9, x10, y10);
            e->Graphics->DrawLine(System::Drawing::Pens::Blue, x10, y10, x11, y11);
            e->Graphics->DrawLine(System::Drawing::Pens::Blue, x11, y11, x12, y12);
            e->Graphics->DrawLine(System::Drawing::Pens::Blue, x12, y12, x7, y7);

            //рёбры между основаниями (высоты)
            e->Graphics->DrawLine(System::Drawing::Pens::Black, x1, y1, x7, y7);
            e->Graphics->DrawLine(System::Drawing::Pens::Red, x2, y2, x8, y8);
            e->Graphics->DrawLine(System::Drawing::Pens::Red, x3, y3, x9, y9);
            e->Graphics->DrawLine(System::Drawing::Pens::Red, x4, y4, x10, y10);
            e->Graphics->DrawLine(System::Drawing::Pens::Red, x5, y5, x11, y11);
            e->Graphics->DrawLine(System::Drawing::Pens::Red, x6, y6, x12, y12);
}
```

Остальные 3 обработчика pictureBox_x_Paint идентичны этому

```
private: System::Void btn_draw_Click(System::Object^ sender, System::EventArgs^ e) {
            hmg_p[0][0] = Convert::ToDouble(textBox_x1->Text);
            hmg_p[0][1] = Convert::ToDouble(textBox_y1->Text);
            hmg_p[0][2] = Convert::ToDouble(textBox_z1->Text);
            hmg_p[0][3] = Convert::ToDouble(textBox_w1->Text);

            hmg_p[1][0] = Convert::ToDouble(textBox_x2->Text);
            hmg_p[1][1] = Convert::ToDouble(textBox_y2->Text);
            hmg_p[1][2] = Convert::ToDouble(textBox_z2->Text);
            hmg_p[1][3] = Convert::ToDouble(textBox_w2->Text);

            hmg_p[2][0] = Convert::ToDouble(textBox_x3->Text);
            hmg_p[2][1] = Convert::ToDouble(textBox_y3->Text);
            hmg_p[2][2] = Convert::ToDouble(textBox_z3->Text);
            hmg_p[2][3] = Convert::ToDouble(textBox_w3->Text);

            hmg_p[3][0] = Convert::ToDouble(textBox_x4->Text);
            hmg_p[3][1] = Convert::ToDouble(textBox_y4->Text);
            hmg_p[3][2] = Convert::ToDouble(textBox_z4->Text);
            hmg_p[3][3] = Convert::ToDouble(textBox_w4->Text);

            hmg_p[4][0] = Convert::ToDouble(textBox_x5->Text);
            hmg_p[4][1] = Convert::ToDouble(textBox_y5->Text);
            hmg_p[4][2] = Convert::ToDouble(textBox_z5->Text);
            hmg_p[4][3] = Convert::ToDouble(textBox_w5->Text);

            hmg_p[5][0] = Convert::ToDouble(textBox_x6->Text);
            hmg_p[5][1] = Convert::ToDouble(textBox_y6->Text);
            hmg_p[5][2] = Convert::ToDouble(textBox_z6->Text);
            hmg_p[5][3] = Convert::ToDouble(textBox_w6->Text);

            hmg_p[6][0] = Convert::ToDouble(textBox_x7->Text);
            hmg_p[6][1] = Convert::ToDouble(textBox_y7->Text);
            hmg_p[6][2] = Convert::ToDouble(textBox_z7->Text);
            hmg_p[6][3] = Convert::ToDouble(textBox_w7->Text);

            hmg_p[7][0] = Convert::ToDouble(textBox_x8->Text);
            hmg_p[7][1] = Convert::ToDouble(textBox_y8->Text);
            hmg_p[7][2] = Convert::ToDouble(textBox_z8->Text);
            hmg_p[7][3] = Convert::ToDouble(textBox_w8->Text);

            hmg_p[8][0] = Convert::ToDouble(textBox_x9->Text);
            hmg_p[8][1] = Convert::ToDouble(textBox_y9->Text);
```

```
            hmg_p[8][2] = Convert::ToDouble(textBox_z9->Text);
            hmg_p[8][3] = Convert::ToDouble(textBox_w9->Text);

            hmg_p[9][0] = Convert::ToDouble(textBox_x10->Text);
            hmg_p[9][1] = Convert::ToDouble(textBox_y10->Text);
            hmg_p[9][2] = Convert::ToDouble(textBox_z10->Text);
            hmg_p[9][3] = Convert::ToDouble(textBox_w10->Text);

            hmg_p[10][0] = Convert::ToDouble(textBox_x11->Text);
            hmg_p[10][1] = Convert::ToDouble(textBox_y11->Text);
            hmg_p[10][2] = Convert::ToDouble(textBox_z11->Text);
            hmg_p[10][3] = Convert::ToDouble(textBox_w11->Text);

            hmg_p[11][0] = Convert::ToDouble(textBox_x12->Text);
            hmg_p[11][1] = Convert::ToDouble(textBox_y12->Text);
            hmg_p[11][2] = Convert::ToDouble(textBox_z12->Text);
            hmg_p[11][3] = Convert::ToDouble(textBox_w12->Text);

            double Result[vert_num][dimension] = { 0 };
            matrix_mult(vert_num, hmg_p, matrix_Center, Result);
            hmg2dek(vert_num, Result, dek_p);
            pictureBox_central->Refresh();

            matrix_mult(vert_num, hmg_p, matrix_Front, Result);
            hmg2dek(vert_num, Result, dek_p);
            pictureBox_front->Refresh();

            matrix_mult(vert_num, hmg_p, matrix_Dimetry, Result);
            hmg2dek(vert_num, Result, dek_p);
            pictureBox_dimetric->Refresh();

            matrix_mult(vert_num, hmg_p, matrix_Cabinet, Result);
            hmg2dek(vert_num, Result, dek_p);
            update_textBoxes();
            pictureBox_cabinet->Refresh();

}
```

```
private: System::Void btn_init_Click(System::Object^ sender, System::EventArgs^ e) {
            double centerX = Convert::ToDouble(textBox_centroidx->Text);
            double centerY = Convert::ToDouble(textBox_centroidy->Text);
            double centerZ = Convert::ToDouble(textBox_centroidz->Text);

            double edge = Convert::ToDouble(textBox_edge->Text);

            double height = Convert::ToDouble(textBox_height->Text);

            double angle_rad = 60 * deg2rad;

            hmg_p[0][0] = centerX + edge;
            hmg_p[0][1] = centerY;
            hmg_p[0][2] = centerZ + height;
            hmg_p[0][3] = 1;

            for (int i = 1; i < vert_num / 2; i++) {
                        hmg_p[i][0] = centerX + edge * cos(angle_rad * i);
                        hmg_p[i][1] = centerY + edge * sin(angle_rad * i);
                        hmg_p[i][2] = centerZ + height;
                        hmg_p[i][3] = 1;
            }

            hmg_p[6][0] = centerX + edge;
            hmg_p[6][1] = centerY;
```

```
            hmg_p[6][2] = centerZ;
            hmg_p[6][3] = 1;

            for (int i = vert_num / 2; i < vert_num; i++) {
                        hmg_p[i][0] = centerX + edge * cos(angle_rad * i);
                        hmg_p[i][1] = centerY + edge * sin(angle_rad * i);
                        hmg_p[i][2] = centerZ;
                        hmg_p[i][3] = 1;
            }

            double Result[vert_num][dimension] = { 0 };
            matrix_mult(vert_num, hmg_p, matrix_Center, Result);
            hmg2dek(vert_num, Result, dek_p);
            pictureBox_central->Refresh();

            matrix_mult(vert_num, hmg_p, matrix_Front, Result);
            hmg2dek(vert_num, Result, dek_p);
            pictureBox_front->Refresh();

            matrix_mult(vert_num, hmg_p, matrix_Dimetry, Result);
            hmg2dek(vert_num, Result, dek_p);
            pictureBox_dimetric->Refresh();

            matrix_mult(vert_num, hmg_p, matrix_Cabinet, Result);
            hmg2dek(vert_num, Result, dek_p);
            update_textBoxes();
            pictureBox_cabinet->Refresh();
}
```

```
private: System::Void btn_rotate_Click(System::Object^ sender, System::EventArgs^ e) {
            double matrix_buf_1[vert_num][dimension] = { 0 };// буфферная матрица преобразования
            double matrix_buf_2[vert_num][dimension] = { 0 };// буфферная матрица преобразования

            double px = Convert::ToDouble(textBox_rotx1->Text);
            double py = Convert::ToDouble(textBox_roty1->Text);
            double pz = Convert::ToDouble(textBox_rotz1->Text);

            double qx = Convert::ToDouble(textBox_rotx2->Text);
            double qy = Convert::ToDouble(textBox_roty2->Text);
            double qz = Convert::ToDouble(textBox_rotz2->Text);

            double cx = hmg_p[0][0];
            double cy = hmg_p[0][1];
            double cz = hmg_p[0][2];

            double alpha = Convert::ToDouble(textBox_degree->Text) * deg2rad;

            // Шаг 1: Перенос P в начало координат----------------------------------------------------
            double T[dimension][dimension] = {
            {1, 0, 0, -px},
            {0, 1, 0, -py},
            {0, 0, 1, -pz},
            {0, 0, 0, 1}
            };

            // Шаг 2: Выравнивание вектора PQ с плоскостью XZ-----------------------------------------
            // Рассчитываем угол φ
            double dx = qx - px;
            double dy = qy - py;
            double dz = qz - pz;
            double phi = atan2(dy, dx);

            // Матрица поворота вокруг Z для выравнивания с XZ
```

```
double Rz[dimension][dimension] = {
                {cos(-phi), -sin(-phi), 0, 0},
                {sin(-phi), cos(-phi), 0, 0},
                {0, 0, 1, 0},
                {0, 0, 0, 1}
};

// Шаг 3: Выравнивание вектора PQ с осью Z----------------------------------------------------
// Рассчитываем угол θ
double theta = atan2(sqrt(dx * dx + dy * dy), dz);

// Матрица поворота вокруг Y для выравнивания с Z
double Ry[dimension][dimension] = {
                {cos(theta), 0, sin(theta), 0},
                {0, 1, 0, 0},
                {-sin(theta), 0, cos(theta), 0},
                {0, 0, 0, 1}
};

// Шаг 4: Поворот вокруг оси Z----------------------------------------------------
// Матрица поворота вокруг Z на угол alpha
double Rz_alpha[dimension][dimension] = {
                {cos(alpha), -sin(alpha), 0, 0},
                {sin(alpha), cos(alpha), 0, 0},
                {0, 0, 1, 0},
                {0, 0, 0, 1}
};

// Шаг 5: Обратные преобразования----------------------------------------------------
// Обратный поворот вокруг Y
double Ry_inv[dimension][dimension] = {
                {cos(-theta), 0, -sin(-theta), 0},
                {0, 1, 0, 0},
                {sin(-theta), 0, cos(-theta), 0},
                {0, 0, 0, 1}
};

// Обратный поворот вокруг Z
double Rz_inv[dimension][dimension] = {
                {cos(-phi), sin(-phi), 0, 0},
                {-sin(-phi), cos(-phi), 0, 0},
                {0, 0, 1, 0},
                {0, 0, 0, 1}
};

// Обратный перенос
double T_inv[dimension][dimension] = {
                {1, 0, 0, px},
                {0, 1, 0, py},
                {0, 0, 1, pz},
                {0, 0, 0, 1}
};

matrix_mult(vert_num, hmg_p, T, matrix_buf_1);
matrix_mult(vert_num, matrix_buf_1, Rz, matrix_buf_2);
matrix_mult(vert_num, matrix_buf_2, Ry, matrix_buf_1);
matrix_mult(vert_num, matrix_buf_1, Rz_alpha, matrix_buf_2);
matrix_mult(vert_num, matrix_buf_2, Ry_inv, matrix_buf_1);
matrix_mult(vert_num, matrix_buf_1, Rz_inv, matrix_buf_2);
matrix_mult(vert_num, matrix_buf_2, T_inv, hmg_p);

double Result[vert_num][dimension] = { 0 };
```

```
        matrix_mult(vert_num, hmg_p, matrix_Center, Result);
        hmg2dek(vert_num, Result, dek_p);
        pictureBox_central->Refresh();

        matrix_mult(vert_num, hmg_p, matrix_Front, Result);
        hmg2dek(vert_num, Result, dek_p);
        pictureBox_front->Refresh();

        matrix_mult(vert_num, hmg_p, matrix_Dimetry, Result);
        hmg2dek(vert_num, Result, dek_p);
        pictureBox_dimetric->Refresh();

        matrix_mult(vert_num, hmg_p, matrix_Cabinet, Result);
        hmg2dek(vert_num, Result, dek_p);
        pictureBox_cabinet->Refresh();

        update_textBoxes();
}
```

1. Преобразования проецирований

Топологическое - спереди:

В общем случае картинная плоскость может не совпадать с плоскостью *XOY*, а быть ей параллельна. Тогда координата *z* у всех точек объекта после проецирования будет равна ν. Соответственно, матрица проецирования имеет вид

$$\mathbf{P}_{\text{В.СПЕРЕДИ}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \nu & 1 \end{bmatrix}.$$

Косоугольное - кабине:

$$\mathbf{P}_{\text{КОС}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ l*\cos\alpha & l*\sin\alpha & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Параметр l задается и определяет вид косоугольной проекции.

Для проекции Кавалье l=1 , для проекции кабине l=1/2. Угол α обычно принимается равным 30 или 45 град.

Аксонометрическое - диметрия:

В изометрии углы между осями XYZ одинаковые и равны 120°, в диметрии равны два из трех углов, в триметрии величина всех углов между осями разная.

$$\mathbf{P}_{\text{АКС}} = \begin{bmatrix} \cos\psi & \sin\varphi * \sin\psi & 0 & 0 \\ 0 & \cos\varphi & 0 & 0 \\ \sin\psi & -\sin\varphi * \cos\psi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Таким образом, для реализации диметрии углы φ и ψ должны удовлетворять условию

$$\text{tg}^2\varphi = \sin^2\psi.$$

Возьмём угол $\psi = 45°$, тогда $\varphi = \pm\arctan(\sin(45°))$

Центральное - одноточечное:

Возьмём центр проекции d = 300

$$\mathbf{P}_{\text{ЦЕНТР}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/d \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

2. Специфичное преобразование

Вращение многогранника вокруг произвольной прямой состоит из следующих шагов:

Обозначим произвольную прямую вектором PQ

1) Перенос P в начало координат

{1, 0, 0, -px},

{0, 1, 0, -py},

{0, 0, 1, -pz},

{0, 0, 0, 1}

2) Выравнивание вектора PQ с плоскостью XZ

Этот шаг требует вычисления угла φ между проекцией вектора на плоскость YZ и осью Y. Этот угол используется для поворота вокруг оси Z.

```
// Рассчитываем угол φ
  double dx = qx - px;
  double dy = qy - py;
  double dz = qz - pz;
  double phi = atan2(dy, dx);

  // Матрица поворота вокруг Z для выравнивания с XZ
  {cos(-phi),    -sin(-phi),    0,    0},
  {sin(-phi),    cos(-phi),     0,    0},
  {0,            0,             1,    0},
  {0,            0,             0,    1}
```

3) Выравнивание вектора PQ с осью Z

После того как вектор будет выровнен с плоскостью XZ, следующий шаг — выравнивание с осью Z, что требует поворота вокруг оси Y.

```
// Рассчитываем угол θ
  double theta = atan2(sqrt(dx * dx + dy * dy), dz);

  // Матрица поворота вокруг Y для выравнивания с Z
  {cos(theta),    0,    sin(theta),    0},
  {0,             1,    0,             0},
  {-sin(theta),   0,    cos(theta),    0},
  {0,             0,    0,             1}
```

4) Поворот вокруг оси Z

Теперь, когда ось вращения выровнена с осью Z, можно выполнить поворот на заданный угол α.

// Матрица поворота вокруг Z на угол alpha

{cos(alpha), -sin(alpha), 0, 0},
{sin(alpha), cos(alpha), 0, 0},
{0, 0, 1, 0},
{0, 0, 0, 1}

5) Обратные преобразования

После вращения нужно выполнить обратные преобразования для возврата фигуры в исходное положение.

// Обратный поворот вокруг Y

{cos(-theta), 0, -sin(-theta), 0},
{0, 1, 0, 0},
{sin(-theta), 0, cos(-theta), 0},
{0, 0, 0, 1}

// Обратный поворот вокруг Z

{cos(-phi), sin(-phi), 0, 0},
{-sin(-phi), cos(-phi), 0, 0},
{0, 0, 1, 0},
{0, 0, 0, 1}

// Обратный перенос

{1, 0, 0, px},
{0, 1, 0, py},
{0, 0, 1, pz},
{0, 0, 0, 1}

Выводы:

Были изучены механизмы проецирований, а также разработано преобразование вращения многогранника вокруг произвольной прямой в 3D пространстве.