Министерство образования и науки РФ

ФГБОУ ВПО «Тамбовский государственный технический университет»

Кафедра «САПР»

Лабораторная работа №3

по дисциплине «Компьютерная графика»

Трёхмерные преобразования

Выполнил: студент гр. Д. Ю.

Проверил: С. А.

Тамбов,

***Цели и задачи****.*

Построить каркас 3d объекта, произвести над ним аффинные преобразования (масштабирование, повороты, перенос).

***Решение задачи****.*

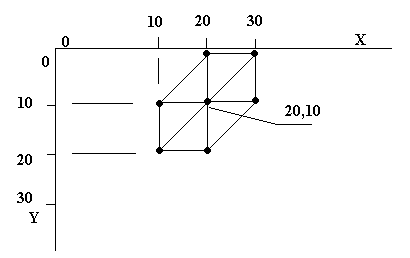
Первым делом строим объект – куб. Размеры каждой из сторон равны 10 пикселей (Рисунок 1). На рисунке 2 изображены координаты вершин объекта.

Рисунок 2 Координатная плоскость

Рисунок 1 Куб

После постороения изображение отцентровывается методом Draw:

private void Draw(Graphics graphics)

{

//переносим обьект в центр

for (int i = 0; i < arrPoint.Length; i++)

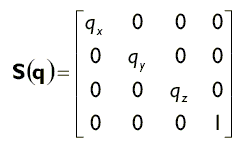
{

arrPoint[i] = arrPoint[i] + \_НачалоКоординат;

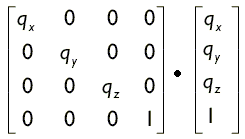
}

graphics.DrawPolygon(\_pen, arrPoint);

}

Для выполнения масштибирования используется матрица следующего вида:

, где q – это некий коэфициент увеличения.

В общем виде операция сводится к перемножению двух матриц:

, где матрица справо это координаты точки объекта. Итогом перемножения являются четыре числа, координаты новой точки, которая в последствии будет выводится на экран. Код отвечающий за масштаб сводится к набору методов:

public Point[] Mashtab(int коэфф)

{

//Взять arrPoint[] и все значения перегнатьчерез матрицу

//на выходе получить матрицу arrPoint[] и отобразить её

int[] S = new int[]

{

коэфф, 0, 0, 0,

0, коэфф, 0, 0,

0, 0, коэфф, 0,

0, 0, 0, 1

};

Point[] tmpArr = new Point[arrPoint.Length];

int i = 0;

foreach (Point point in arrPoint)

{

tmpArr[i] = new Point((point.X - \_НачалоКоординат.Height)\*коэфф,

(point.Y - \_НачалоКоординат.Width)\*коэфф);

i++;

}

return Centrovka(tmpArr);

}

В метод передают коэфициент масштабирования, затем берём массив точек текущего объекта и каждую точку переносим в начало координат, после увеличиваем на коэфициент, а затем центруем полученное.

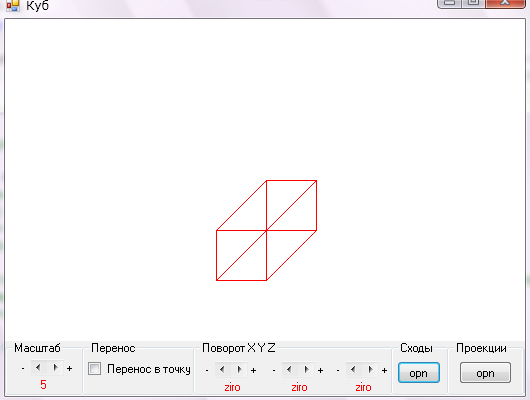
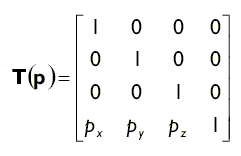
На рисунке 3 показан объект после операции масштабирования и центровки.

Рисунок 3 Масштаб +5

Следующим реализован перенос объекта в точку.

Матрица имеет следующий вид:

, где p – это координаты той точки, в которую необходимо передвинуть объект.

Сколько не пробовал этот вариант, так он и не заработал адекватно. Реализация в коде имеет следующий тип:

Кликнули по области холста, получили точку для переноса в неё. Передаем её в метод Perenos:

public Point[] Perenos(Point tochka)

{

Point[] tmpArr = new Point[arrPoint.Length];

int i = 0;

foreach (Point point in \_tempArray)

{

tmpArr[i] = new Point((point.X + tochka.X) - \_НачалоКоординат.Height-10,

(point.Y + tochka.Y) - \_НачалоКоординат.Width-10);

i++;

}

//\_tempArray = tmpArr;

return tmpArr;

/\*//Как должно но не работает

Point[] tmp = new Point[15];

int i = 0;

foreach (Point point in \_tempArray)

{

int[] tmper = A\_mul\_B('p', new int[] {point.X, point.Y, 0, 0}, tochka);

tmp[i] = new Point(tmper[0], tmper[1]);

i++;

}

return tmp;

\*/

}

Точки текущего объекта переносим в начало координат и рисуем со сдвигом на коэфициент точки переноса. Не используем центровку точек! Этот вариант работает, но без использования матрицы. С матрицей логика такая же, каждую точку старого объекта умножаем в методе A\_mul\_B:

internal int[] A\_mul\_B(char flagOper, int[] A, PointF koeff

{

switch (flagOper)

{

case 'p':

{

int[] S = new int[]

{

1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

(int)koeff.X, (int)koeff.Y, 0, 1

};

return new int[]

{

(1\*A[0] + 0\*A[1] + 0\*A[2] + 0\*A[3]),//X

(0\*A[0] + 1\*A[1] + 0\*A[2] + 0\*A[3]),//Y

(0\*A[0] + 0\*A[1] + 1\*A[2] + 0\*A[3]),//Z

(int)(koeff.X\*A[0] + koeff.Y\*A[1] + 0\*A[2] + 1\*A[3])//A

};

}

break;

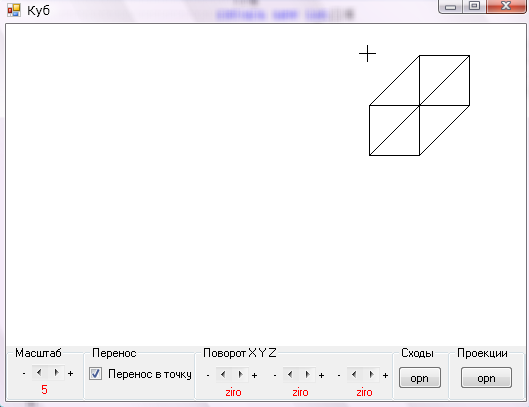
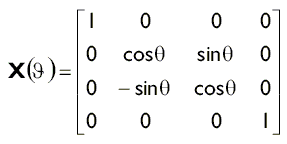
Перенос в точку изображен на рисунке 4.

Рисунок 4 Операция переноса объекта из центра холста в точку

Следующим реализован поворот объекта.

Для поворота относительно оси Х используют следующую матрицу:

, где Ɵ – это угол поворота.

Метод поворота по оси Х принимает значение угла и выполняет преобразование каждой точки объекта в методе A\_mul\_B:

public Point[] PovorotX(PointF tochka)

{

Point[] tmp = new Point[15];

int i = 0;

foreach (Point point in \_tempArray)

{

int[] tmper = A\_mul\_B('x', new int[] {point.X, point.Y, 0, 0}, tochka);

tmp[i] = new Point(tmper[0], tmper[1]);

i++;

}

return tmp;

}

internal int[] A\_mul\_B(char flagOper, int[] A, PointF koeff

{

switch (flagOper)

{

case 'x':

{

int[] S = new int[]

{

1, 0, 0, 0,

0, (int)Math.Cos(koeff.X), (int)Math.Sin(koeff.X), 0,

0, -(int)Math.Sin(koeff.X), (int)Math.Cos(koeff.X), 0,

0, 0, 0, 1

};

return new int[]

{

(1\*A[0] + 0\*A[1] + 0\*A[2] + 0\*A[3]),//X

(int)(0\*A[0] + Math.Cos(koeff.X)\*A[1] + Math.Sin(koeff.X)\*A[2] + 0\*A[3]),//Y

(int)(0\*A[0] - Math.Sin(koeff.X)\*A[1] + Math.Cos(koeff.X)\*A[2] + 0\*A[3]),//Z

(0\*A[0] + 0\*A[1] + 0\*A[2] + 1\*A[3])//A

};

}

break;

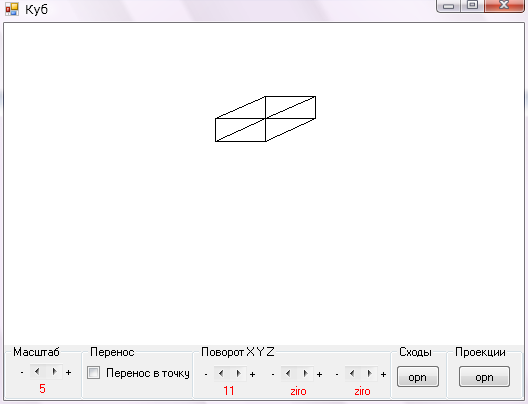
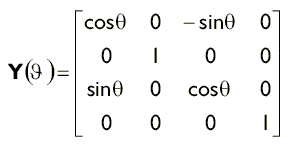
Результат изображен на рисунке 5.

Рисунок 5 Поворот объекта по Х на 11 единиц или 11\*0.1

Для поворота относительно оси Y используют следующую матрицу:

, где Ɵ – это угол поворота.

Метод поворота по оси Y принимает значение угла и выполняет преобразование каждой точки объекта в методе A\_mul\_B:

public Point[] PovorotY(PointF tochka)

{

Point[] tmp = new Point[15];

int i = 0;

foreach (Point point in \_tempArray)

{

int[] tmper = A\_mul\_B('y', new int[] { point.X, point.Y, 0, 0 }, tochka);

tmp[i] = new Point(tmper[0], tmper[1]);

i++;

}

return tmp;

}

internal int[] A\_mul\_B(char flagOper, int[] A, PointF koeff

{

switch (flagOper)

{

case 'y':

{

int[] S = new int[]

{

(int)Math.Cos(koeff.X), 0, -(int)Math.Sin(koeff.X), 0,

0, 1, 0, 0,

(int)Math.Sin(koeff.X), 0, (int)Math.Cos(koeff.X), 0,

0, 0, 0, 1

};

return new int[]

{

(int)(Math.Cos(koeff.X)\*A[0] + 0\*A[1] - Math.Sin(koeff.X)\*A[2] + 0\*A[3]),//X

(0\*A[0] + 1\*A[1] + 0\*A[2] + 0\*A[3]),//Y

(int)(Math.Sin(koeff.X)\*A[0] + 0\*A[1] + Math.Cos(koeff.X)\*A[2] + 0\*A[3]),//Z

(0\*A[0] + 0\*A[1] + 0\*A[2] + 1\*A[3])//A

};

}

break;

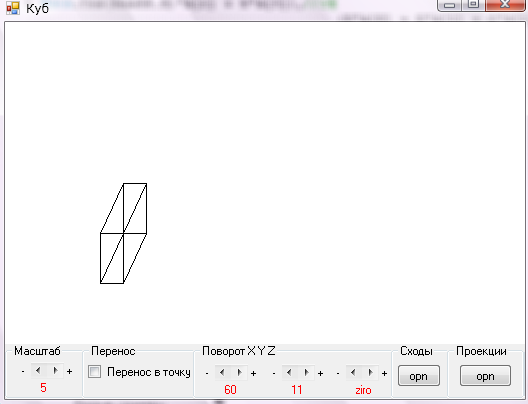
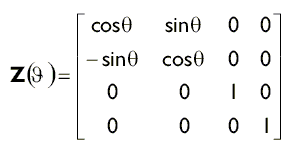
Результат изображен на рисунке 6.

Рисунок 6 Поворот по Y на 11 единиц или 11\*0.1

Для поворота относительно оси Z используют следующую матрицу:

, где Ɵ – это угол поворота.

Метод поворота по оси Z принимает значение угла и выполняет преобразование каждой точки объекта в методе A\_mul\_B:

public Point[] PovorotZ(PointF tochka)

{

Point[] tmp = new Point[15];

int i = 0;

foreach (Point point in \_tempArray)

{

int[] tmper = A\_mul\_B('z', new int[] { point.X, point.Y, 0, 0 }, tochka);

tmp[i] = new Point(tmper[0], tmper[1]);

i++;

}

return tmp;

}

internal int[] A\_mul\_B(char flagOper, int[] A, PointF koeff

{

switch (flagOper)

{

case 'z':

{

int[] S = new int[]

{

(int)Math.Cos(koeff.X), (int)Math.Sin(koeff.X), 0, 0,

-(int)Math.Sin(koeff.X), (int)Math.Cos(koeff.X), 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

};

return new int[]

{

(int)(Math.Cos(koeff.X)\*A[0] + Math.Sin(koeff.X)\*A[1] + 0\*A[2] + 0\*A[3]),//X

(int)(-Math.Sin(koeff.X)\*A[0] + Math.Cos(koeff.X)\*A[1] + 0\*A[2] + 0\*A[3]),//Y

(0\*A[0] + 0\*A[1] + 1\*A[2] + 0\*A[3]),//Z

(0\*A[0] + 0\*A[1] + 0\*A[2] + 1\*A[3])//A

};

}

break;

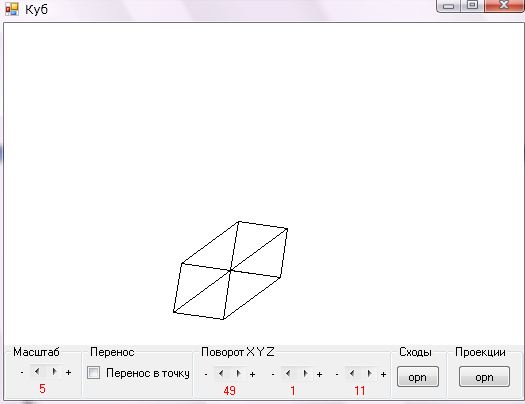
Результат изображен на рисунке 7.

Рисунок 7 Поворот по оси Z на 11 единиц или 11\*0.1

Далее реализовывались точки схода.

Суть в следующем, изображение становится удоляющимся в даль по заданной оси, расстояние от объекта до мнимой точки горизонта задаётся равным 600-800 единиц.

/// <summary>

/// Расстояние от проекцПлоскости до смотрящего

/// </summary>

private const float \_d = 600;

/// <summary>

/// Проекция для точек схода

/// </summary>

internal PointF[] Proekc(bool x, bool y, bool z)

{

/\*

Одна точка схода по Z

\* x' = x/(1+z/d)

\* y' = y/(1+z/d)

Две точки схода по Z и Y

\* x' = x/(1+z/d + y/d)

\* y' = y/(1+z/d + y/d)

Три точки схода по Z\_Y\_X

\* x' = x/(1+z/d + y/d + x/d)

\* y' = y/(1+z/d + y/d + x/d)

\*/

PointF[] tmp = new PointF[15];

int i = 0;

if(x&&y&&z)//x-y-z

foreach (Point point in \_tempArray)

{

tmp[i] = new PointF(point.X / (1.0f + 0 / \_d + point.X / \_d + point.Y / \_d),

point.Y / (1.0f + 0 / \_d + point.X / \_d + point.Y / \_d));

i++;

}

else if (!x&&y&&z)//y-z

{

i = 0;

foreach (Point point in \_tempArray)

{

tmp[i] = new PointF(point.X / (1 + 0 / \_d + point.Y / \_d),

point.Y / (1 + 0 / \_d + point.Y / \_d));

i++;

}

}

else if (!x && !y && z)//z

{

i = 0;

foreach (Point point in \_tempArray)

{

tmp[i] = new PointF(point.X / (1 + 0 / \_d),

point.Y / (1 + 0 / \_d));

i++;

}

}

if (x && y && !z)//x-y

{

i = 0;

foreach (Point point in \_tempArray)

{

tmp[i] = new PointF(point.X / (1 + point.X / \_d + point.Y / \_d),

point.Y / (1 + point.X / \_d + point.Y / \_d));

i++;

}

}

else if (x && !y && !z)//x

{

i = 0;

foreach (Point point in \_tempArray)

{

tmp[i] = new PointF(point.X / (1 + point.X / \_d),

point.Y / (1 + point.X / \_d));

i++;

}

}

if (x && !y && z)//x-z

{

i = 0;

foreach (Point point in \_tempArray)

{

tmp[i] = new PointF(point.X / (1 + point.X / \_d + 0 / \_d),

point.Y / (1 + point.X / \_d + 0 / \_d));

i++;

}

}

if (!x && y && !z)//y

{

i = 0;

foreach (Point point in \_tempArray)

{

tmp[i] = new PointF(point.X / (1 + point.Y / \_d),

point.Y / (1 + point.Y / \_d));

i++;

}

}

return tmp;

}

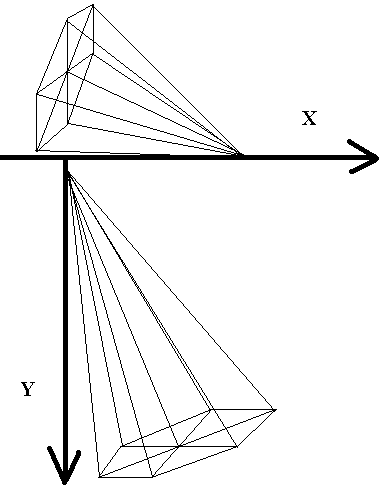
Метод преобразует каждую точку объекта по формулам. Результат вообще непредсказуемый, рисунок 8.

Рисунок 8 XYZ три точки схода

Рисунок 8 Точки схода по Х и по У

Реализация кабинетного и военного отображения в методе:

internal PointF[] Otobrajenie(bool kabinet, bool voennik)

{

/\*

x' = x + z \* (L \* Math.Cos(alpha))

y' = y + z \* (L \* Math.Sin(alpha))

\* Военник=L=1

\* Кабинет=L=0.5

\* alpha=30\_OR\_45

\*/

PointF[] arrTmp = new PointF[15];

int i = 0;

float Cos = (float)Math.Cos(45);

float Sin = (float)Math.Sin(45);

float Z = 2.5f;

if (kabinet)

{

foreach (Point point in \_tempArray)

{

arrTmp[i] = new PointF(point.X + Z \* (0.5f \* Cos), point.Y + Z \* (0.5f \* Sin));

i++;

}

}

else if (voennik)

{

foreach (Point point in \_tempArray)

{

arrTmp[i] = new PointF(point.X + Z \* (1.0f \* Cos), point.Y + Z \* (1.0f \* Sin));

i++;

}

}

return arrTmp;

}

В зависимости от флагов построение идет по одной или второй формулам.