Steffen's flexible polyhedron

A preprint version of a “Mathematical graphics” column from

Mathematica in Education and Research.

Mark McClure

Department of Mathematics
University of North Carolina at Asheville
Asheville, NC 28804

mcmcclure@unca.edu

Abstract

- **Initialization**

1. Introduction

> *A closed spatial figure allows no changes as long as it is not ripped apart.*
> Leonard Euler, as quoted in [1], page 241.

Euler's above sentiment expresses a long-standing, unquestioned, and perhaps natural belief that all polyhedra are rigid. Even if constructed with hinged edges Euler states, a polyhedron will not bend or flex without distorting the faces. The statement is true of *convex* polyhedra. In fact, Cauchy proved the stronger statement that any two combinatorially equivalent convex polyhedra with congruent faces are themselves congruent. (For an elementary proof of Cauchy's theorem, as well as a nice introduction to this topic, see chapter 6 of [1].)

However, there are non-convex polyhedra which do in fact flex. In this issue's column, we construct a model of the simplest such flexible polyhedron discovered by Klaus Steffen. Although Steffen does not seem to have published this result, it has become well known and often cited in the literature of rigidity of structures. In addition to construction of the model, we investigate the range of genuine non-intersecting flexibility.

2. The construction

Our construction is based on the following net which describes how to make a model of the polyhedron out of paper by cutting and folding. Note that solid lines should be thought of as “mountain folds” and dotted lines as “valley folds”. (The **`SteffenNet`** command is defined in this notebook's initialization cells, as are several related functions.)

```
SteffenNet[ShowEdgeLengths → True,
   ShowBoundaryLabels → True,
   ShowVertexLabels → True] // Show;
```

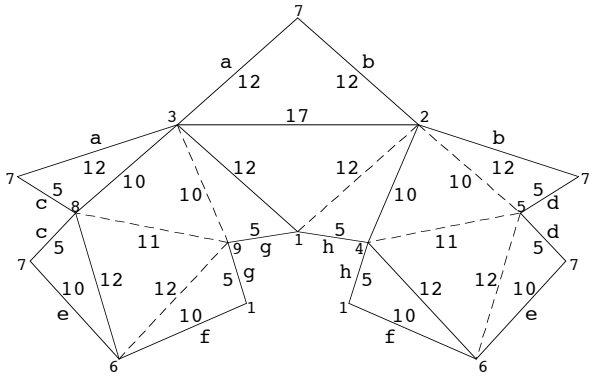


Fig. 1 A net for Steffen's polyhedron

Note that each edge is labelled with its length. Edges on the boundary are labelled with letters and edges with matching labels should match up when the model is folded. Similarly, the vertices are labelled with numbers in such a way that the numbers should match correctly. The resulting polyhedron has 14 triangular faces, 21 edges, and 9 vertices.

Using this net, it is now fairly straightforward to set up a system of equations to determine the 3D locations of the vertices for the folded model. We begin by choosing three points p_1, p_2, and p_3 that satisfy the distance relations described in the diagram, but which are otherwise arbitrary.

```
p1 = {0, 0, 0};
p2 = {-12, 0, 0};
p3 = {1 / 24, -17 Sqrt[287] / 24, 0};
```

Now, since the polyhedron flexes, there will be a degree of flexibility in choosing the next point p_4. In fact, the polyhedron has one degree of freedom, so we will need one independent parameter to determine the coordinates of p_4. Note that p_1 and p_2 both lie on the x-axis and that p_4 will be rotated at some angle θ about that axis. Thus we might suppose that the coordinates of p_4 are given by

```
p4 = {x, r Cos[θ], r Sin[θ]};
```

Note that θ represents the angle between the face bound by p_1, p_2, and p_3 and the face bound by p_1, p_2, and p_4.

Next, we need to set up two equations describing the distance relations between p_4 to p_1 and p_4 to p_2. We can then solve those equations for the two unknowns r and x to determine the location of p_4. These, and subsequent, equations will be set up using the **mag2** function, which stands for “magnitude squared”. Of course, we'll need a specific value of θ to generate an actual picture. For the purposes of this example, we choose $\theta = \pi / 12$.

```
θ = Pi / 12; mag2 = #.# &;
conditions1 = {mag2[p1 - {x, r Cos[θ], r Sin[θ]}] == 25,
   mag2[p2 - {x, r Cos[θ], r Sin[θ]}] == 100};
sols = FindRoot[conditions1, {r, 4}, {x, -3}];
p4 = p4 /. sols;
```

The next 3 points must satisfy the following set of 9 equations in 9 unknowns.

```
p5 = {x5, y5, z5}; p6 = {x6, y6, z6}; p7 = {x7, y7, z7};
conditions2 = {mag2[p5 - p4] == 121,
```

```
    mag2[p5 - p2] == 100, mag2[p6 - p4] == 144,
    mag2[p6 - p1] == 100, mag2[p7 - p2] == 144,
    mag2[p7 - p3] == 144, mag2[p5 - p6] == 144,
    mag2[p6 - p7] == 100, mag2[p7 - p5] == 25};
sols = FindRoot[conditions2, {x5, -5}, {y5, -6},
    {z5, -3.5}, {x6, 3.5}, {y6, 0.1}, {z6, -9},
    {x7, -5}, {y7, -5}, {z7, -8}];
{p5, p6, p7} = {p5, p6, p7} /. sols;
```

Finally, the eighth and ninth vertices are fairly simple to find.

```
p8 = {x8, y8, z8};
conditions3 = {mag2[p8 - p3] == 100,
    mag2[p8 - p6] == 144, mag2[p8 - p7] == 25};
sols = FindRoot[conditions3, {x8, -3.4},
    {y8, -9.7}, {z8, -9.1}];
p8 = p8 /. sols;
p9 = {x9, y9, z9};
conditions4 = {mag2[p9 - p1] == 25,
    mag2[p9 - p3] == 100, mag2[p9 - p6] == 144};
sols = FindRoot[conditions4, {x9, -4},
    {y9, -3}, {z9, -.5}];
p9 = p9 /. sols;
```

We now have the coordinates of the of the nine vertices and we need to transform this into a graphic. More than this, we will be interested in performing a number of computational operations on the polyhedron, so it makes sense to store the polyhedron in an appropriate data structure. Considering this, we'll follow the lead of Byrge Birkeland who described a very natural data structure to store a polyhedron in this journal [2]. Our polyhedron (or polytope, as Birkeland calls it) will have the following form.

```
polytopePattern = {v : {{_?NumericQ, _?NumericQ, _?NumericQ} ...},
    f : {{__Integer?Positive} ..}} /; Max[Flatten[f]] ≤ Length[v];
```

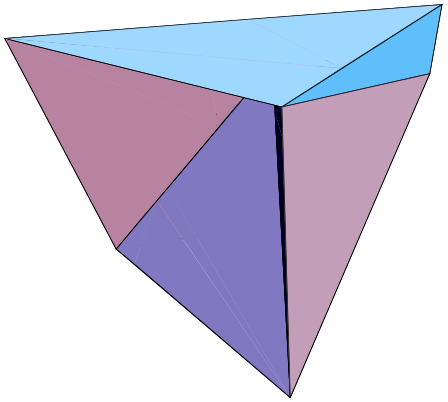
Note that **v** is the list of vertices and **f** is a list of faces; each face is a list of integers indicating which vertices of **v** are the vertices of the face. Thus we can represent Steffen's polyhedron as follows.

```
steffenPolyhedron = {
    {p1, p2, p3, p4, p5, p6, p7, p8, p9},
    {{1, 2, 3}, {7, 3, 2}, {1, 4, 2}, {2, 4, 5},
     {2, 5, 7}, {1, 6, 4}, {4, 6, 5}, {5, 6, 7},
     {6, 8, 7}, {6, 9, 8}, {1, 9, 6}, {3, 7, 8},
     {3, 8, 9}, {1, 3, 9}}};
MatchQ[steffenPolyhedron, polytopePattern]
```

```
True
```

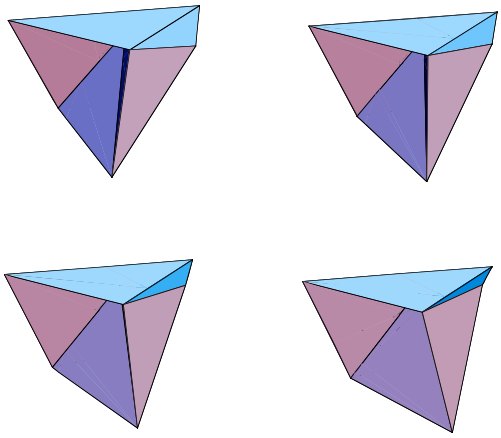
Now it's a straightforward matter to convert this into a **Graphics3D** object and display it.

```
PolytopeToGraphics3D[{v_, f_} /; MatchQ[{v, f}, polytopePattern],
  opts___] := Graphics3D[Polygon[v[[#]]] & /@ f, opts,
  ViewPoint → {2.594, 1.647, 1.418}, Boxed → False];
Show[PolytopeToGraphics3D[steffenPolyhedron]];
```



These commands have all been encapsulated in the **SteffenPolyhedron** command defined in the initialization cells. This makes it easy to create a list of graphics to generate an animation.

```
Show[GraphicsArray[Partition[Table[PolytopeToGraphics3D[
      SteffenPolyhedron[θ]],
     {θ, Pi / 60, Pi / 6, Pi / 20}], 2]]];
```



Several questions now arise naturally. Perhaps the most basic is, how can we generate better images? For example, how can we make an animated image that we can grab and rotate to see various viewpoints? It seems that the best tool for this is currently JavaView [3]. Assuming you have JavaView installed on your machine, the following commands will open an interactive animation of the flexing polyhedron which you can rotate with a mouse in a separate window. (Note that these cells are set to non-evaluatable to prevent problems on machines where JavaView is not present.)

```
Needs["JavaView`JLink`"];
viewer = InstallJavaView[];
```

```
gs = Table[PolytopeToGraphics3D[SteffenPolyhedron[t],
    PlotRange → {{-12., 5}, {-12, 5}, {-10, 4}},
    PolygonIntersections → False],
   {t, Pi / 60, Pi / 6, Pi / 20}];
JavaView[gs, Animatable → True];
```

We won't go into anymore details here, since JavaView's integration with *Mathematica* is well documented on its homepage. I have also set up a pre-generated JavaView animation on my *Mathematica* graphics webpage: http://facstaff.unca.edu/mcmcclur/mathematicaGraphics/SteffenPolyhedron/.

3. Testing for intersection points

Looking at Steffen's polyhedron from several different viewpoints, it is not easy to tell which values of θ lead to a genuine non-intersecting polyhedron. To help determine this, we've defined a function **NonIntersectingQ** in the initialization cells. This command accepts a polyhedron with triangular faces and returns either **True** or **False**.

The concepts behind **NonIntersectingQ** are fairly basic techniques in three-dimensional computational geometry. For example, we can use vector algebra to write a simple command to check if the point p_0 is on the line segment $[p_1, p_2]$ as follows.

```
PointOnSegmentQ[p0 : {x0_, y0_, z0_},
   seg : {p1 : {x1_, y1_, z1_}, p2 : {x2_, y2_, z2_}}] := Module[
   {v1, v2},
   v1 = p1 - p0;
   v2 = p2 - p0;
   v1.v2 == -Sqrt[(v1.v1) (v2.v2)]];
```

We can then use this to check if two segments intersect, which can then be used to check if two triangles intersect, which can then be used to check if a triangulated polyhedron has points of intersection.. The code is a bit tedious, as many different cases need to be considered; it is all contained in an initialization cell. Note that this code based on ideas in Joseph O' Rourke's outstanding text on computational geometry [4].

Using **NonIntersectingQ** we can now find a nice range of θ values for which Steffen's polyhedron is non-intersecting.

```
Table[NonIntersectingQ[SteffenPolyhedron[t]],
 {t, Pi / 60, Pi / 6, Pi / 60}]
```

```
{True, True, True, True, True, True, True, True, True, True}
```

We can generate a very nice image to see how close the polyhedron is to intersection using JavaView's transparency option.

```
g = PolytopeToGraphics3D[SteffenPolyhedron[Pi / 60]];
JavaView[g, Transparency → .5];
```

4. Volume

Finally, there is a lovely result stating that the volume of a polyhedron which does flex must maintain a constant volume throughout the flex. This was conjectured by Bob Connelly in the 1970s and became known as the bellows conjecture, since it states that such a polyhedron would not make a good bellows.

The following formula computes the volume of a polyhedron with triangular faces in terms of the vertices. Note that the vertices of the faces are presumed to be listed in a common orientation.

```
Volume[{v : {{_?NumericQ, _?NumericQ, _?NumericQ} ...},
    f : {{_Integer?Positive, _Integer?Positive, _Integer?Positive} ..}} /;
   Max[Flatten[f]] ≤ Length[v]] := Module[
  {contrib},
  contrib[{{x0_, y0_, z0_}, {x1_, y1_, z1_}, {x2_, y2_, z2_}}] :=
   (z0 + z1 + z2) (x0 (y1 - y2) + x1 (y2 - y0) + x2 (y0 - y1)) / 6;
  Abs[Plus @@ (contrib[v[[#]]] & /@ f)]
 ];
```

Let's test the bellows conjecture on Steffen's polyhedron. Note that it is strangely valid even for self-intersecting polyhedra.

```
Volume[SteffenPolyhedron[#]] & /@ Range[0, Pi / 4, Pi / 20.]
```

```
{200.777, 200.777, 200.777, 200.777, 200.777, 200.777}
```

References

1. Cromwell, P. *Polyhedra*. Cambridge University Press. Cambridge, UK 1997.

2. Birkeland, B. Three Dimensional Polytopes *Mathematica in Education and Research*. **10** (2) 2005

3. JavaView - http://www.javaview.de/.

4. O'Rourke. *Computational Geometry in C (2nd Ed)*. Cambridge University Press. Cambridge, UK 1998.