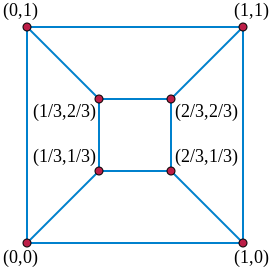
# Embedding Triangulated Terrain Models

## Introduction

A Tutte or barycentric embedding of a simple, connected, planar graph is a crossing-free straight-line embedding with a convex outer face each interior vertex being at the average - or barycenter- of its neighbors' positions. If the outer polygon is fixed, this condition on the interior vertices determines their position uniquely as the solution to a system of linear equations for which, when solved, yields the planar embedding. Tutte's spring theorem, proven by W. T. Tutte, states that this unique solution is always crossing-free, and more strongly that every face of the resulting planar embedding is convex [1]. It is called the spring theorem because the embedding can be found as the equilibrium position for a system of springs representing the edges of the graph.

The graph of a cube is shown in Figure 1 for which one of its faces is arbitrarily chosen as the outer face. The four vertices of the outer face are fixed at the four corners of a unit square. The remaining four vertices are placed at the four points whose x and y coordinates are combinations of 1/3 and 2/3, as in the figure. The result is a Tutte embedding because each interior vertex has average coordinates of its neighbours.



**Figure 1.** Tutte embedding of a cube

The condition that a vertex be at the average of its neighbors' positions may be expressed as two linear equations, one for the x coordinate of the vertex and another for the y coordinate of the vertex. For a graph with n vertices, h of which are fixed in position on the outer face, there are two equations for each interior vertex and also two unknowns - the coordinates - for each interior vertex. Therefore, this gives a system of linear equations with 2(n − h) equations and 2(n − h) unknowns. As previously stated, the solution of this linear system is the Tutte embedding. Tutte showed that for 3-vertex-connected planar graphs, this system is non-degenerate. Therefore, it has a unique solution, and the graph has a unique Tutte embedding that can be found in polynomial time.

My idea was that computing an embedding similar to Tutte’s would allows a 2.5D terrain model to be flattened and visualized in 2D with appropriate distortion. This, of course, requires an appropriate weighting scheme that deviates from the average position scheme in Tutte’s embedding. For example, viewing the map of Earth on a globe yields very different visual results than viewing Earth mapped to a rectangle as we often do as countries near the North and South poles become awkwardly distorted. Furthermore, an embedding of a 2.5D model could lead to reduced storage space since the z coordinates would not need to be stored and, thus, the information would be accurately compressed. My implementation does not provide the steps to reverse the embedding but the functionality would prove useful as future work.

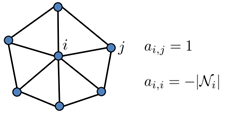
My application takes as input a triangulated mesh having a convex boundary and a weighting scheme. I have implemented three weighting schemes in order to compare their distortion. They are Uniform Laplacian, Laplace-Beltrami, and Mean value weights. The distortion of the parameterizations is visualized by saving the 2D parametrized positions as texture coordinates of the original mesh. Then, a texture is assigned to the mesh, and the mesh can be displayed using Meshlab [6].

For my implementation I have used the C++ programming language with the OpenMesh library [2] for storing and traversing a mesh as well as outputting the embedding as a .obj file. I also used the Eigen library [3] for its matrix data structure and for solving the systems of linear equations.

## Weighting Schemes

Different weighting schemes were used to test different types of distortion when embedding a mesh. Each embedding provides a rule for assigning a value to entry i,j of a matrix, a. For all schemes, boundary vertices, ai,i, are assigned 0. Furthermore, if vertices i and j are not connected by an edge in the mesh, ai,j is assigned 0.

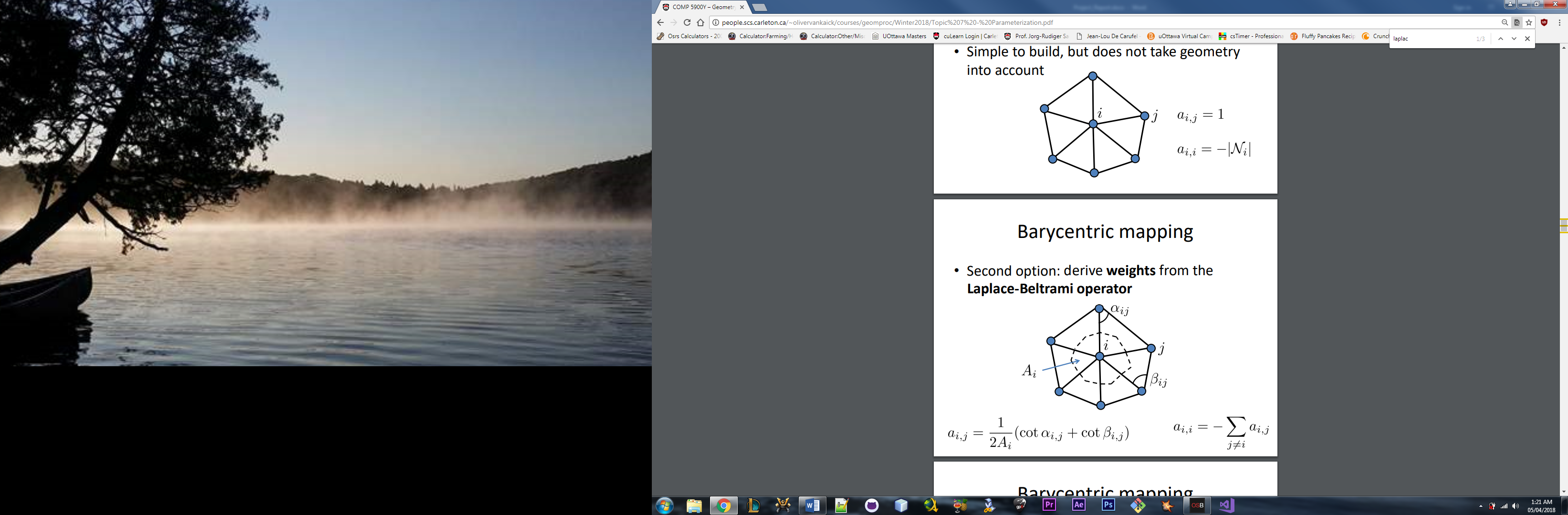
The Uniform Laplacian weighting scheme [5] applies uniform weights to neighbouring vertices. It is simple to build but does not take geometry into account. Figure 2 shows an example of applying the scheme to vertices i and j where ai,j is the i,jth position of the matrix a. ai,j is assigned 1 if there is an edge from i to j, otherwise 0. And ai,i is assigned -|Ni| where |Ni| is the valence of vertex i.



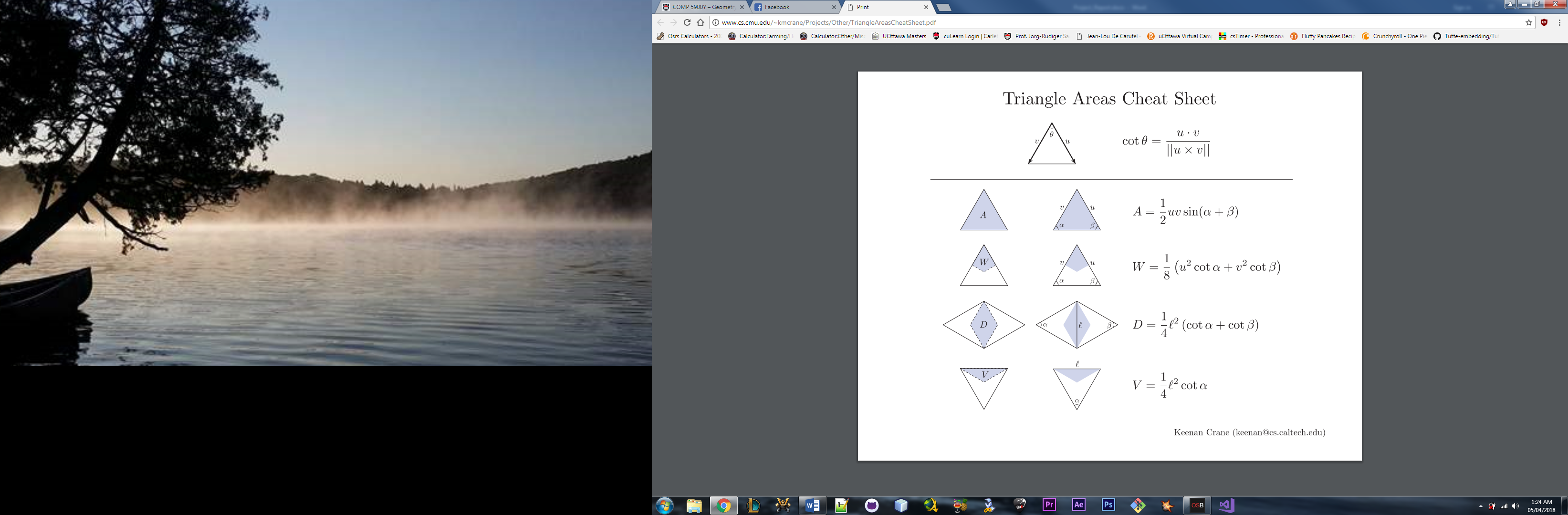
**Figure 2.** Uniform Laplacian Weighting Scheme [5]

The Laplace-Beltrami weighting scheme [5] derives weights from the Laplace-Beltrami operator which a linear operator defined as the divergence of the gradient of a function. In Euclidean space, the gradient of a function is defined as the unique vector field whose dot product with any vector v at each point x is the directional derivative of f along v. Basically, the gradient as the amount of slope at a spot on a surface which describes all the directions at once. The divergence is a vector operator measures the magnitude of a vector field's source or sink at a given point, in terms of a signed scalar [7].

Figure 3 shows applying the Laplace-Beltrami weighting scheme to vertices i and j. The equation for ai,j is defined such that Ai is the sum of Voronoi areas of all triangles adjacent to vertex i. Figure 4 shows the formula for calculating the Voronoi area of a single triangle. The diagonal of the matrix is, again, defined to be -1 multiplied by the valence of vertex i. This scheme takes the geometry of the mesh into account. That is, it captures triangle angles and edge lengths, and thus will adapt the parameterization according to the geometry of the shape. There is, however, a problem with triangles having obtuse angles due to coefficients becoming negative and violating the requirements of Tutte’s theorem.

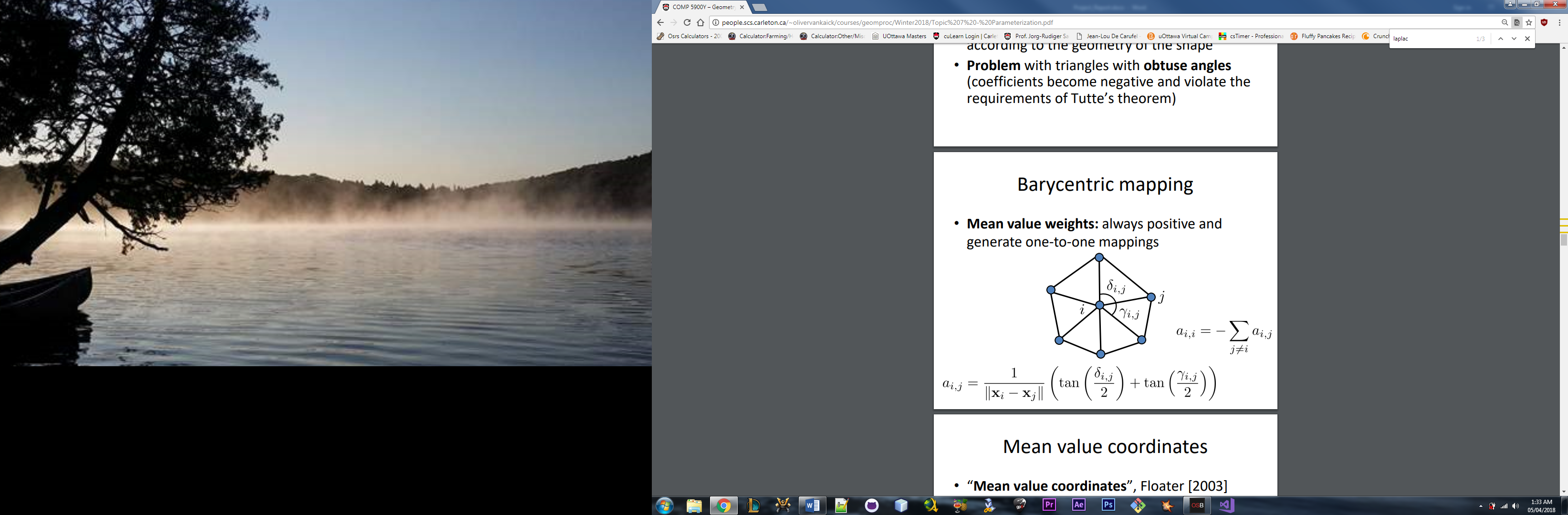


**Figure 3.** Laplace-Beltrami Weighting Scheme [5]



**Figure 4.** Formula for Voronoi Area of a Triangle [4]

The Mean Value weighting scheme [5] is based on the Mean Value Theorem and is a generalization of barycentric weights to arbitrary polygons. The scheme always produces positive values and always generates a one-to-one mapping. Figure 5 shows the application of this weighting scheme to vertices i and j. The equation for ai,j is defined such that ||xi – xj|| is the Euclidean distance between vertices i and j. The diagonal of the matrix is, again, defined to be -1 multiplied by the valence of vertex i.



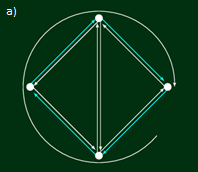
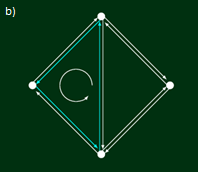
**Figure 5.** Mean Value Weighting Scheme [5]

## Navigating a Mesh using OpenMesh

OpenMesh is a C++ library providing a generic and efficient data structure for representing and manipulating polygonal meshes [2]. OpenMesh implements a halfedge data structure. A halfedge is a directed edge representing connectivity from a vertex, i, to a vertex j. Halfedges come in pairs called twins, where the twin halfedge is directed from vertex j to i.

Halfedges make it possible to circulate around a face in order to enumerate all its vertices, halgedges, or neighboring faces. When starting at a vertex' halfedge and iterating to the opposite of its previous one, one can easily circulate around this vertex and get all its one-ring neighbors, the incoming/outgoing halfedges, or the adjacent faces. Figure 6a and 6b show how a simple mesh can be traversed by using the halfedges. OpenMesh also provides linear iterators that enumerate vertices, halfedges, edges, and faces.

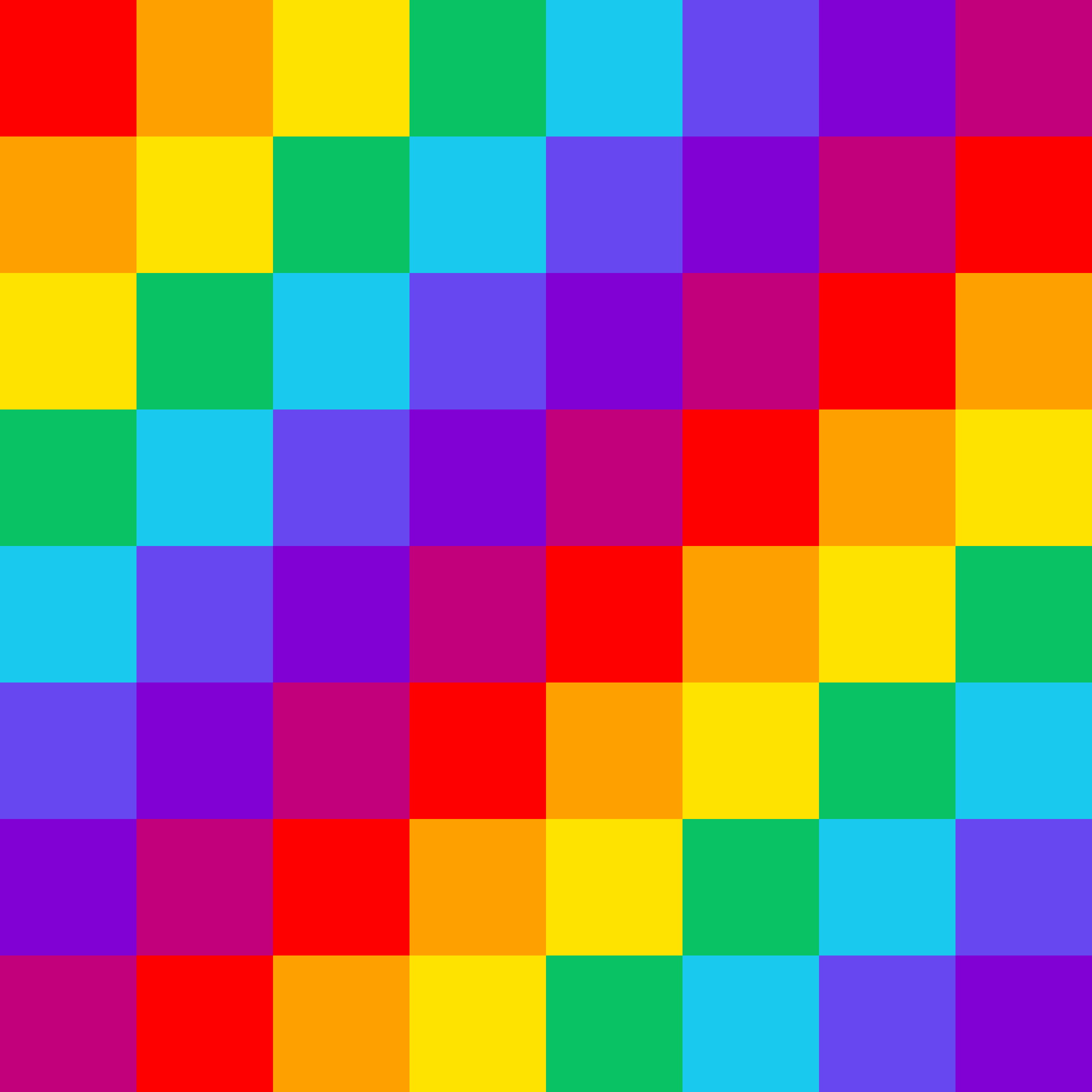
For a given vertex, OpenMesh provides the functionality to iterate through its adjacent vertices, ingoing and outgoing halfedges, and adjacent faces. Similarily, for a given face, it is possible to iterate the vertices or halfedges in the face, and the adjacent faces.

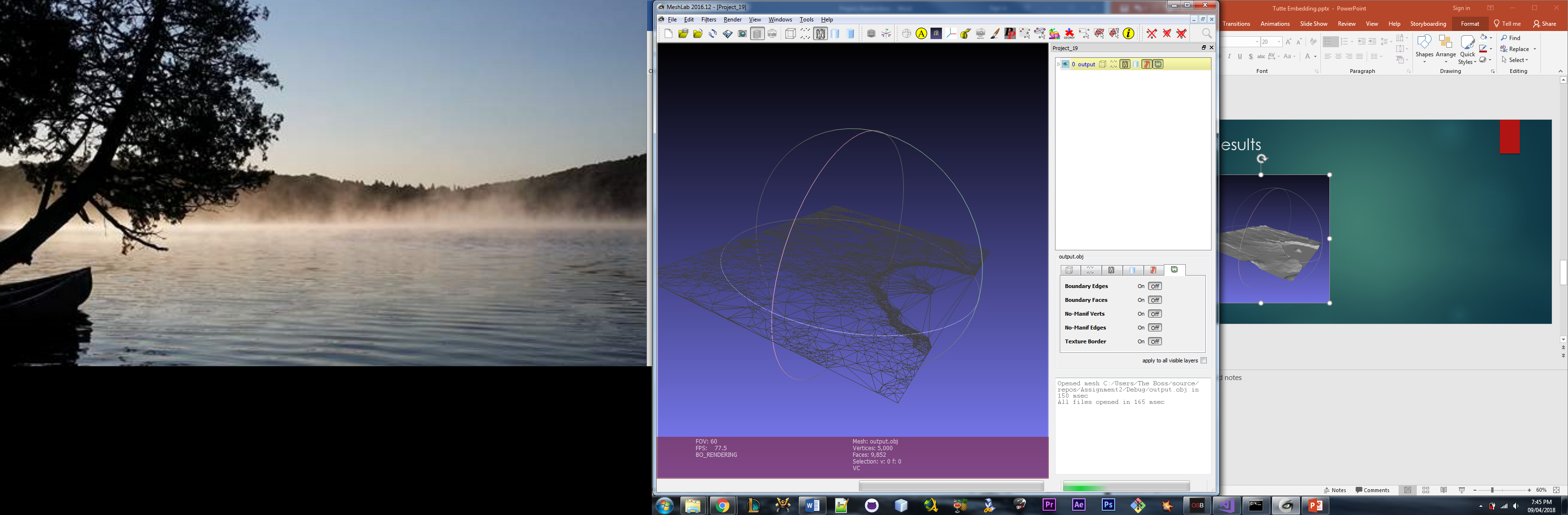
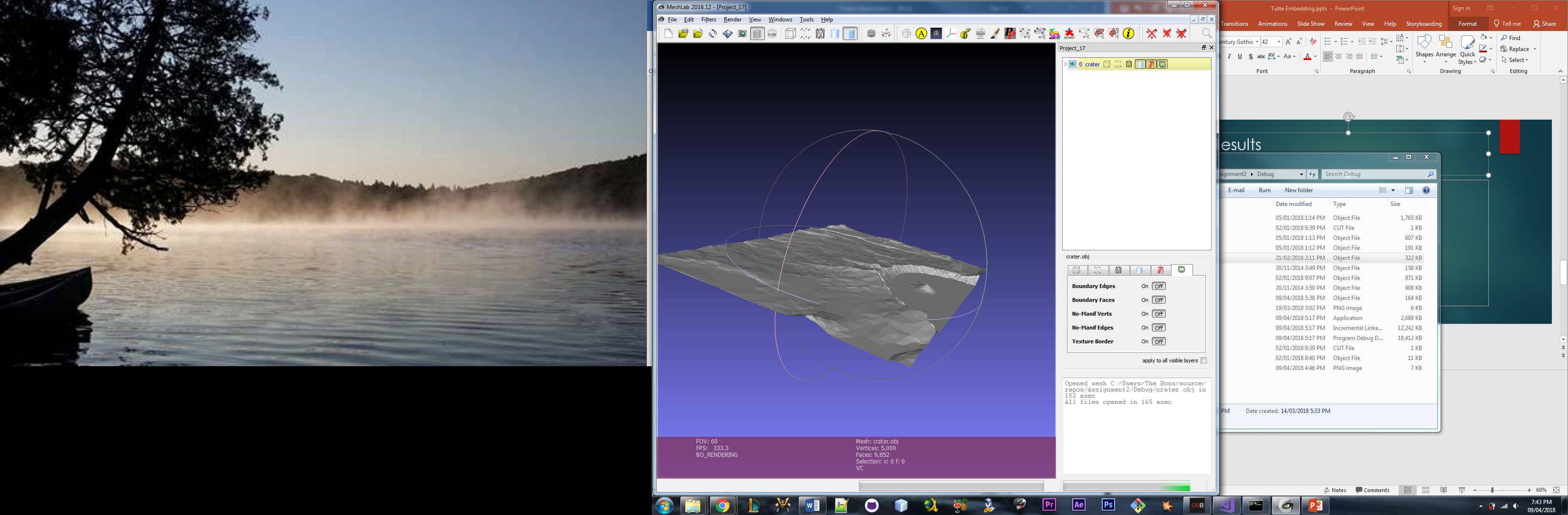
**Figure 6.** Using a Halfedge Circulator [8]

## Results

To test the results of each weighting scheme, 2D parametrized positions were saved as texture coordinates of the original mesh. Applying a texture like the one shown in Figure 7 allows the distortion to be easily compared between schemes. The mesh can then be visualized using Meshlab. Figure 8 shows a 2.5D triangulated model with a convex boundary displayed in MeshLab.



**Figure 7.** Texture Used to Test Weighting Scheme Distortion



**Figure 8.** 2.5D Terrain Model in Meshlab

Before this project I did not have a lot of experience programming in C++ and figuring out two new libraries, OpenMesh and Eigen, proved to be quite time consuming.

The OpenMesh library is very powerful but the documentation was not that helpful. Particularly, the error messages were not helpful for debugging and the documentation was not very detailed. Otherwise, the library was very nice to use. Traversing a mesh is much more complicated using code than by hand but I learned a lot about the halfedge data structure and found it very intuitive once I got used to it.

The Eigen library was used for its matrix data structure and for solving the linear systems of equations. The library is simple to use and I found the documentation quite good. Plenty of resources and sample code is available online so getting started with the library was easy. The library offers a special sparse matrix data structure which sped up the run time of the linear system solver by many orders of magnitude. Majority of my time was spent deciding which linear system solver I wanted to use. I tried many solvers but decided to use the LDL decomposition because it yielded the quickest results. Other solvers I tried and elected against were the QR decomposition, the Cholesky decomposition and the Jacobi SVD decomposition.

Unfortunately, I was unable to complete this project. I have fully implemented the three weighting schemes described in the Weighting Schemes section of this paper and have fully implemented the system of equations. Once solved, the parameterization is applied as texture coordinates to the mesh and it is written to an output .obj file. However, I have been unable to properly visualize the results of the parameterization.

First, I thought that I was unable to figure out how to apply a texture properly to my mesh in Meshlab. As an alternative, I figured it would be possible to view the distortion by updating the mesh vertices to match the calculated parameterization. That is, I assumed the mesh shown in Figure 8b would show the distortion. Unfortunately, all weighting schemes yielded the same result which was a mesh identical to the original mesh.

After extensive testing of the weighting schemes, I concluded that they have all been implemented correctly as they yield correct results. Furthermore, I have tested solving various toy examples of linear systems of equations which all yielded correct results. I believe the problem, then, is that I have set up either the matrix, a, or the vectors, u and v, incorrectly which results in an inaccurate parameterization.

I have only tested my implementation with the mesh shown in Figure 8 because I was not able to finish the project. I would have liked to do additional testing with a mesh having more extreme changes in elevation. Additionally, I would have liked to write a more detailed comparison between the weighting schemes, as this was an important part of the motivation behind this project.

## Conclusion

I have attempted to implement Tutte’s embedding for 2.5D triangulated models by setting up a linear system of equations and solving it, allowing the model to be flattened and visualized in 2D with appropriate distortion. I have implemented three different vertex weighting schemes: Uniform Laplacian, Laplace-Beltrami, and Mean value weights but was unable to compare them. I assume that the Laplace-Beltrami scheme will yield the most accurate distortion results with respect to face area representing the distance between any two points on the mesh because it is the only scheme that takes geometry of the mesh into account. I stored the parameterization as texture coordinates of the original mesh such that a texture can be applied to the mesh to visualize in Meshlab and compare the distortion using the different weighting schemes.

## References

[1] Tutte, W. T. (1963), "How to draw a graph", *Proceedings of the London Mathematical Society*, **13**: 743–767. https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/plms/s3-13.1.743

[2] OpenMesh C++ Library. https://www.openmesh.org/

[3] Eigen C++ Library. http://eigen.tuxfamily.org/index.php?title=Main\_Page

[4] Triangle Areas Cheat Sheet. http://www.cs.cmu.edu/~kmcrane/Projects/Other/TriangleAreasCheatSheet.pdf

[5] Floater, M. (March 2003). “Computer Aided Geometric Design Volume 20, Issue 1”. https://www.sciencedirect.com/science/article/pii/S0167839603000025

[6] MeshLab. http://www.meshlab.net/

[7] Belkin, M.; Niyogi, P. (1961). “Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering”. <http://papers.nips.cc/paper/1961-laplacian-eigenmaps-and-spectral-techniques-for-embedding-and-clustering.pdf>

[8] “How to navigate on a mesh”. http://www.openmesh.org/Documentation/OpenMesh-2.0-Documentation/mesh\_navigation.html