

Aufgabe 2: Dreiecksbeziehungen

Teilnahme ID: 48313

Nikolas Kilian

29. April 2019

Inhaltsverzeichnis

1	Lösungsidee	1
2	Umsetzung	1
3	Beispiele	2
3.1	Beispiel 1	2
3.2	Beispiel 2	2
3.3	Beispiel 3	3
3.4	Beispiel 4	3
3.5	Beispiel 5	4
4	Code	4

1 Lösungsidee

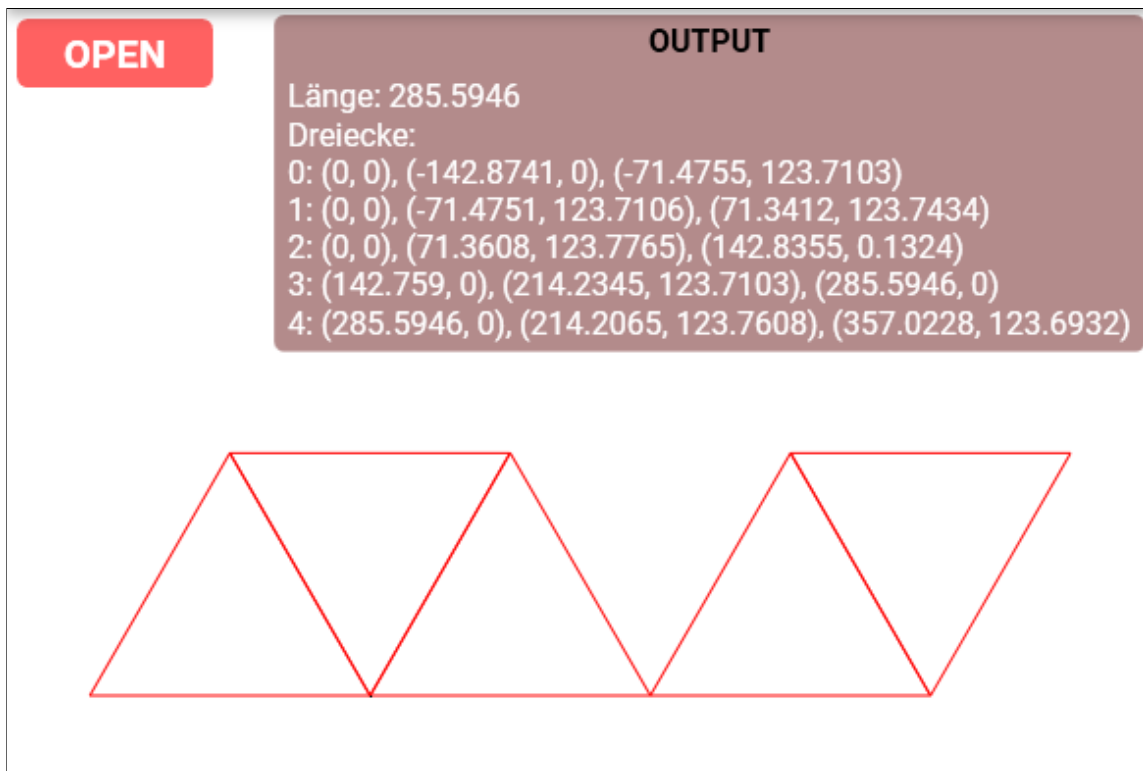
Mein Lösungsansatz ist ein Greedy-Algorithmus, wobei ich schrittweise Dreiecke an eine Liste von Dreiecken anhänge. Jeder Schritt probiert alle noch nicht hinzugefügten Dreiecke in allen ihrer Versionen (Rotation & Spiegelung) anzuhängen, und hängt immer das Dreieck was dabei am wenigsten horizontalen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

2 Umsetzung

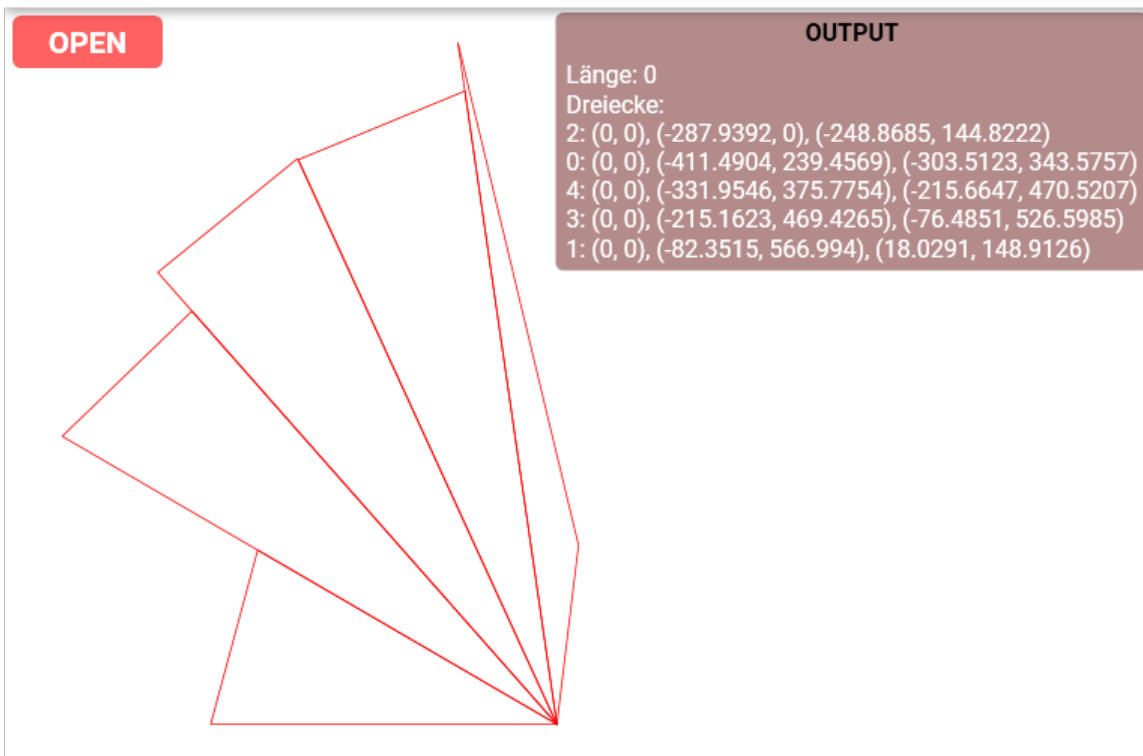
Aufgrund von Zeitproblemen ist meine Umsetzung nicht vollständig Um das Arbeiten mit Dreiecken leichter zu machen, wandle ich zuerst die Dreiecke in ihre Seitenlängen und Winkel um. Die Umsetzung meiner Lösungsidee ist größtenteils trivial, bis auf das Anhängen von Dreiecken. Zum Anhängen von Dreiecken 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

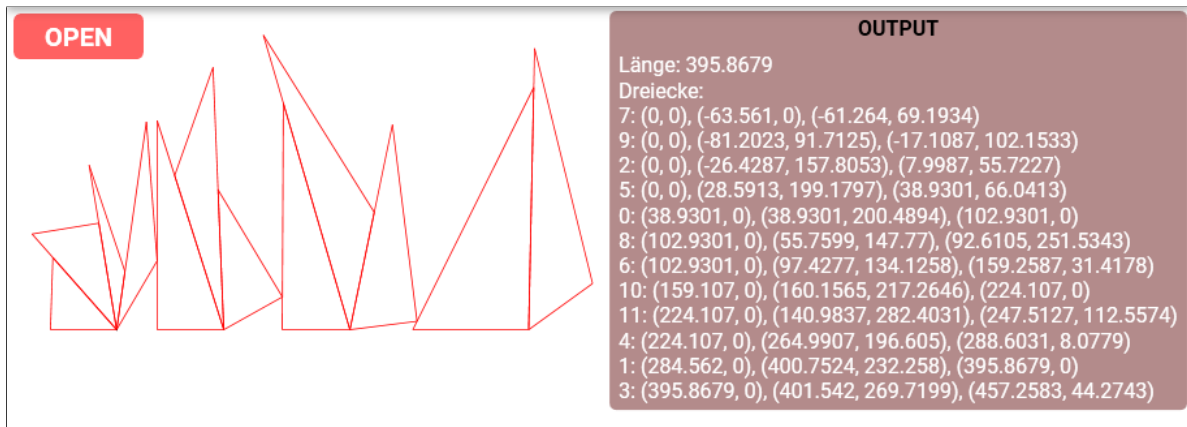
3.1 Beispiel 1



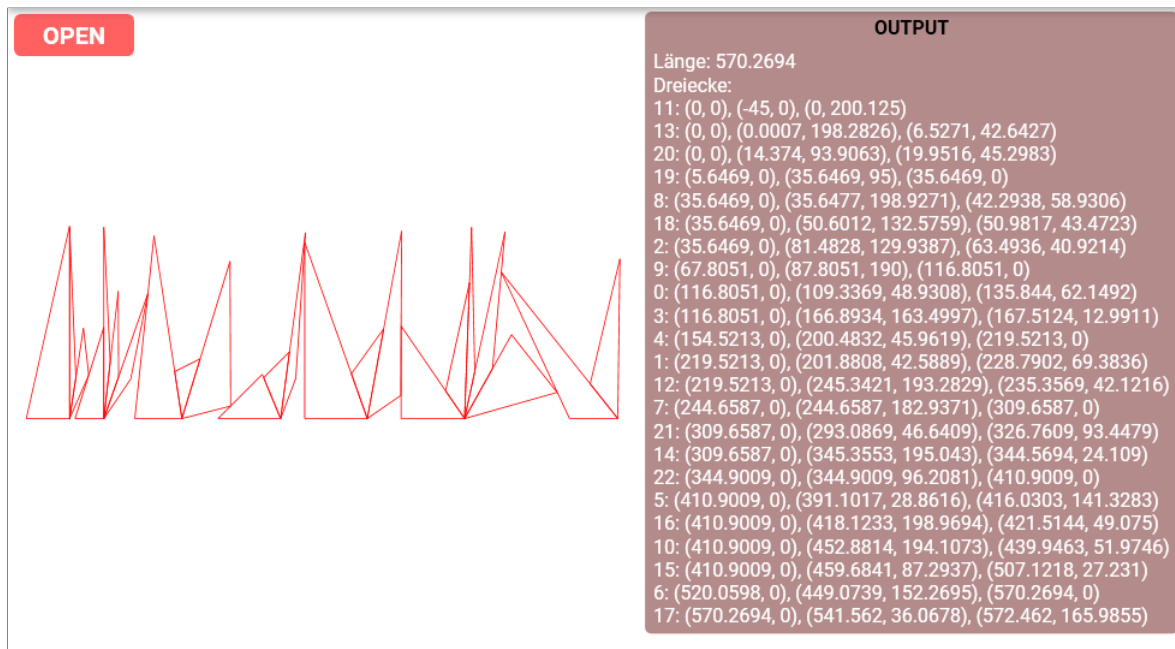
3.2 Beispiel 2



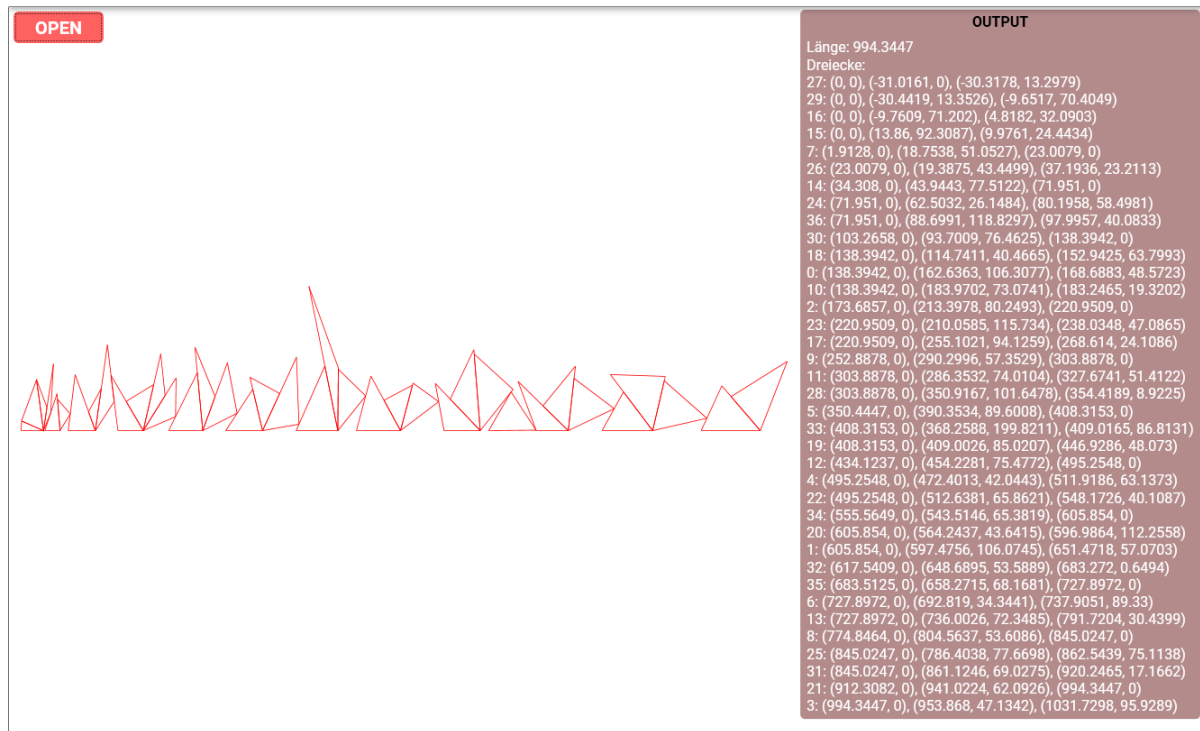
3.3 Beispiel 3



3.4 Beispiel 4



3.5 Beispiel 5



4 Code

Listing 1: class Results

```

1 output.Text = $"{Länge: {TriangleArranger.SortedLength(triangles).
  ToString("0.####")}}
2 Dreiecke:
3 {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

```

1 public double[] angles, lengths;
2
3 public TriangleArchetype(Triangle triangle)
4 {
5     lengths = new[]
6     {
7         triangle.a.Distance(triangle.b),
8         triangle.b.Distance(triangle.c),
9         triangle.c.Distance(triangle.a),
10    };
11
12    angles = new[]
13    {
14        MathHelper.SmallerAngleSide(triangle.a.Angle(triangle.b) -
          triangle.a.Angle(triangle.c)),

```

```
15     MathHelper.SmallerAngleSide(triangle.b.Angle(triangle.a) -
16         triangle.a.Angle(triangle.c)),
17     MathHelper.SmallerAngleSide(triangle.c.Angle(triangle.a) -
18         triangle.a.Angle(triangle.b)),
19 };
20 }
21
22 public TriangleArchetype Turn(int amount) => this.Let(@this => new
23     TriangleArchetype // Lambdas can't use this => pass it as an
24     argument
25 {
26     angles = Enumerable.Range(0, 3).Select(z => @this.angles[(z +
27         amount) % 3]).ToArray(),
28     lengths = Enumerable.Range(0, 3).Select(z => @this.lengths[(z +
29         amount) % 3]).ToArray()
30 });
31
32 public TriangleArchetype Mirror() => this.Let(@this => new
33     TriangleArchetype
34 {
35     angles = Enumerable.Range(0, 3).Select(z => @this.angles[(3 - z)
36         % 3]).ToArray(),
37     lengths = Enumerable.Range(0, 3).Select(z => @this.lengths[(3 - z)
38         % 3]).ToArray()
39 });
40 };
```

Listing 3: class Triangle

```
1 public Vector a, b, c;
2 public Vector this[int index] => MathHelper.PositiveModulo(index, 0,
3     3).Let(x =>
4     x == 0 ? a
5     : x == 1 ? b
6     : x == 2 ? c
7     : throw new InvalidOperationException());
8
9 public Triangle(TriangleArchetype archetype, Vector positionOffset,
10     double angleOffset)
11 {
12     a = positionOffset;
13     b = a + new Vector(archetype.angles[0] + angleOffset) * archetype
14         .lengths[0];
15     c = a + new Vector(archetype.angles[2] + angleOffset) * archetype
16         .lengths[2];
17 }
18
19 public bool Intersects(Triangle other)
20 {
21     var edges = new[] { (a, b), (b, c), (c, a) };
22     var otherEdges = new[] { (other.a, other.b), (other.b, other.c),
23         (other.c, other.a) };
24     return edges.Any(x => otherEdges.Any(y => Vector.
25         IntersectingLines(x.Item1, x.Item2, y.Item1, y.Item2)));
26 }
27
28 public bool Surrounds(Vector other, double epsilon) =>
29     (Vector.OrientationApprox(a, b, other, epsilon), Vector.
30         OrientationApprox(b, c, other, epsilon), Vector.
31         OrientationApprox(c, a, other, epsilon))
32     .Let(x => x.Item1 == x.Item2 && x.Item2 == x.Item3 && x.Item1 !=
```

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

Listing 4: static class TriangleArranger

```
1 public static List<Triangle> ArrangeTriangles(in List<
    TriangleArchetype> triangleArchetypesIn, out Dictionary<Triangle,
    int> order)
2 {
3     var orderOut = new Dictionary<Triangle, int>();
4     var triangles = new List<Triangle>();
5
6     Triangle last;
7     triangles.Add(last = new Triangle(new Vector(-1, 0), new Vector
    (-1, 0), new Vector(0, 0)));
8
9     List<TriangleArchetype> triangleArchetypes = new List<
    TriangleArchetype>(triangleArchetypesIn);
10
11     int n = 0;
12     while (triangleArchetypes.Any())
13     {
14         var (value, comparable) = triangleArchetypes
15             .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()) })
16             .Select(x => (x.Item1, AddTriangle(x.Item2, new List<
                Triangle>(triangles))))
17             .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)));
18
19         triangles.Add(last = value.Item2);
20         triangleArchetypes.Remove(value.Item1);
21         orderOut[value.Item2] = triangleArchetypesIn.IndexOf(value.
            Item1);
22     }
23
24     order = orderOut;
25     return triangles.Skip(1).ToList();
26 }
27
28 public static double epsilon = 1E-10;
29
30 public static Triangle AddTriangle(TriangleArchetype toAdd, List<
    Triangle> triangles)
31 {
32     Triangle added;
33
34     Vector upper = new Vector(toAdd.angles[0]) * toAdd.lengths[0];
35     double rise = upper.x / upper.y;
36
37     double maxX = double.NegativeInfinity;
38
39     foreach ((Vector start, Vector end) in triangles.SelectMany(x =>
        new[] { (x.a, x.b), (x.b, x.c), (x.c, x.a) }))
40     {
41         if (start.y.Approx(end.y, epsilon)) maxX = Math.Max(maxX,
            Math.Max(start.x, end.x));
42     }
43 }
```

```
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) / (
47             end.y - start.y), 0, 1);
48         maxX = Math.Max(maxX, Math.Max(startX * (1 - lambda) +
49             endX * lambda, end.y > start.y ? startX : endX));
50     }
51     }
52     added = null; // Avoid unassigned compilation errors
53     // Use small steps to approximate max angle
54     double lastSuccess = 0, lastFailure = Math.PI - toAdd.angles[0];
55     for (double angleCurrent = lastFailure; Math.Abs(lastSuccess -
56         lastFailure) > 1E-5; angleCurrent = (lastSuccess + lastFailure
57         ) / 2d)
58     {
59         added = new Triangle(toAdd, new Vector(maxX, 0), angleCurrent
60             );
61         // Check for intersections
62         if (triangles.Any(x => x.Intersects(added)
63             || x.Surrounds(added.a, epsilon) || x.Surrounds(added.b,
64             epsilon) || x.Surrounds(added.c, epsilon)
65             || added.Surrounds(x.a, epsilon) || added.Surrounds(x.b,
66             epsilon) || added.Surrounds(x.c, epsilon)))
67         {
68             lastFailure = angleCurrent;
69         }
70         else
71         {
72             lastSuccess = angleCurrent;
73         }
74     }
75     added = new Triangle(toAdd, new Vector(maxX, 0), lastSuccess);
76     return added;
77 }
```