**Министерство образования и науки Российской Федерации**

федеральное государственное автономное образовательное учреждение

высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Инженерная школа информационных технологий и робототехники

Отделение Информационных технологий

Профиль Геоинформационные системы

**ОТЧЕТ  
по лабораторной работе №3**

Базовые алгоритмы 2D-геометрии

по дисциплине Компьютерная графика

Выполнил студент группы 8И5Б \_\_\_\_\_\_\_\_\_\_ Смирнов П.О.

\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ 2018 г.

Отчет принят:

Принял: \_\_\_\_\_\_\_\_\_\_\_ Шумихин В.С.

\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ 2018 г.

Томск 2018 г.

Цель работы

Изучить способы выполнения базовых преобразований на плоскости.

Задачи

1. Реализовать чтение из файла координат вершин многоугольника, заданных в мировой системе координат.
2. Реализовать переход от мировых координат к экранным координатам.
3. Реализовать отрисовку фигур и осей координат в окне, так чтобы было место для отображения фигур после выполнения геометрических преобразований.
4. Реализовать выполнение преобразований над фигурами в соответствии с заданием по командам пользователя (использовать меню, кнопки или горячие клавиши).

**Вариант 13**

### пренести все фигуры в первую четверть;

### изменить масштаб, увеличив в 2 раза;

### повернуть относительно начала координат на 45 градусов против часовой стрелки.

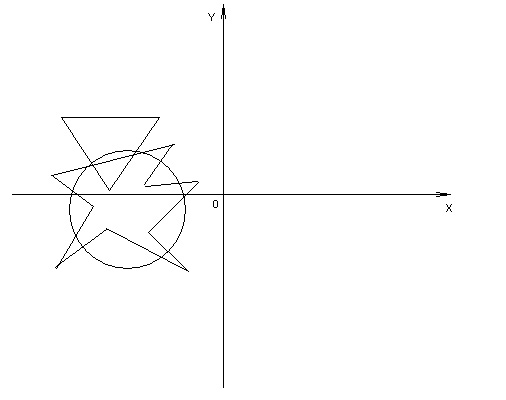


Рисунок 1 – Вариант задания

**Ход работы**

Геометрически каждая точка на плоскости задается значениями координат относительно выбранной системы координат. Для выполнения преобразований на плоскости используются однородные координаты. Будем рассматривать координаты точки как элементы списка. Для преобразования фигур необходимо координаты точек умножать на соответствующую матрицу преобразования.

Для конвертации фигур с файла был создан абстрактный класс.

[XmlInclude(typeof(OurPoint))]

[XmlInclude(typeof(Poliline))]

[XmlInclude(typeof(Circle))]

[XmlInclude(typeof(Array3x3))]

[XmlInclude(typeof(List<OurPoint>))]

public abstract class AbstractObject

{

public abstract void Draw(object sender, PaintEventArgs e);

}

Сериализация и дессериализация происходит в xml формат.

public void SaveInFile(string path, List<AbstractObject> objects)

{

XmlSerializer OurSerializer = new XmlSerializer(typeof(List<AbstractObject>));

TextWriter tw = new StreamWriter(@path);

OurSerializer.Serialize(tw, objects);

tw.Close();

}

public List<AbstractObject> ReadFromFile(string path)

{

List<AbstractObject> objects = new List<AbstractObject>();

XmlSerializer OurSerializer = new XmlSerializer(typeof(List<AbstractObject>));

StreamReader or = new StreamReader(@path);

objects = (List<AbstractObject>)OurSerializer.Deserialize(or);

return objects;

}

Т.к. в нашем варианте присутствует окружность, для ее отрисовки необходимо использовать параметрическое описание.

public override void Draw(object sender, PaintEventArgs e)

{

Pen OurPen = new Pen(Color.Black, 1);

float temp1 = Angle1;

float temp2 = Angle2;

Angle1 = (float)((Angle1 / 180) \* Math.PI);

Angle2 = (float)((Angle2 / 180) \* Math.PI);

float koef = (float)(Math.PI \* 2 / Math.Abs(Angle2 - Angle1));

float iterations = (float)Math.Round((2 \* Radius + 5) / koef);

float delta = (Angle2 - Angle1) / iterations;

float x1 = Center.X + Radius \* (float)Math.Cos(Angle1);

float y1 = Center.Y - Radius \* (float)Math.Sin(Angle1);

for (int i = 0; i < iterations; i++)

{

Angle1 += delta;

float x2 = Center.X + Radius \* (float)Math.Cos(Angle1);

float y2 = Center.Y - Radius \* (float)Math.Sin(Angle1);

e.Graphics.DrawLine(OurPen, x1, y1, x2, y2);

x1 = x2;

y1 = y2;

}

Angle1 = temp1;

Angle2 = temp2;

}

Для работы с матрицами был создан класс Array3x3 в котором реализованы методы перемножения матриц с координатами точек.

public class Array3x3

{

public List<OurPoint> secondListOfPoints = new List<OurPoint>();

public void Initialize()

{

if (secondListOfPoints.Count == 0)

{

AddRow(new OurPoint(0, 0, 0));

AddRow(new OurPoint(0, 0, 0));

AddRow(new OurPoint(0, 0, 0));

}

}

public Array3x3 ToCopy()

{

Array3x3 buffer = new Array3x3();

for(int i = 0; i < secondListOfPoints.Count; i++)

{

buffer.AddRow(new OurPoint(secondListOfPoints[i].X, secondListOfPoints[i].Y, secondListOfPoints[i].H));

}

return buffer;

}

public void InsertValues(Array3x3 inputArray)

{

for (int i = 0; i < inputArray.secondListOfPoints.Count; i++)

{

for (int j = 0; j < 3; j++)

{

secondListOfPoints[i].listOfCoordinates[j] = inputArray.secondListOfPoints[i].listOfCoordinates[j];

switch (j)

{

case 0:

{

secondListOfPoints[i].X = inputArray.secondListOfPoints[i].listOfCoordinates[j];

break;

}

case 1:

{

secondListOfPoints[i].Y = inputArray.secondListOfPoints[i].listOfCoordinates[j];

break;

}

case 2:

{

secondListOfPoints[i].H = inputArray.secondListOfPoints[i].listOfCoordinates[j];

break;

}

}

}

}

}

public Array3x3()

{

}

public void AddRow(OurPoint inputPoint)

{

secondListOfPoints.Add(new OurPoint(inputPoint.X, inputPoint.Y, inputPoint.H));

}

public void MultiplyArrays(Array3x3 left, Array3x3 right)

{

Array3x3 resultArray = new Array3x3();

resultArray.Initialize();

float buffer = 0;

for (int k = 0; k < left.secondListOfPoints.Count; k++)

{

if (k >= 3)

{

resultArray.AddRow(new OurPoint(0, 0, 0));

}

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

buffer += left.secondListOfPoints[k].listOfCoordinates[j] \* right.secondListOfPoints[j].listOfCoordinates[i];

}

resultArray.secondListOfPoints[k].listOfCoordinates[i] = buffer;

buffer = 0;

}

}

if (left.secondListOfPoints.Count != 1)

{

InsertValues(resultArray);

}

}

}

**Работа программы**

На рисунке 2 изображена работа программы. С левой стороны расположен пункт меню. На нем расположены инструменты для загрузки файла, изменения параметров фигуры(поворот, перенос, увеличение), выполнение всех операций разом и возврат в исходное состояние.

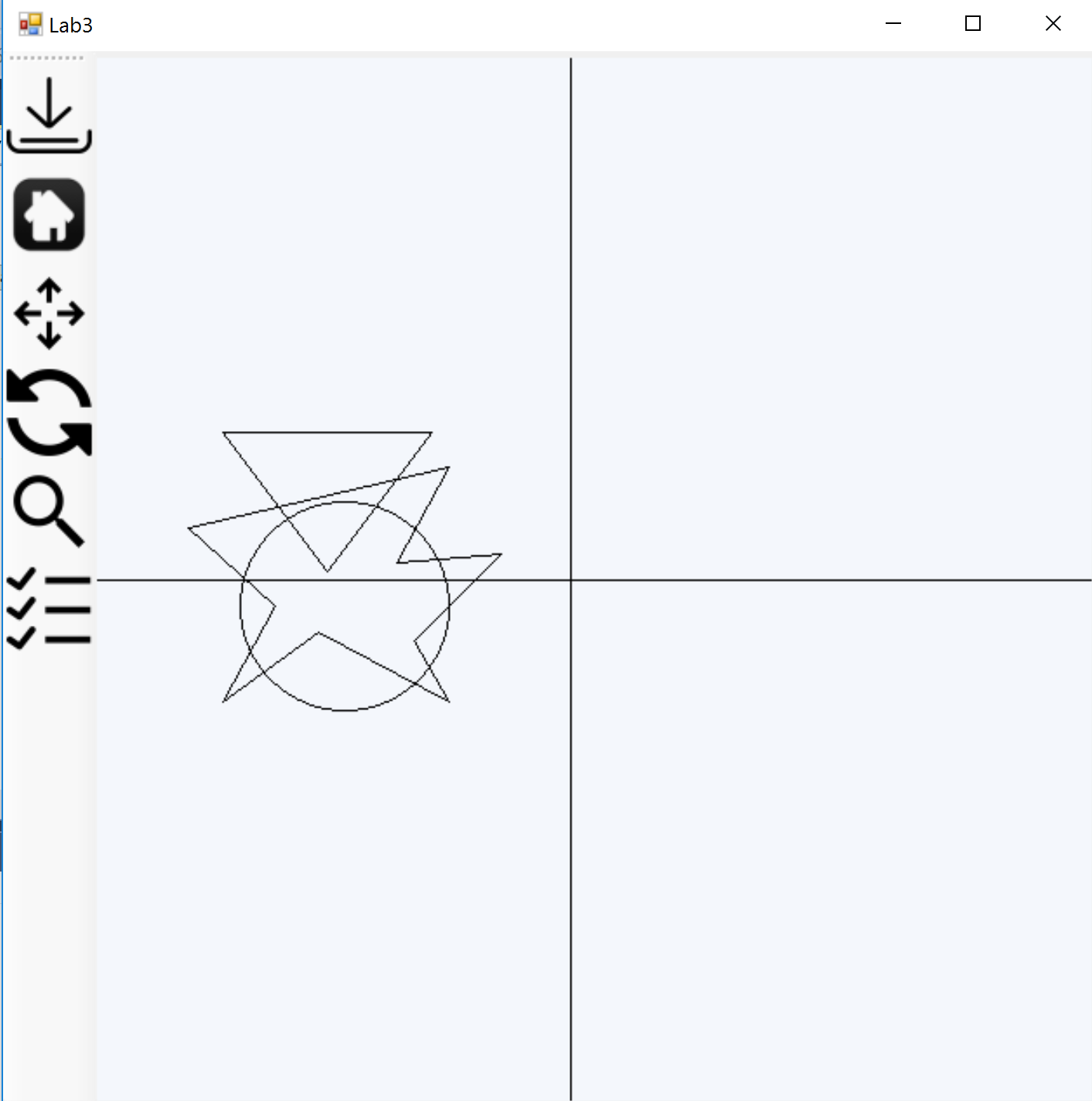


Рисунок 2 – Главное меню

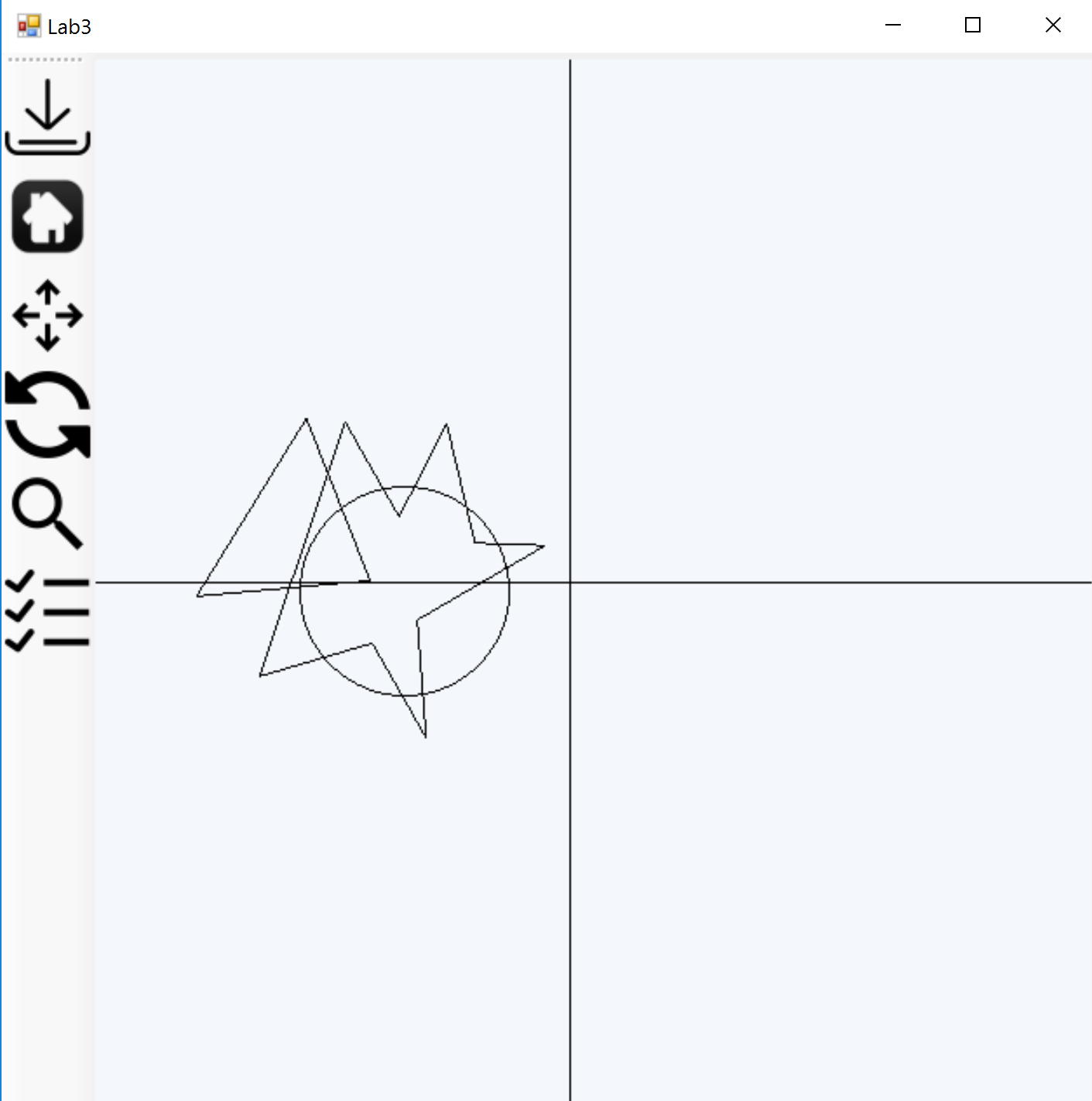


Рисунок 3 – Изменение параметров фигур

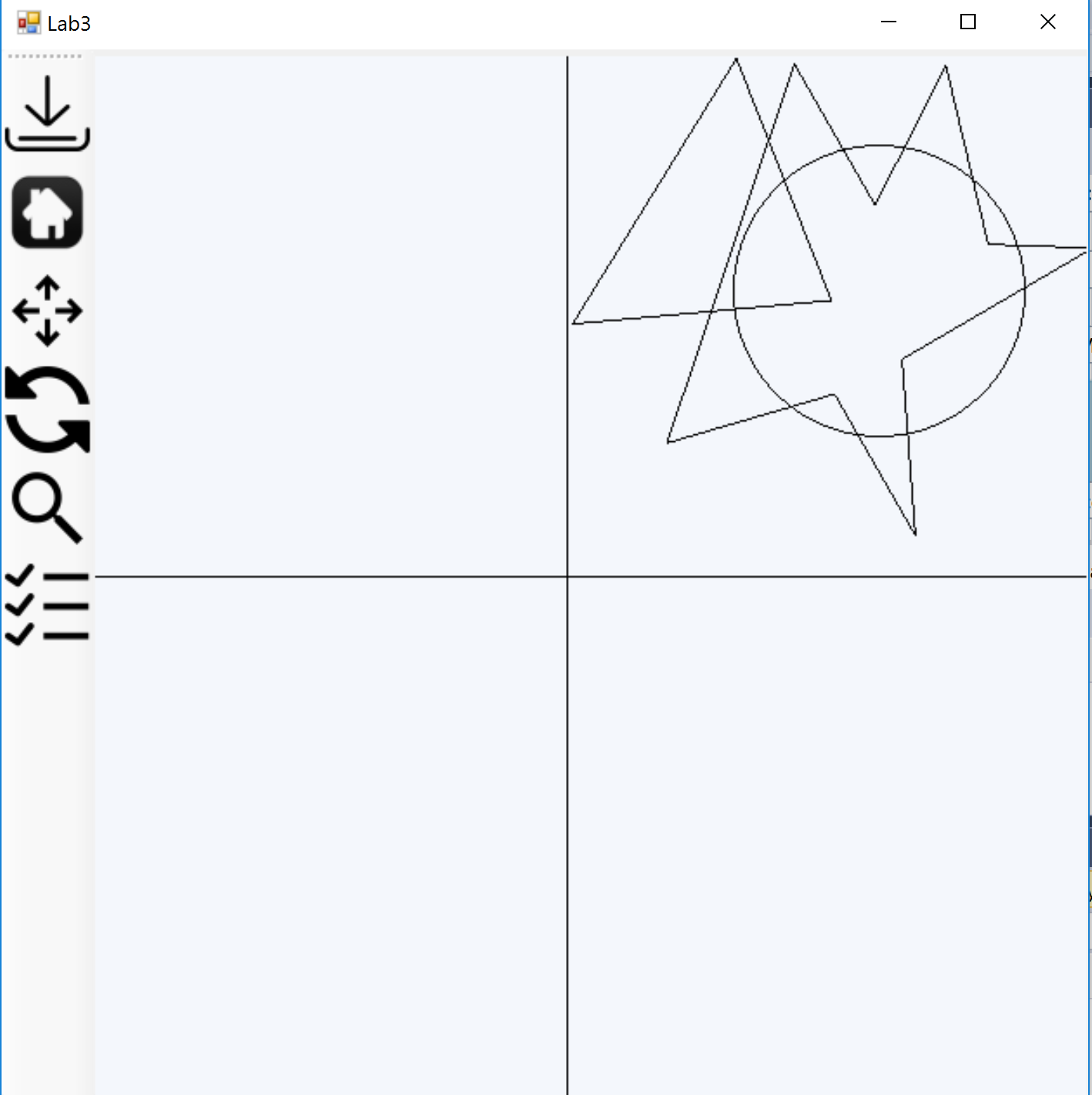


Рисунок 4 – Выполнение всех преобразований разом

# **Вывод**

В ходе проделанной лабораторной работы были изучены такие типы геометрических преобразований, как повороты, переносы, масштабирование. Все преобразования выполняются путем умножения координат точек на соответствующие матрицы преобразования.

**Приложение**

public class OurPoint : AbstractObject

{

public float X { get; set; }

public float Y { get; set; }

public float H { get; set; }

public List<float> listOfCoordinates = new List<float>();

public OurPoint()

{

}

public OurPoint(float x, float y, float h = 1)

{

X = x;

listOfCoordinates.Add(X);

Y = y;

listOfCoordinates.Add(Y);

H = h;

listOfCoordinates.Add(H);

}

public override void Draw(object sender, PaintEventArgs e)

{

}

}

public partial class Form1 : Form

{

public List<AbstractObject> ListOfObjects;

List<AbstractObject> bufferListAbstract;

bool draw = true;

int functionMode = 0;

public Form1()

{

InitializeComponent();

ListOfObjects = new List<AbstractObject>();

bufferListAbstract = new List<AbstractObject>();

}

private void OpenFile\_Click(object sender, EventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "XML file (\*.xml)|\*.xml";

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

Parser OurParser = new Parser();

ListOfObjects.Clear();

ListOfObjects = OurParser.ReadFromFile(openFileDialog.FileName);

DrawElements();

}

}

public void DrawElements()

{

pictureBox1.Invalidate();

}

public OurPoint WorldToScreen(OurPoint inputPoint)

{

var point = new OurPoint();

point.X = (inputPoint.X + pictureBox1.Width / 2);

point.Y = -inputPoint.Y + pictureBox1.Height / 2;

return point;

}

private void pictureBox1\_Paint(object sender, PaintEventArgs e)

{

Pen OurPen = new Pen(Color.Black, 1);

Line line1 = new Line(WorldToScreen(new OurPoint(0, -300)), WorldToScreen(new OurPoint(0, 300)));

Line line2 = new Line(WorldToScreen(new OurPoint(-300, 0)), WorldToScreen(new OurPoint(300, 0)));

e.Graphics.DrawLine(OurPen, line1.pointStart.X, line1.pointStart.Y, line1.pointEnd.X, line1.pointEnd.Y);

e.Graphics.DrawLine(OurPen, line2.pointStart.X, line2.pointStart.Y, line2.pointEnd.X, line2.pointEnd.Y);

if (draw)

{

foreach (AbstractObject ob in ListOfObjects)

{

ob.Draw(this, e);

}

}

else

{

foreach (AbstractObject ob in bufferListAbstract)

{

ob.Draw(this, e);

}

}

}

private void toolStripButton1\_Click(object sender, EventArgs e)

{

for (int i = 0; i < bufferListAbstract.Count; i++)

{

if (bufferListAbstract[i].GetType() == typeof(Poliline))

{

Poliline buffer = new Poliline();

buffer = (Poliline)bufferListAbstract[i];

buffer.ToReturn();

}

for (int k = 0; k < 3; k++)

{

bufferListAbstract.RemoveAt(0);

}

draw = true;

}

pictureBox1.Invalidate();

}

private void toolStripButton2\_Click(object sender, EventArgs e)

{

functionMode = 1;

MakeChoice();

}

private void toolStripButton3\_Click(object sender, EventArgs e)

{

functionMode = 2;

MakeChoice();

}

public void MakeChoice()

{

for (int i = 0; i < ListOfObjects.Count; i++)

{

if (ListOfObjects[i].GetType() == typeof(Poliline))

{

Poliline buffer = new Poliline();

Poliline bufferFinal = new Poliline();

buffer = (Poliline)ListOfObjects[i];

for (int j = 0; j < buffer.listOfPoints.Count; j++)

{

bufferFinal.ourArrays.AddRow(new OurPoint(buffer.listOfPoints[j].X, buffer.listOfPoints[j].Y));

bufferFinal.listOfPoints.Add(new OurPoint(buffer.listOfPoints[j].X, buffer.listOfPoints[j].Y));

bufferFinal.StartPointX = buffer.listOfPoints[j].X;

bufferFinal.StartPointY = buffer.listOfPoints[j].Y;

}

draw = false;

switch(functionMode)

{

case 1:

{

bufferFinal.comeBackArray = bufferFinal.ourArrays.ToCopy();

bufferFinal.Increase();

bufferFinal.Move(-80, -150);

bufferListAbstract.Add(bufferFinal);

break;

}

case 2:

{

bufferFinal.comeBackArray = bufferFinal.ourArrays.ToCopy();

bufferFinal.Move(300, -150);

bufferListAbstract.Add(bufferFinal);

break;

}

case 3:

{

bufferFinal.comeBackArray = bufferFinal.ourArrays.ToCopy();

bufferFinal.Rotate(45);

bufferFinal.Move(-150, 280);

bufferListAbstract.Add(bufferFinal);

break;

}

}

if (bufferListAbstract.Count >= 4)

{

for (int k = 0; k < 3; k++)

{

bufferListAbstract.RemoveAt(0);

}

}

}

if (ListOfObjects[i].GetType() == typeof(Circle))

{

Circle buffer = new Circle();

buffer = (Circle)ListOfObjects[i];

Circle bufferFinal = new Circle(new OurPoint(buffer.Center.X, buffer.Center.Y), buffer.Radius, buffer.Angle1, buffer.Angle2);

draw = false;

switch (functionMode)

{

case 1:

{

bufferFinal.Increase();

bufferListAbstract.Add(bufferFinal);

break;

}

case 2:

{

bufferFinal.Move(300, -150);

bufferListAbstract.Add(bufferFinal);

break;

}

case 3:

{

bufferFinal.Rotate(45);

bufferFinal.Move(35, -10);

bufferListAbstract.Add(bufferFinal);

break;

}

}

}

pictureBox1.Invalidate();

}

}

private void toolStripButton4\_Click(object sender, EventArgs e)

{

functionMode = 3;

MakeChoice();

}

public void MakeAllOperations()

{

for (int i = 0; i < ListOfObjects.Count; i++)

{

if (ListOfObjects[i].GetType() == typeof(Poliline))

{

Poliline buffer = new Poliline();

Poliline bufferFinal = new Poliline();

buffer = (Poliline)ListOfObjects[i];

for (int j = 0; j < buffer.listOfPoints.Count; j++)

{

bufferFinal.ourArrays.AddRow(new OurPoint(buffer.listOfPoints[j].X, buffer.listOfPoints[j].Y));

bufferFinal.listOfPoints.Add(new OurPoint(buffer.listOfPoints[j].X, buffer.listOfPoints[j].Y));

bufferFinal.StartPointX = buffer.listOfPoints[j].X;

bufferFinal.StartPointY = buffer.listOfPoints[j].Y;

}

draw = false;

bufferFinal.comeBackArray = bufferFinal.ourArrays.ToCopy();

bufferFinal.Increase();

bufferFinal.Move(200, -300);

bufferFinal.Rotate(45);

bufferFinal.Move(100, 440);

bufferListAbstract.Add(bufferFinal);

}

if (ListOfObjects[i].GetType() == typeof(Circle))

{

Circle buffer = new Circle();

buffer = (Circle)ListOfObjects[i];

Circle bufferFinal = new Circle(new OurPoint(buffer.Center.X, buffer.Center.Y), buffer.Radius, buffer.Angle1, buffer.Angle2);

bufferFinal.Increase();

bufferFinal.Move(310, -180);

bufferListAbstract.Add(bufferFinal);

draw = false;

}

}

if (bufferListAbstract.Count >= 4)

{

for (int i = 0; i < 3; i++)

{

bufferListAbstract.RemoveAt(0);

}

}

pictureBox1.Invalidate();

}

private void toolStripButton5\_Click(object sender, EventArgs e)

{

MakeAllOperations();

}

}