Aufgabe 2: Dreiecksbeziehungen

Teilnahme ID: 48313
Nikolas Kilian

29. April 2019

Inhaltsverzeichnis

1 Lösungsidee

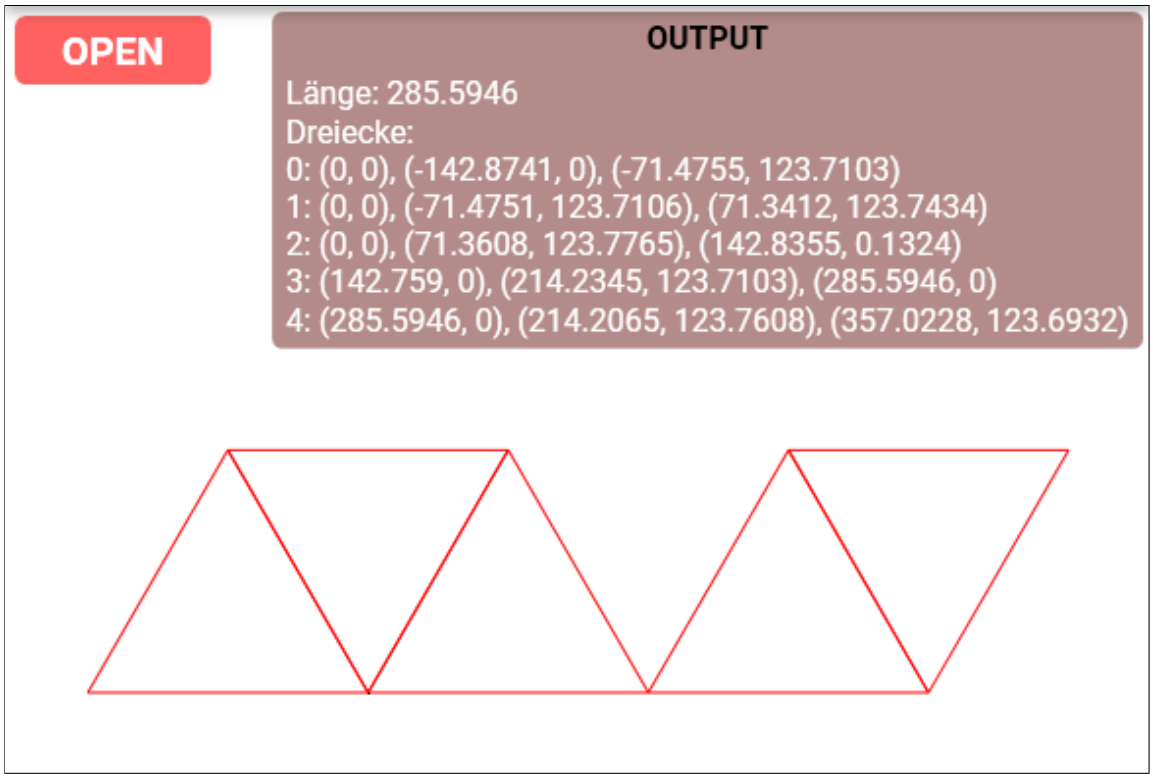
Mein Lösungsansatz ist ein Greedy-Algorithmus, wobei ich schrittweise Dreiecke an eine Liste von Dreiecken anhänge. Jeder Schritt probiert alle noch nicht hinzugefügten Dreiecke in allen ihrer Versionen (Rotation & Spiegelung) anzuhängen, und hängt immer das Dreieck was dabei am wenigsten horizontalen Platz benötigt an.

2 Umsetzung

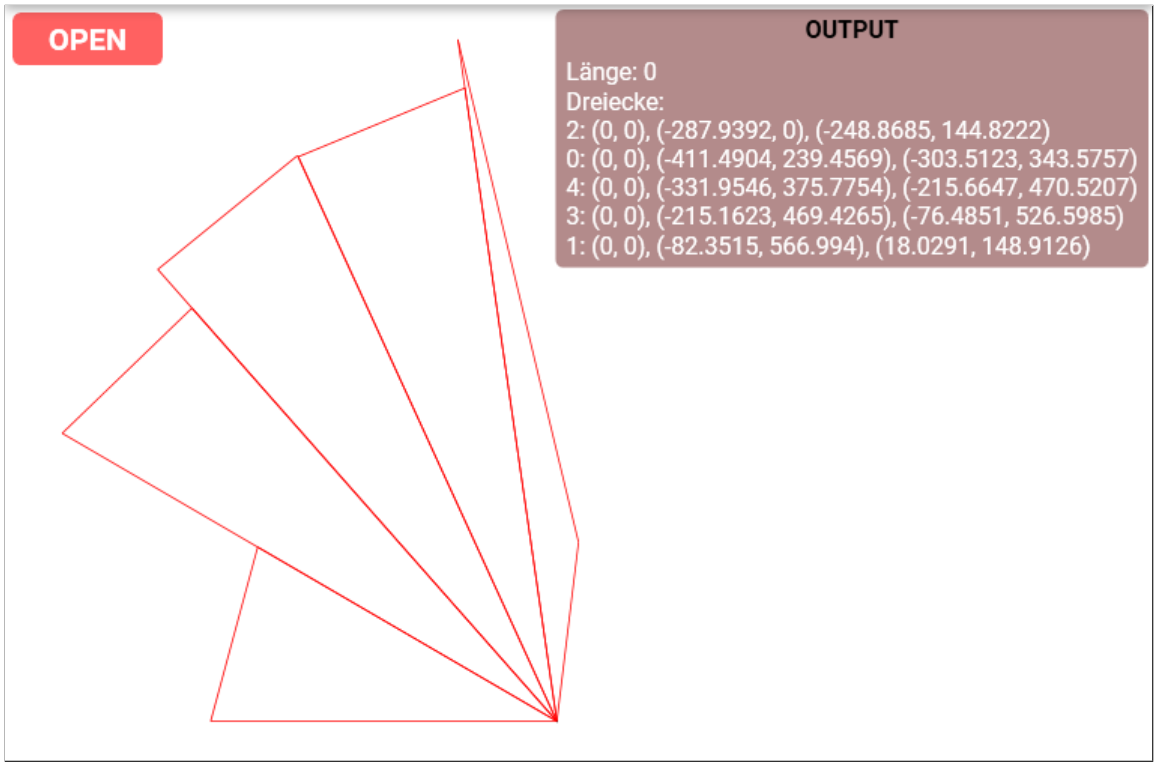
Aufgrund von Zeitproblemen ist meine Umsetzung nicht vollständig
Um das Arbeiten mit Dreiecken leichter zu machen, wandele ich zuerst die Dreiecke in ihre Seitenlängen und Winkel um. Die Umsetzung meiner Lösungsidee ist größtenteils trivial, bis auf das Anhängen von Dreiecken. Zum Anhängen von Dreiecken ermittle ich zuerst die geringste x-Position für die das Dreieck flachliegend rechts angehängt werden kann, daraufhin probiere ich das Dreieck soweit wie möglich nach rechts drehen, indem ich mit immer kleineren Schritten diesen Winkel approximiere.

3 Beispiele

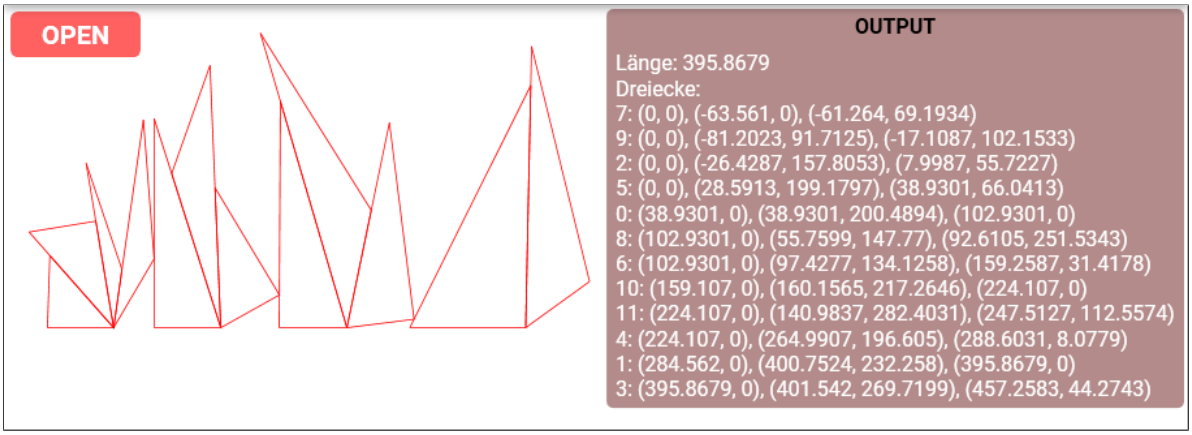
3.1 Beispiel 1



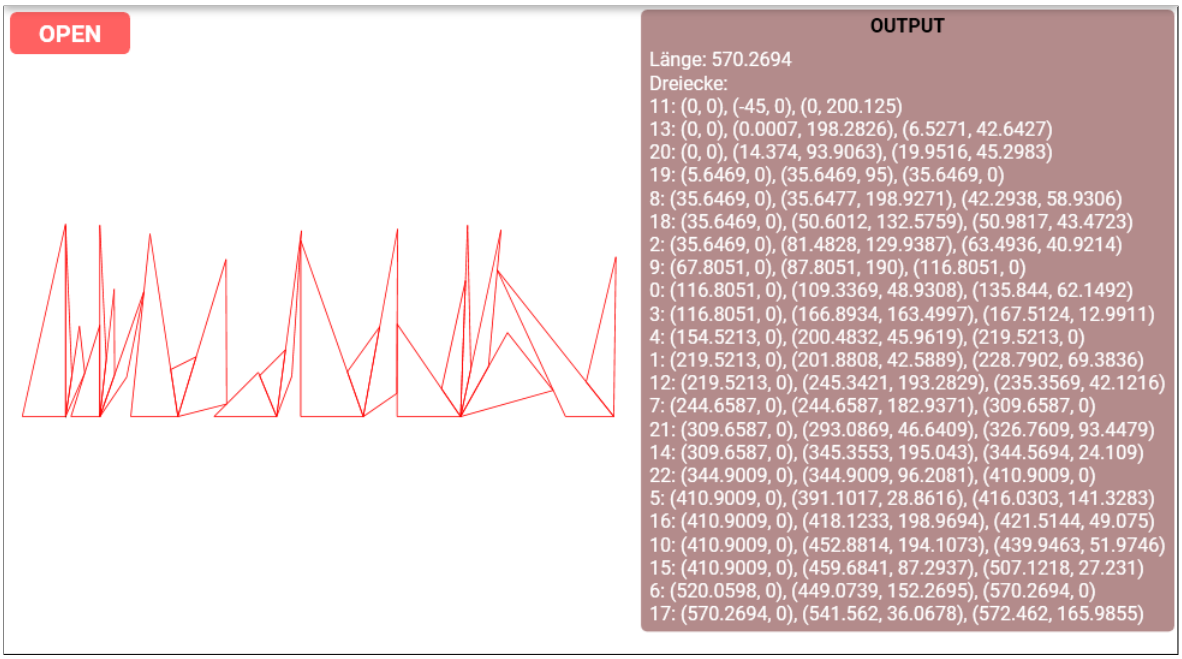
3.2 Beispiel 2



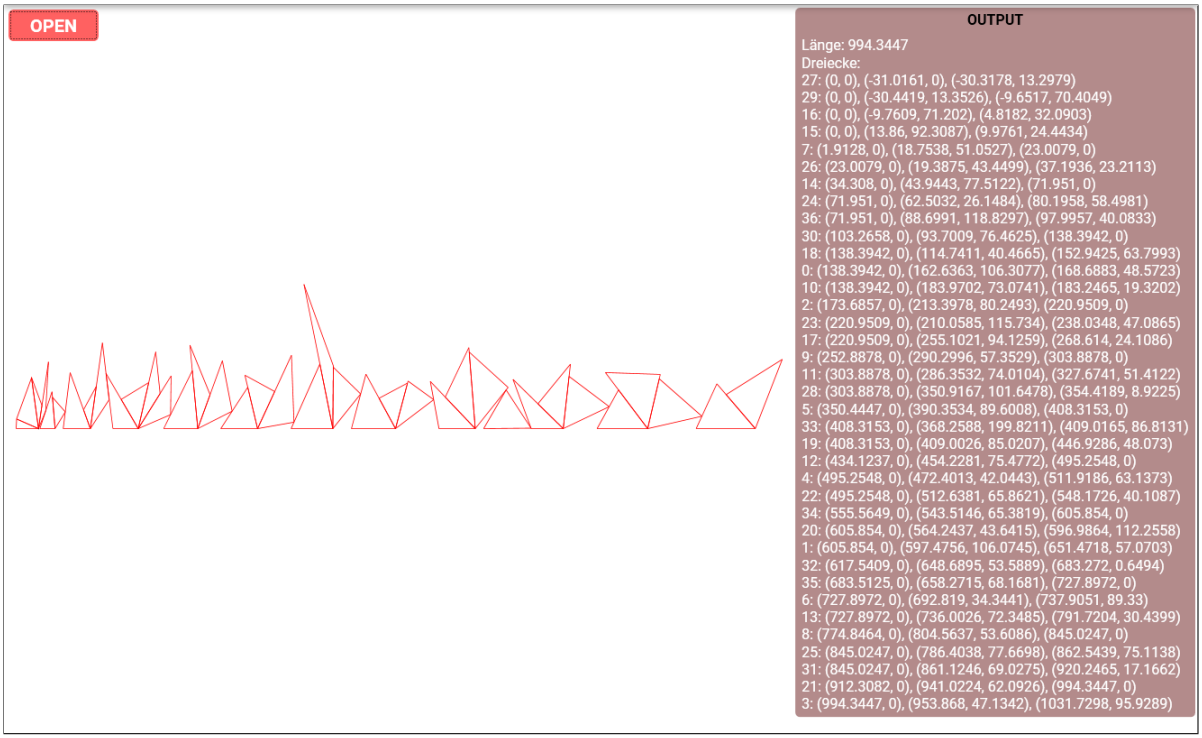
3.3 Beispiel 3



3.4 Beispiel 4



3.5 Beispiel 5



4 Code

Listing 1: class Results

```
output.Text = $@"Länge: {TriangleArranger.SortedLength(triangles).
    ToString("0.####")}
Dreiecke:
{string.Join("\n", triangles.Select(x => $"{order[x]}: ({x.a.x.
    ToString("0.####")}, {x.a.y.ToString("0.####")}), ({x.b.x.ToString
    ("0.####")}, {x.b.y.ToString("0.####")}), ({x.c.x.ToString
    ("0.####")}, {x.c.y.ToString("0.####")})"))}";
```

Listing 2: class Triangle

```
public double[] angles, lengths;

public TriangleArchetype(Triangle triangle)
{
    lengths = new[]
    {
        triangle.a.Distance(triangle.b),
        triangle.b.Distance(triangle.c),
        triangle.c.Distance(triangle.a),
    };

    angles = new[]
    {
        MathHelper.SmallerAngleSide(triangle.a.Angle(triangle.b) -
            triangle.a.Angle(triangle.c)),
```

```
15          MathHelper.SmallerAngleSide(triangle.b.Angle(triangle.a) -
                triangle.a.Angle(triangle.c)),
16          MathHelper.SmallerAngleSide(triangle.c.Angle(triangle.a) -
                triangle.a.Angle(triangle.b)),
17      };
18  }
19
20  public TriangleArchetype Turn(int amount) => this.Let(@this => new
        TriangleArchetype // Lambdas can't use this => pass it as an
        argument
21  {
22      angles = Enumerable.Range(0, 3).Select(z => @this.angles[(z +
            amount) % 3]).ToArray(),
23      lengths = Enumerable.Range(0, 3).Select(z => @this.lengths[(z +
            amount) % 3]).ToArray()
24  });
25  public TriangleArchetype Mirror() => this.Let(@this => new
        TriangleArchetype
26  {
27      angles = Enumerable.Range(0, 3).Select(z => @this.angles[(3 - z)
            % 3]).ToArray(),
28      lengths = Enumerable.Range(0, 3).Select(z => @this.lengths[(3 - z
            ) % 3]).ToArray()
29  });
```

Listing 3: class Triangle

```
1   public Vector a, b, c;
2   public Vector this[int index] => MathHelper.PositiveModulo(index, 0,
        3).Let(x =>
3           x == 0 ? a
4       : x == 1 ? b
5       : x == 2 ? c
6       : throw new InvalidOperationException());
7
8   public Triangle(TriangleArchetype archetype, Vector positionOffset,
        double angleOffset)
9   {
10      a = positionOffset;
11      b = a + new Vector(archetype.angles[0] + angleOffset) * archetype
            .lengths[0];
12      c = a + new Vector(angleOffset) * archetype.lengths[2];
13  }
14
15  public bool Intersects(Triangle other)
16  {
17      var edges = new[] { (a, b), (b, c), (c, a) };
18      var otherEdges = new[] { (other.a, other.b), (other.b, other.c),
            (other.c, other.a) };
19      return edges.Any(x => otherEdges.Any(y => Vector.
            IntersectingLines(x.Item1, x.Item2, y.Item1, y.Item2)));
20  }
21
22  public bool Surrounds(Vector other, double epsilon) =>
23      (Vector.OrientationApprox(a, b, other, epsilon), Vector.
            OrientationApprox(b, c, other, epsilon), Vector.
            OrientationApprox(c, a, other, epsilon))
24      .Let(x => x.Item1 == x.Item2 && x.Item2 == x.Item3 && x.Item1 !=
```

```
            Vector.VectorOrder.Collinear);
```

Listing 4: static class TriangleArranger

```
public static List<Triangle> ArrangeTriangles(in List<TriangleArchetype> triangleArchetypesIn, out Dictionary<Triangle, int> order)
{
    var orderOut = new Dictionary<Triangle, int>();
    var triangles = new List<Triangle>();

    Triangle last;
    triangles.Add(last = new Triangle(new Vector(-1, 0), new Vector(-1, 0), new Vector(0, 0)));

    List<TriangleArchetype> triangleArchetypes = new List<TriangleArchetype>(triangleArchetypesIn);

    int n = 0;
    while (triangleArchetypes.Any())
    {
        var (value, comparable) = triangleArchetypes
            .SelectMany(x => new[] { (x, x), (x, x.Mirror()), (x, x.Turn(1)), (x, x.Turn(1).Mirror()), (x, x.Turn(2)), (x, x.Turn(2).Mirror()) })
            .Select(x => (x.Item1, AddTriangle(x.Item2, new List<Triangle>(triangles))))
            .MinValue(x => Math.Max(x.Item2.a.x, Math.Max(x.Item2.b.x, x.Item2.c.x)) - Math.Min(x.Item2.a.x, Math.Min(x.Item2.b.x, x.Item2.c.x)));

        triangles.Add(last = value.Item2);
        triangleArchetypes.Remove(value.Item1);
        orderOut[value.Item2] = triangleArchetypesIn.IndexOf(value.Item1);
    }

    order = orderOut;
    return triangles.Skip(1).ToList();
}

public static double epsilon = 1E-10;

public static Triangle AddTriangle(TriangleArchetype toAdd, List<Triangle> triangles)
{
    Triangle added;

    Vector upper = new Vector(toAdd.angles[0]) * toAdd.lengths[0];
    double rise = upper.x / upper.y;

    double maxX = double.NegativeInfinity;

    foreach ((Vector start, Vector end) in triangles.SelectMany(x => new[] { (x.a, x.b), (x.b, x.c), (x.c, x.a) }))
    {
        if (start.y.Approx(end.y, epsilon)) maxX = Math.Max(maxX, Math.Max(start.x, end.x));
```

```
42          else
43          {
44              double startX = start.x - rise * start.y;
45              double endX = end.x - rise * end.y;
46              double lambda = MathHelper.Clamp((upper.y - start.y) / (
                    end.y - start.y), 0, 1);
47              maxX = Math.Max(maxX, Math.Max(startX * (1 - lambda) +
                    endX * lambda, end.y > start.y ? startX : endX));
48          }
49      }
50
51      added = null; // Avoid unassigned compilation errors
52                    // Use small steps to approximate max angle
53      double lastSuccess = 0, lastFailure = Math.PI - toAdd.angles[0];
54      for (double angleCurrent = lastFailure; Math.Abs(lastSuccess -
            lastFailure) > 1E-5; angleCurrent = (lastSuccess + lastFailure
            ) / 2d)
55      {
56          added = new Triangle(toAdd, new Vector(maxX, 0), angleCurrent
                );
57          // Check for intersections
58          if (triangles.Any(x => x.Intersects(added)
59          || x.Surrounds(added.a, epsilon) || x.Surrounds(added.b,
                epsilon) || x.Surrounds(added.c, epsilon)
60          || added.Surrounds(x.a, epsilon) || added.Surrounds(x.b,
                epsilon) || added.Surrounds(x.c, epsilon)))
61          {
62              lastFailure = angleCurrent;
63          }
64          else
65          {
66              lastSuccess = angleCurrent;
67          }
68      }
69      added = new Triangle(toAdd, new Vector(maxX, 0), lastSuccess);
70
71      return added;
72  }
```