Geometric Evolution on Computationally Abstract Manifolds

Alex Henniges
Thomas Williams
Mitch Wilson

University of Arizona Undergraduate Research Program
Supervisor: Dr. David Glickenstein

July 7, 2008

1 Introduction

Over the course of this summer, we examined and learned about various topological and geometric manipulations across various dimensional objects. We use a lot of C++ to code our work, and have a lot of work to show for it. In this section we will introduce the concepts behind our project, as well as discuss our plan of action. Things we ought to put here:

- Definitions, like triangulations and such
- A basic verbal-type background of what we are doing, not too many equations
- Our aspect of what we are doing
- Our interpretation of the end goal
- What we aim to accomplish over the course of the summer

1.1 Introduction to triangulations and circle packing

Suppose you are asked to make a sphere, but are only given a small number of edges, vertices, or faces to work with. What do you do? In geometry, a common question asked is how to make representations of complex shapes with as little stuff as possible. In fact, you are probably already familiar with the most basic representation of a sphere- a tetrahedron. Using only four vertices, six edges, and four faces, the tetrahedron is able to give us an approximation to a sphere. Similarly, three vertices (a triangle) can be used to approximate a circle in two dimesnions. Naturally, if we add more vertices, we are able to better illustrate our shapes.

Of course, spheres aren't the only things geometers care about. Another shape of interest is known as the torus. It looks like a donut. It turns out that you can make a visualization of this object using only 7 vertices [5]. With more vertices the shape becomes better curved and reminiscent of breakfast. Mmm, torus. It is also possible to add "handles" to any shape, resulting in the addition of another torus. There are also various other shapes which can be represented with varying numbers of vertices, like Klein bottles and cross-caps.[6] has more information on these fascinating shapes.

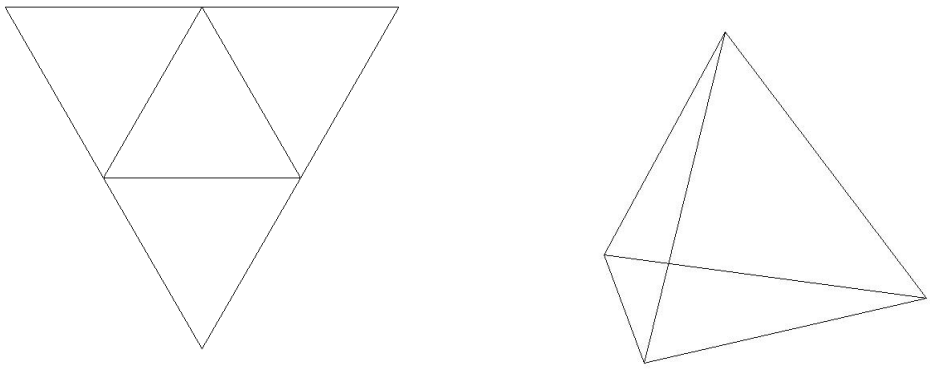

Figure 1: An example of triangulation. A triangle can be folded up into a tetrahedron.

Like in modern video games and Hollywood movies, we can generate various shapes of many different sizes using polygons that mold to form the special effect. For these and all shapes, we will focus on building them solely out of triangles. Since any regular polygon can be broken up into smaller triangles, we can essentially represent any shape with enough vertices. We define a shape as *triangulizable* if we can connect all triangles in a particular fashion such that they create a closed 3-dimenional shape.

An issue arises regarding the uniqueness of these triangulations. There are over 28 *trillion* ways to triangulate a sphere using only 23 vertices. It is currently unknown how many ways a single torus can be made using that number of vertices (to us, at least). However, all toruses do share something very peculiar, as do all sphere configurations. The Euler characteristic, χ, is a value associated with three dimensional shapes. It is defined as $\chi = V - E + F$, where V, E, and F are the number of vertices, edges, and faces a given manifold has. Certain shapes have the same χ value. For example, any torus has $\chi = 0$, and any representation of a sphere has $\chi = 2$. Table 1 has listings for many common shapes.

While we can indicate how these different vertices are connected to each other, there is still an issue regarding the lengths of these edges. There are an infinite number of possible edge length configurations just as there are an infinite number of possible triangles. We introduce a method known as circle

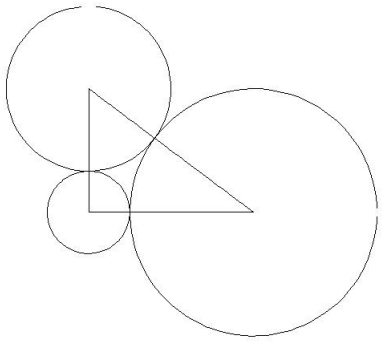

Figure 2: An example of circle packing. Three mutually tangent circes forming the perimeter of a triangle.

Name	Vertices, V	Edges, E	Faces, F	$\chi, V - E + F$
Tetrahedron	4	6	4	2
Hexahedron or cube	8	12	6	2
Octahedron	6	12	8	2
Dodecahedron	20	30	12	2
Icosahedron	12	30	20	2
Torus	9	27	18	0
Double torus	10	36	24	-2

Table 1: Listings of vertices, edges, and faces for some common shapes [8]

packing to quantify edge lengths. In circle packing, the shapes in question can be constructed using circles of the same dimension on each of the vertices. Circles can be used to make the sides of a polygon; spheres can be used to make the edges of a polyhedron. These shapes are mutually tangent to each other, and two radii make up one edge. Thus, the length is simply the sum of the two radii of the adjacent vertices $l_{ij} = r_i + r_j$.

1.2 Combinatorial Ricci flow

Just because we have a triangulation and that it is circle packed does not mean that it is already compact and organized. Vertices can be too large or too small, and the end shape can be somewhat repulsive. We would like

Figure 3: This is a double torus. It has χ = -2.

---

to have a way to make these radii as uniform as possible while keeping the original shape intact. We introduce the equation for combinatorial Ricci flow, which allows an individual radius to change over time. In Euclidean space, the differential equation can be written as

$$\frac{dr_i}{dt} = -K_i r_i \tag{1}$$

where K_i is a characteristic called the curvature of a vertex, and r_i is the radius or weight of a vertex i. The value of K_i changes with time. Its value is found by determining the angles of all triangles containing vertex i. Using side lengths (or radii) we can determine the angle using the law of cosines. For a triangle with lengths a, b and c, the angle opposite side c is:

$$\angle C = \arccos \frac{a^2 + b^2 - c^2}{2ab}$$

with similar formulas for the other angles. We take the sum of all angles associated with a vertex i and define the curvature K_i as:

$$K_i = 2\pi - \sum \angle i \tag{2}$$

So since the differential equation depends on a variable whose value changes over time, we cannot solve it so easily.

A potential issue we noted is that based on the equation, more often than not, the lengths of the radii would decrease. Take the example of a simple tetrahedron with all sides of equal length, we find that the curvature of

each vertex always equals π. Thus in solving the differential equation computationally we would decrease each vertex by the same amount, but the curvature of each vertex would still remain π. The radii would continue to decrease until they approach zero length. We then have to address that issue since computers don't like working with numbers near zero, as in the denominator of the arccosine function. Weird, unwanted stuff might happen. Let us introduce the ability to resize the length of each radius by a scalar, α. We denote each scaled length by $\tilde{r_i}$ and equate as

$$\tilde{r_i} = \alpha r_i \tag{3}$$

Thus in plugging in to the differential equation we get

$$\begin{aligned} \frac{d\tilde{r_i}}{dt} &= \frac{d(\alpha r_i)}{dt} = \alpha \frac{dr_i}{dt} + r_i \frac{d\alpha}{dt} \\ &= -\alpha K_i r_i + \frac{\tilde{r_i}}{\alpha} \frac{d\alpha}{dt} \\ &= -\tilde{K_i} \tilde{r_i} + \frac{\tilde{r_i}}{\alpha} \frac{d\alpha}{dt} \end{aligned} \tag{4}$$

Since we are scaling all sides by the same factor, this does not effect the curvature of the surface, so $\tilde{K_i} = K_i$. It also turns out that

$$\frac{1}{\alpha} \frac{d\alpha}{dt} = \frac{d(\log \alpha)}{dt}$$

In order to find an appropriate value for α we decided to use the following criteria:

$$\prod \tilde{r_i}(t) = \prod \alpha r_i(t) = C, \text{ a constant} \tag{5}$$

This can be used to prevent radii from decreasing to zero. It turns out if you take the derivative of (5) with respect to time you find that

$$\frac{d(\log \alpha)}{dt} = \frac{\text{sum of all curvatures}}{\text{number of vertices}} = \overline{K}, \text{average curvature} \tag{6}$$

It turns out that $\overline{K}$ is constant for any iteration of our triangulation and is equal to $\frac{2\pi\chi}{|V|}$, where $|V|$ is the number of vertices and χ is the Euler

characteristic. This is noted in [2]. Plugging this back into (4) we determine that

$$\frac{d\tilde{r_i}}{dt} = -\tilde{K_i}\tilde{r_i} + \overline{K}\tilde{r_i} = (\overline{K} - K_i)\tilde{r_i} \tag{7}$$

However, since everything is now a function of $\tilde{r_i}$ and not α, we can treat α as a constant and thus $\frac{d\alpha}{dt} = 0$. Therefore, we can easily plug $\tilde{r_i} = \alpha r_i$ back in to the differential equation, so we end up with:

$$\begin{aligned} \frac{d\tilde{r_i}}{dt} &= (\overline{K} - K_i)\tilde{r_i} \\ \alpha\frac{dr_i}{dt} &= (\overline{K} - K_i)\alpha r_i \\ \frac{dr_i}{dt} &= (\overline{K} - K_i)r_i \end{aligned} \tag{8}$$

This is known as normalized Ricci flow, as discussed in [2]. In the case of our basic tetrahedron from earlier, the radii would not change after each iteration as $\overline{K} = K_i = \pi$ for each vertex and thus $\frac{dr_i}{dt} = 0$. With its roots in differential geometry, Ricci flow has become a popular topic in mathematics, finding uses and applications in many different fields. It even helped solve one of the Millenium Puzzles, valued at $1,000,000, in 2003. [9]

2 Analysis/Results

Here we will discuss some of the things that we have been working on, as well as document examples to illustrate the benefits and correctness of our procedures.

2.1 Geometric layout

When beginning the creation of the program, the structure design was critical. Not only would it help dictate the direction of the project over the course of its lifespan, but the design decisions would affect the speed and efficiency of all added functionality. It was agreed that system would have to hold the different simplices and that they would be referencing each other. This part of the program would need to be structured in a way that was quick and easy to move from one simplex to another. As seen in figu.4, all simplices are assumed to have lists of references to other simplices, what we call local simplices, broken down by dimension. So for the two-dimensional case, each simplex has lists of local vertices, local edges, and local faces, as shown in Table 2.

The lists are vectors of integers. The vector, provided in the C++ library, was chosen so that the list can dynamically change in size. The integers are a decision based on both speed and size. Instead of, for example, a vertex having a list of actual edges ($\overline{AB}, \overline{CF}$, etc.) or pointers to edges, the vertex has a list of integers representing the edges. The actual edges are then obtained through the *Triangulation* class, which holds maps from integers to simplices. The *Triangulation* class is made up of static methods and maps and is designed so only one triangulation exists at any time. Because the maps are static, they can be accessed at any time from anywhere in the code without the need to pass pointers through function calls. Vertices also hold

Vertices	Edges	Faces
adjacent vertices	component vertices	component vertices
constructed edges	adjacent edges	component edges
constructed faces	construced faces	adjacent faces

Table 2: A layout of how data is organized

a weight, representing the radii from a circle packing on the triangulation. Edges then have a length based on the weights of its two vertices. When a vertex's weight is changed, the local edges to that vertex act as an observer and will update their lengths automatically.

The function *calcFlow* runs a Ricci flow over the triangulation and records the data in a file. The algorithm for the ODE, provided by J-P Moreau, employs a Runge-Kutta method [7]. First, the file name for the data to be written in is provided. Then, a dt is given by the user that represents the time step for the system. The next parameter is a pointer to an array of initial weights to use. This is followed by the number of steps to calculated and record. Lastly, a boolean is provided, where *true* indicates that the adjusted differential equations that are scaled to a constant, equation (8) should be used. Otherwise, the standard equation (1) is employed. Printed to the file is each step with every vertex and its weight and curvature at that moment. At the end of each step, the net curvature is printed, as shown below.

```
Step 1    Weight   Curv          Step 50   Weight   Curv
Vertex 1: 6.000 / 0.7442        Vertex 1: 4.557 / 0.008509
Vertex 2: 3.000 / -1.122        Vertex 2: 4.530 / -0.01185
Vertex 3: 3.000 / -1.373        Vertex 3: 4.534 / -0.009091
Vertex 4: 8.000 / 1.813         Vertex 4: 4.563 / 0.01268
Vertex 5: 6.000 / 1.227         Vertex 5: 4.550 / 0.002772
Vertex 6: 2.000 / -3.046        Vertex 6: 4.527 / -0.01455
Vertex 7: 4.000 / -0.3045       Vertex 7: 4.541 / -0.003563
Vertex 8: 8.000 / 1.989         Vertex 8: 4.559 / 0.01018
Vertex 9: 5.000 / 0.07239       Vertex 9: 4.553 / 0.004906
Net Curv: 0.0000                Net Curv: 0.0000
```

After the initial design of *calcFlow*, tests were run to determine its speed. The time it took to run was directly proportional to the number of steps in the flow. However, it was also proportional to more than the square of the number of vertices of the triangulation. As a result, while a four-vertex triangulation can run a 1000 step flow in three seconds, it would take a twelve-vertex triangulation 43 seconds to run the same flow. After inspecting the speed of the non-adjusted flow in comparison, it became clear that the calculation of net curvature, which remains constant in two-dimensional manifold cases, was being calculated far too often. After being adjusted so

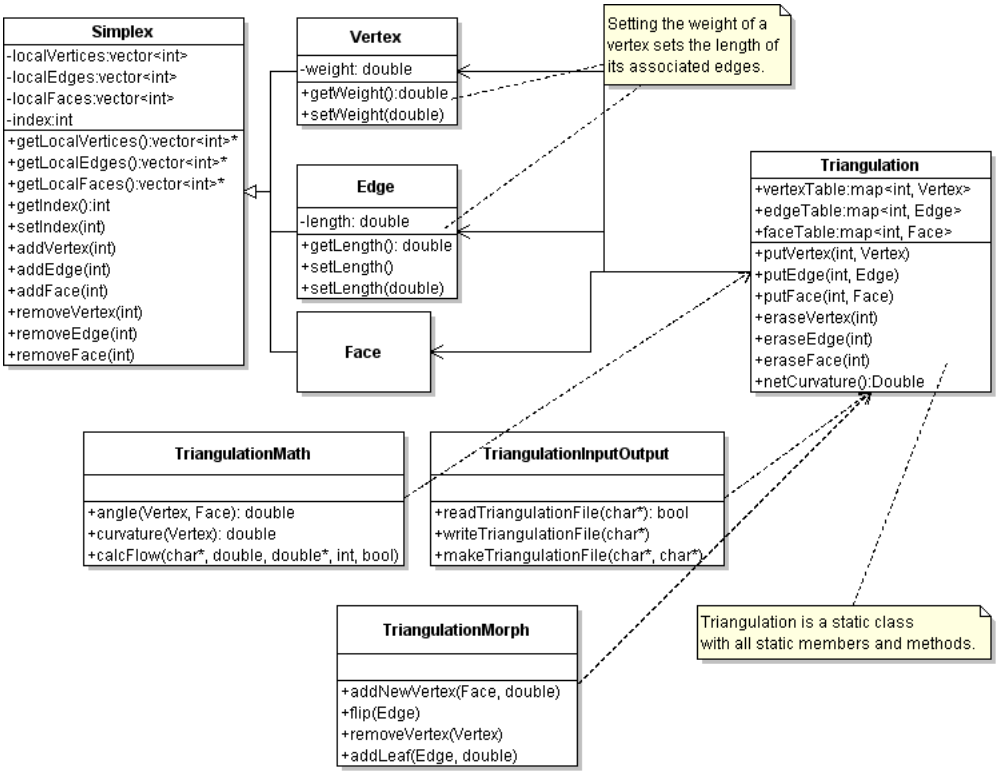

Figure 4: A layout of commands and how they are connected.

that it is calculated just once per step, the speed of the flow is much faster so that a four-vertex system with 1000 steps takes just one second and twelve vertices is much improved with a time of only four seconds. As expected, the net curvatures of our trials remained constant, ensuring that our program was running as we expected it to.

2.2 Converting known triangulations

While we are able to manually construct a few basic triangulations by hand, as we add more vertices that will become ever harder. [5] has millions of known triangulations of varying size. However, the format is different than our setup, so we developed an algorithm to take a given triangulation, saved on its own as a text file, and be able to convert it into the form that we use. We were able to transform this

```
{manifold_lex_d2_n5_o1_g0_#1=[[1,2,3],[1,2,4],[1,3,4],
[2,3,5],[2,4,5],[3,4,5]]}
```

which solely documents the faces, into this:

```
Vertex: 1       Edge: 1         Edge: 6         Face: 2
2 3 4           1 2             3 4             1 2 4
1 2 4           2 3 4 5 7       2 3 4 5 8 9     1 4 5
1 2 3           1 2             3 6             1 3 5
Vertex: 2       Edge: 2         Edge: 7         Face: 3
1 3 4 5         1 3             2 5             1 3 4
1 3 5 7         1 3 4 6 8       1 3 5 8 9       2 4 6
1 2 4 5         1 3             4 5             1 2 6
Vertex: 3       Edge: 3         Edge: 8         Face: 4
1 2 4 5         2 3             3 5             2 3 5
2 3 6 8         1 2 5 6 7 8     2 3 6 7 9       3 7 8
1 3 4 6         1 4             4 6             1 5 6
Vertex: 4       Edge: 4         Edge: 9         Face: 5
1 2 3 5         1 4             4 5             2 4 5
4 5 6 9         1 2 5 6 9       4 5 6 7 8       5 7 9
2 3 5 6         2 3             5 6             2 4 6
Vertex: 5       Edge: 5         Face: 1         Face: 6
2 3 4           2 4             1 2 3           3 4 5
7 8 9           1 3 4 6 7 9     1 2 3           6 8 9
4 5 6           2 5             2 3 4           3 4 5
```

which allows us to keep track of every vertex, edge, and face.

2.3 Flips, and the addition and deletion of vertices

There may be times where we want to change a given triangulation by a small amount. Rather than start from scratch, we can perform flips. *Flips* are a technique which change the topology of a surface by a small degree. There are three basic types:

- 1-3 flip Here we take a triangle and turn it into three triangles. In other words, we add a vertex to a triangulation and connect it to the three vertices of the triangle it is inside.

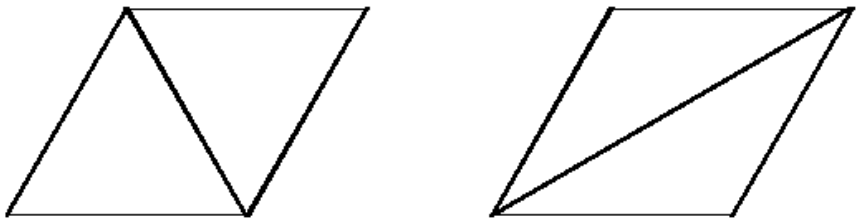

Figure 5: An example of a flip.

- 2-2 flip In this flip we take an edge that is part of two faces, and switch the vertices. See figure 5.
- 3-1 flip Here we do the opposiste of a 1-3 flip. We take a vertex that is connected to three other vertices and remove it entirely. This move is more restrictive than the others since not every vertex is connected to exactly three vertices.

Example: Adding a vertex to a one-holed torus. See figure 6.

By inserting a vertex within a flat triangulation for a torus, we essentially place a sphere on top of three ofther spheres, and then observe what happens as we run it trough out *calcFlow* program. We discover that the new vertex gets squeezed in between the other vertices and decreases in size, as if it were never there at all. To counter this, the other vertices grow slightly to maintain Eq. (5). The new vertex lies completely within the void not occupied by the other three triangles. We also ran this test with a 7-vertex torus and obtained similar results. The additional vertex tried to disappear altogether, and even with normalized Ricci flow its weight dropped down to zero, as if it weren't there at all. While this does break our code, we find it very interesting. I wonder if it matters which face you add the vertex to.

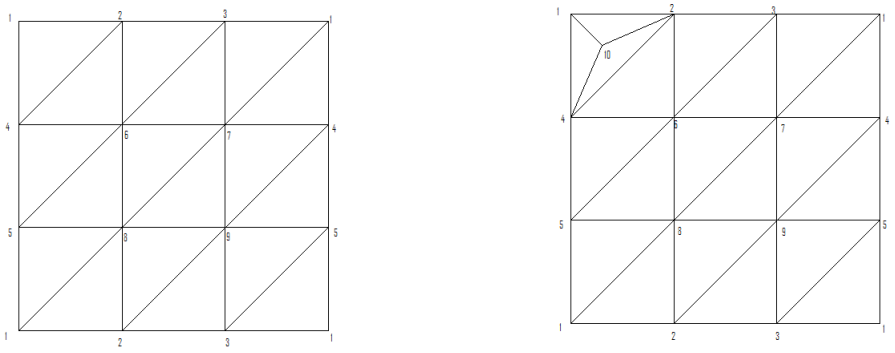

Figure 6: A triangulation of the torus, and the addition of a new vertex.

2.4 Duals

Every triangle that is capable of being circle-packed has a circle that is internally tangent to all three edges. It is known that this *incircle* is perpendicular to each edge and has a radius $r = \sqrt{\frac{(s-a)(s-b)(s-c)}{s}}$ where s is the semi-perimenter of the triangle, and a, b,and c are the side lengths. Let us rewrite this in terms of the vertex weights, since we are using circle packing. Thus, a side length is simply the sum of two vertex radii. We can then simplify this equation to $r = \sqrt{\frac{r_i r_j r_k}{r_i + r_j + r_k}}$, with $i, j$and k being the vertices of a triangle. This can be repated for multiple triangles. As a plus, all these incircles are mutually tangent. We can define a dual edge $\star e$ as the sum of the radii of the incircles with that side in question. $\star e = r_{in1} + r_{in2}$. See Figure 7 for an illustration of a dual edge.

With enough dual edges, we can obtain a polygon surrounding a vertex. The area of the region formed by these duals is called the dual area, evaluated by

$$\star A_i = r_i \sum \frac{\star e_i}{2} = r_i \sum r_{\text{inscribed}}$$

The dual length and dual area have some interesting properties that we hope to investigate. David Glickenstein examines these and more in [4].

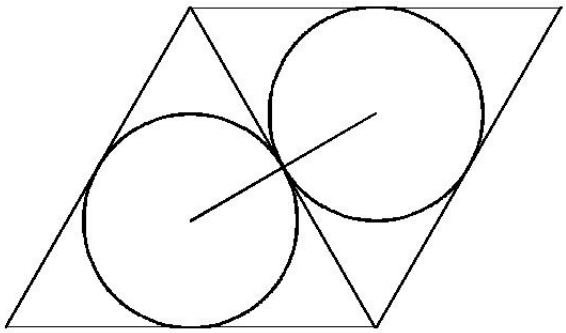

Figure 7: A dual to an edge

2.5 Adding toruses

We can add a torus handle to a triangulation. This instantly reduces the χ value of the surface by 2.

2.6 Special cases

Double Triangles. Doesn't work with our *makeTriangulationFile* format, but has some interesting properties. Associated with flips.

We have also added the ability to add a double triangle to a given edge.
Connecting polyhedra by a vertex. Since the Euler characteristic of a sphere is 2, does that value change when we add a disjoint sphere? In fact, their $\chi's$ are additive, and the new Euler characteristic of the surface(s) is 4. [8]However, if we connect these two spheres in any fashion, how does that affect their χ value? How will it react to Ricci flow?

Example: two tetrahedra connected at a vertex. It turns out χ =7 Vertices - 12 Edges + 8 Faces = 3, which is not a case we had seen before. In previous cases χ was an even integer. Starting each vertex with equal weight, we obtained an unexpected result. The center weight becomes very large, and

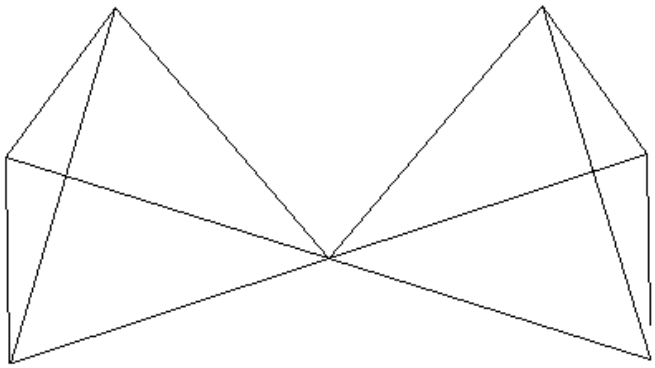

Figure 8: Two tetrahedra conjoined at a vertex.

the others become smaller in comparison. We concluded that since the central vertex now had a much lower curvature than the other vertices, it reacted differently to *calcFlow*. One good thing we noted was that the net curvature $= 6\pi = 2\pi\chi$, which was a relief.

3 Future Work

Things we could do:

- Circle packing is a very special way to characterize our side lengths. If we relax this criterium and let the circles overlap, we can introduce a second weight Φ that . We can then evaluate our side lengths as

$$l_{ij} = \sqrt{r_i^2 + r_j^2 + 2r_i r_j \cos(\Phi(e_{ij}))}$$

  With this more general interpretation, we can examine questions asked by Chow and Luo in [2].

- We would also like to start investigating 3-dimensional constructs. We can adjust our current code as much as we need to, and ultimately be able to evaluate Yamabe flow, as discussed by [3]. We can also try and investigate how our code applies (if it does) to hyperbolic and spherical coordinates.

4 Conclusions

We learned a lot of things along the way. At the beginning, we all had varying familiarities with college geometry; some of us hadn't touched the stuff since high school. But we are excited to see where Dr. Glickenstein's project is headed and we hope to remain a part of it in the months and years to come.

We would like to thank Dr. David Glickenstein for having us on this project, Dr. Robert Indik and the University of Arizona Math Department for their help and support, and the VIGRE foundation for making this possible.

References

[1] M. Brown. *Ordinary Differential Equations and Their Applications.* Springer-Verlag, New York, NY, 1983.

[2] B. Chow and F. Luo. *Combinatorial Ricci Flows on Surfaces.* Journal of Differential Geometry 63, Volume , 97-129, 2003.

[3] D. Glickenstein. *A combnatorial Yamabe flow in three dimensions.* Topology 44, 791-808, 2005.

[4] D. Glickenstein. Geometric triangulations and discrete laplacians on manifolds. Material given to us by David, 2008.

[5] F. H. Lutz. The manifold page. http://www.math.tu-berlin.de/diskregeom/stellar/.

[6] Wolfram Mathworld. http://mathworld.wolfram.com/.

[7] J-P Moreau. Differential equations in c++. http://pagesperso-orange.fr/jean-pierre.moreau/c_eqdiff.html.

[8] Wikipedia. Euler characteristic. http://en.wikipedia.org/wiki/Euler_characteristic.

[9] Wikipedia. Poincarè conjecture. http://en.wikipedia.org/wiki/Poincar%C3%A9_conjecture.

Appendix

Proof of Eq. (6)

Remarks on Runge-Kutta method for solving (8)

The method used by Moreau in [7] to solve a diffeential equation involves using a Runge-Kutta method. After investigating various methods to run our differential equation, we agreed that a Runge-Kutta format would be most beneficial for this type of problem. Even though it is more computationally complex than the simpler Euler's method, it makes up in its ability to converge and in its accuracy. According to [1] the error associated with using Runge-Kutta is on the order of h^4, whereas with a standard Euler approximation the error is simply of order h, with $h = dt$ being our step incremental.

About the Authors

Alex Henniges is a junior double majoring in Math and Computer Science. Plus he's cool.

Thomas Williams is a senior in Comprehensive Mathematics with a minor in Computer Science and a strong background in Math Education. He's cool, too.

Mitch Wilson is a senior majoring in Applied Math and Mechanical Engineering. He's ok.