COMP0119: Acquisition and Processing of 3D Geometry

Coursework 2: Curvature Discretisation and Mesh Smoothing

This report will demonstrate the algorithm implemented for the curvature discretisation and mesh smoothing.

Section 1: Curvature Discretisation

Task 1: Uniform Mean and Gaussian Curvature

Uniform Mean Curvature

To compute the uniform mean curvature, we need to first construct the Laplacian operator in sparse matrix form using the number of neighbouring vertices:

The number of neighbouring vertices can be retrieved using igl::adjacency\_list() which returns the index of connected vertices for each vertex. For each connected vertex, we assign the to it and at last we need to make sure the diagonal direction is filled with negative 1. Next, we need to take the part into the equation by multiplication, which is just the vertex information of the input mesh.

The discrete mean curvature is given as and the second half part has already been calculated before. Therefore, we can simply compute the norm of each element in the  and multiply it by 0.5 to get the uniform mean curvature **H**.

The implementation explained above can be found in MS::LaplacianMatrix() and MS:: UniformMeanCurvature().

Uniform Mean Curvature

To compute the Gaussian curvature, we need to know the face area of the one-ring neighbours of each vertex and the angle between two continuous edges. The equation is given as:

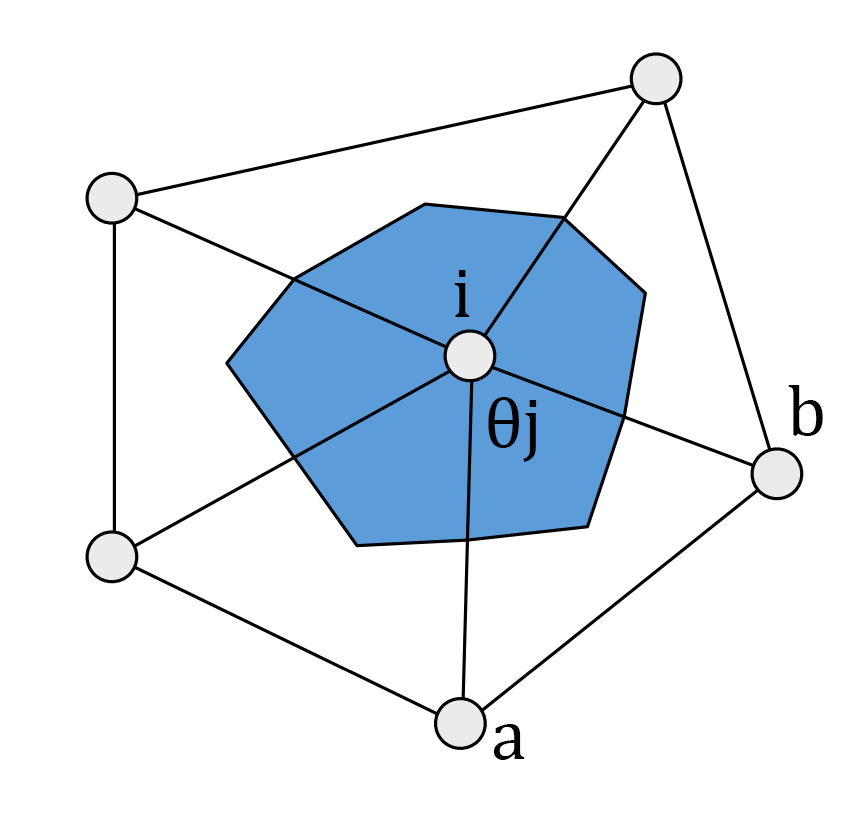
First, we need to construct a sparse matrix representing the mass. To compute the area (mass) of each face on the mesh, we can use the Heron’s formula since we have access to the face information and know each vertex coordinate within a triangle:

, where *a*, *b*, *c* is the length of each side and

We store all the value into an area list and can easily access the value using the face index as many times as we want.

The next step is to identify those faces that connect to the current vertex. We can use the igl::vertex\_triangle\_adjacency() to obtain a list of connected face for each vertex, then use the face index to lookup the value we compute in the previous step. Because we want to use the Barycentric cells, we sum up the area of connected faces for each vertex and divide it by . We apply each value to the diagonal direction, and this will give us the **M** matrix.

The next step is to calculate the angle between each two edges using the vector dot product. The angle can be computed as:



, where ,

Because we already have the connected face information of each vertex, we can immediately know the index of and vertex and thus perform the calculation above. By summing up the angle in radians and putting it back to the Gaussian curvature equation, we obtain the **K** for the input mesh.

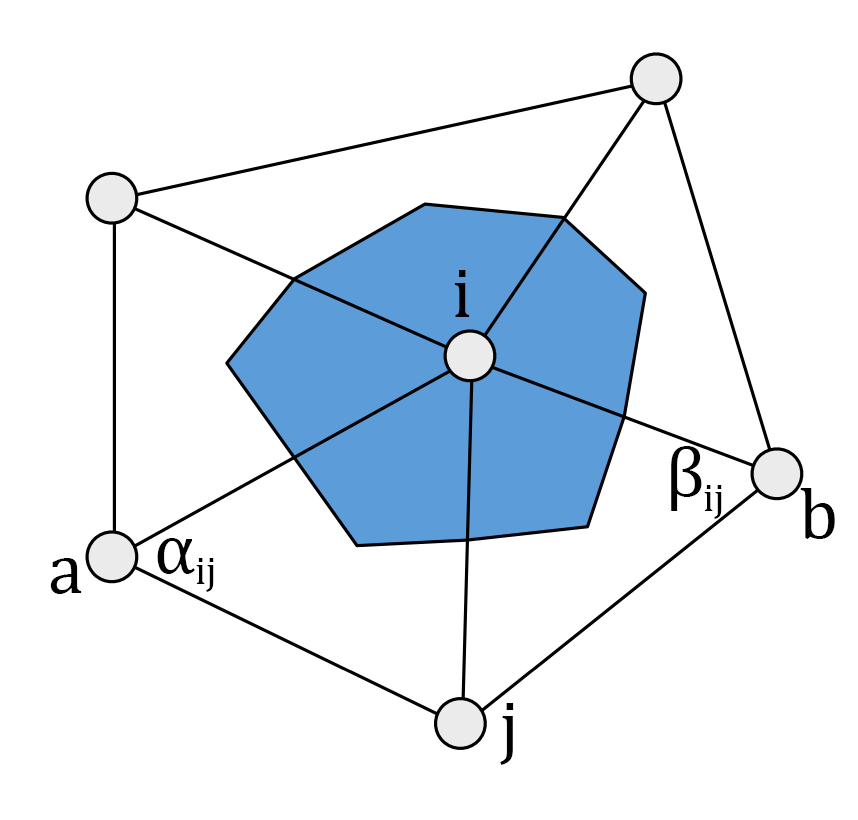
The implementation demonstrated above can be found in MS::BarycentricMassMatrix() and MS::GaussianCurvature().

Task 2: Non-Uniform Mean Curvature

To calculate the non-uniform mean curvature, we will need to apply the Laplace-Beltrami matrix to the input mesh using the following equation:

The Laplace-Beltrami operator is formed by two matrix that we need to implement here: . The **M** matrix has already been implemented and we just need to take an inverse version of it using coefficient-wise inverse function in Eigen: cwiseInverse(). The cotangent matrix **C** is given by:

The implementation of the cotangent matrix **C** is similar to the process we take to find the angle in Gaussian curvature: we have to know the connected vertices and faces in order to find the edges. They can be found using igl::adjacency\_list() and igl::vertex\_triangle\_adjacency(). The algorithm is explained below:



Since we have the connected vertices of each vertex , we can use the known vertex , or edge to find the third vertex that forms each face, and , using the adjacent face information (this method **ONLY** applies to manifold meshes and it is guaranteed to return two vertices). Then the and value can be computed using dot product from , and , respectively so that we are able to compute the cotangent value of obtained angles. Once we fill the values into the sparse matrix based on the , condition, we can get the cotangent matrix **C**. The last step is exactly same as the uniform mean curvature except that we replace the uniform Laplacian operator with Laplace-Beltrami operator:

The implementation illustrated above can be found in MS::CotangentMatrix(), MS::LaplaceBeltramiMatrix() and MS::NonUniformMeanCurvature().

Task 3: Reconstruction

Bla

Section 2: Mesh Smoothing

Task 5: Explicit Laplacian

Bla

Task 6: Implicit Laplacian

Bla

Task 7: Denoising

Bla

Reference

Desbrun, M., et al. (1999). Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow. *SIGGRAPH '99 Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, 317-324. doi: 10.1145/311535.311576

Lu, T. (2013). *Lecture 12: Discrete Laplacian*. Stanford University. Retrieved from <https://graphics.stanford.edu/courses/cs468-13-spring/assets/lecture12-lu.pdf>