Assignment 2 Discrete Curvature and Spectral and Meshes & Laplacian Mesh Smoothing

Haoyi Xu

23066240

1 Abstract

- 1 Uniform Laplace: compute mean curvature (H) and Gaussian curvature (K) at each vertex. *Complete*
- 2 First and Second Fundamental forms *Complete*
- 3 Non-uniform (Discrete Laplace-Beltrami) *Complete*
- 4 Modal analysis: Compute and show mesh reconstructions using the smallest k eigen vectors of the discrete Laplace-Beltrami operator. *Complete*
- 5 Implement explicit Laplacian mesh smoothing *Complete*
- 6 Implement implicit Laplacian mesh smoothing *Complete*
- 7 Evaluate the performance of Laplacian meshd enoising in the context of data smoothing. *Complete*

2 Uniform Laplace

2.1 Mean Curvature

For every vertex, I compute using the Laplace operator and uniform discretization the meancurvature H.

To be more specific, I use the equation below to compute the absolue value of the curvature of every vertex first:

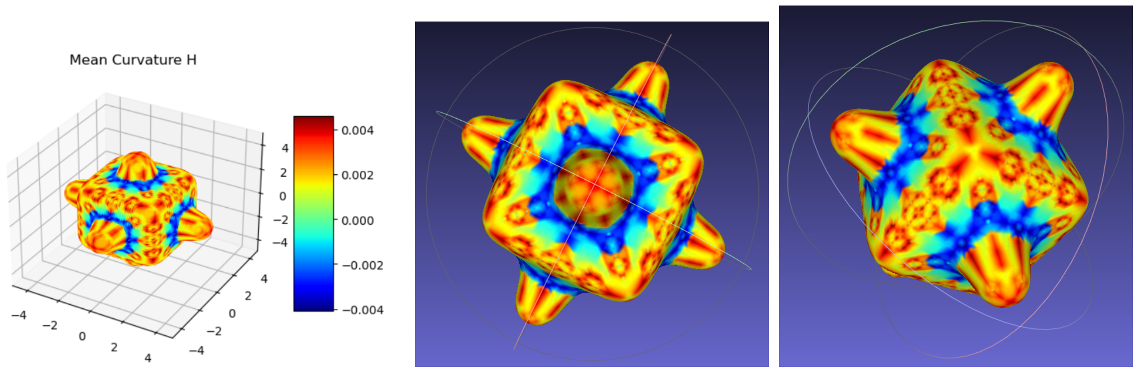
$$H_i = \frac{1}{2} \|\Delta_{\mathcal{S}} \mathbf{x_i}\| \tag{1}$$

where the divergence part is

$$\Delta_{\mathcal{S}} f\left(v_i\right) := \frac{1}{\left|\mathcal{N}_1\left(v_i\right)\right|} \sum_{v_j \in \mathcal{N}_1\left(v_i\right)} \left(\mathbf{x}_j - \mathbf{x}_i\right) \tag{2}$$

Then I use the result of the dot product between the div vector and the normal vector to determine the sign of the curvature of each vertex.

I visualized the result as shown in followings:



(a) Plot (b) Meshlab 1 (c) Meshlab 2

Figure 1: The Result of Mean Curvature of bumpy-cube.obj

2.2 Gaussian Curvature

In this section, I estimate discrete Gaussian curvature K at each vertex using the angle deficit at each vertex of a mesh and normalize Gaussian curvature K by area A_i.

To be more specific, the equation used here is

$$K = \left(2\pi - \sum_j \theta_j\right) / A \tag{3}$$

where θ_j refers to the degrees of the angles at vertex j in all triangles that have vertex j as a vertex as the following figure shows:

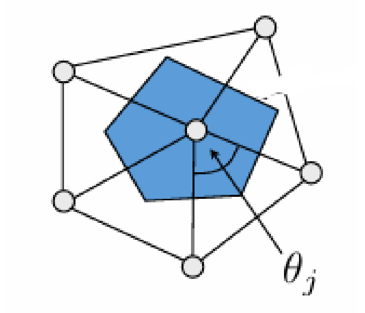
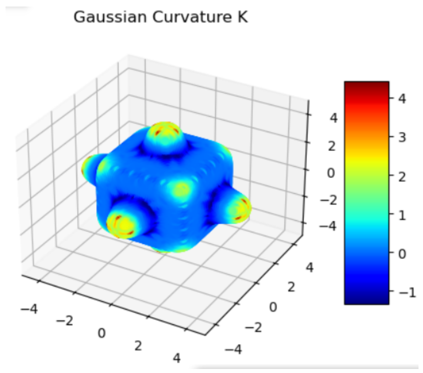


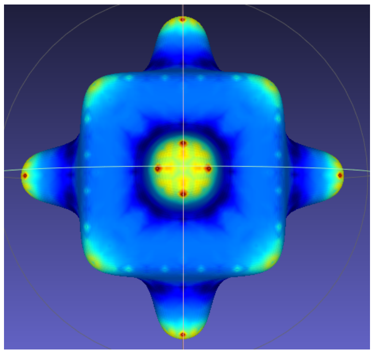
Figure 2: The Definition of θ_j

And I used the method Mixed Cells to get $A(v_i)$ which is the blue part of the figure above. The Mixed Cells Algorithm approximate the area of the polygon by dividing it into a combination of non-obtuse and obtuse triangles. To calculate the area, it connects the midpoints of edges with either the circumcenters for non-obtuse triangles or the midpoint of the opposite edge for obtuse triangles. The method provides a better approximation of the area than simpler methods but is more complex to compute.

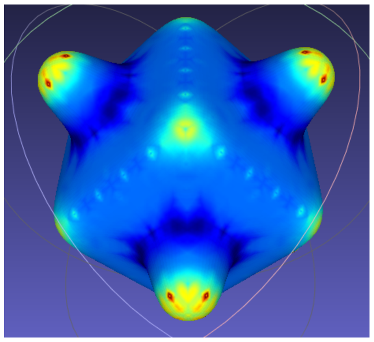
I visualized the result as shown in followings:



(a) Plot



(b) Meshlab 1



(c) Meshlab 2

Figure 3: The Result of Gaussian Curvature of bumpy-cube.obj

2.3 Compare and Discuss

By comparing the results obtained from mean curvature and Gaussian curvature, we can clearly observe that the latter's results are much better than the former's. This is because the curvature distribution of the latter is much smoother, and the noise is also significantly reduced, allowing us to clearly capture the key areas in the model. The former has many intense changes that were not expected, indicating that this curvature algorithm is not stable.

3 First and Second Fundamental Forms

3.1 1st and 2nd Fundamental Forms

In this section, I took the parameteric equations of an ellipsoid as

$$p(u,v) := \{a\cos(u)\sin(v), b\sin(u)\sin(v), c\cos(v)\} \text{ for } u \in [0,2\pi) \text{ and } v \in [0,\pi] \tag{4}$$

and analytically compute the first and second fundamental at any point $p(u,v)$ on the ellipsoid.

The First Fundamental form is;

$$\mathbf{I} = \begin{pmatrix} E & F \\ F & G \end{pmatrix} := \begin{pmatrix} \mathbf{x}_u^T \mathbf{x}_u & \mathbf{x}_u^T \mathbf{x}_v \\ \mathbf{x}_u^T \mathbf{x}_v & \mathbf{x}_v^T \mathbf{x}_v \end{pmatrix} \tag{5}$$

where

$$\begin{aligned} \mathbf{x_u} &= \begin{bmatrix} -a \cdot \sin(u) \cdot \sin(v) & b \cdot \cos(u) \cdot \sin(v) & 0 \end{bmatrix} \\ \mathbf{x_v} &= \begin{bmatrix} a \cdot \cos(u) \cdot \sin(v) & b \cdot \sin(u) \cdot \cos(v) & -c \cdot \sin(v) \end{bmatrix} \end{aligned} \tag{6}$$

The Second Fundamental form is;

$$\mathbf{II} = \begin{pmatrix} e & f \\ f & g \end{pmatrix} := \begin{pmatrix} \mathbf{x}_{uu}^T \mathbf{n} & \mathbf{x}_{uv}^T \mathbf{n} \\ \mathbf{x}_{uv}^T \mathbf{n} & \mathbf{x}_{vv}^T \mathbf{n} \end{pmatrix} \tag{7}$$

where

$$\begin{aligned} \mathbf{x_{uu}} &= \begin{bmatrix} -a \cdot \cos(u) \cdot \sin(v) & -b \cdot \sin(u) \cdot \sin(v) & 0 \end{bmatrix} \\ \mathbf{x_{uv}} &= \begin{bmatrix} -a \cdot \sin(u) \cdot \sin(v) & b \cdot \cos(u) \cdot \cos(v) & 0 \end{bmatrix} \\ \mathbf{x_{vv}} &= \begin{bmatrix} -a \cdot \cos(u) \cdot \sin(v) & -b \cdot \sin(u) \cdot \sin(v) & -c \cdot \cos(v) \end{bmatrix} \end{aligned} \tag{8}$$

3.2 Normal Curvature κ_n

I then used the result of 1st and 2nd Fundamental Forms to compute the normal curvature κ_n at the point(a,0,0) as a function of a direction vector on the local tangent plane. The equation used here is as follow

$$\kappa_n(\bar{\mathbf{t}}) = \frac{\bar{\mathbf{t}}^T \mathbf{I} \mathbf{t}}{\bar{\mathbf{t}}^T \mathbf{I} \bar{\mathbf{t}}} = \frac{ea^2 + 2fab + gb^2}{Ea^2 + 2Fab + Gb^2} \tag{9}$$

The function required in this section is basically the implementation of the above formula. Using 1, 2, 3 as the parameters of the ellipsoid and $(1, 0)$ as the direction vector, the resulting normal curvature κ_n computed by my function is 0.25.

4 Discrete Laplacian-Beltrami

4.1 Compute Method

In this section, I estimate discrete Gaussian curvature K at each vertex using the angle deficit at each vertex of a mesh and normalize Gaussian curvature K by area A_i.

To be more specific, the equation used here is

$$\Delta_S \mathbf{x} = -2H\mathbf{n} \tag{10}$$

where the divergence part is:

$$\Delta_{\mathcal{S}} f(v_i) := \frac{1}{2A(v_i)} \sum_{v_j \in \mathcal{N}_1(v_i)} (\cot \alpha_{ij} + \cot \beta_{ij}) (f(v_j) - f(v_i)) \tag{11}$$

where α_{ij} and β_{ij} represent: In the two triangles sharing a common edge ij, the 2 angles opposite to the edge ij as shown in the figure below

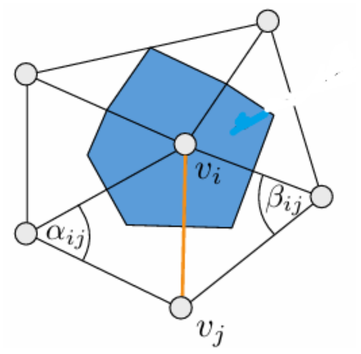


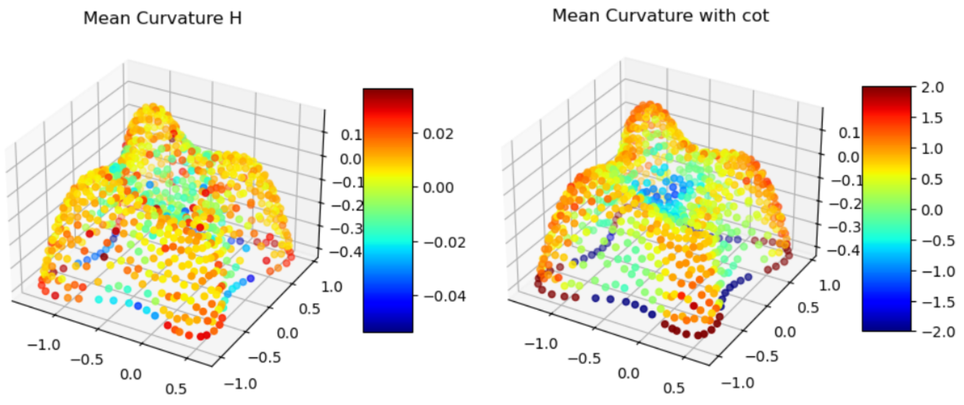
Figure 4: α_{ij} and β_{ij}

And I used the method Mixed Cells to get $A(v_i)$ as section2.

4.2 Visualize and Compare

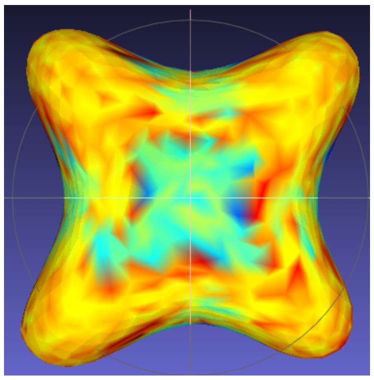
4.2.1 *lilium$_s$.obj*

Visual results of Uniform Curvature calculated and Non-uniformed(Discrete Laplace-Beltrami) calculated are shown as follows



(a) Uniform Curvature (b) Non-uniformed Curvature

Figure 5: Plot Visual results of *lilium$_s$.obj*


(a) Uniform Curvature

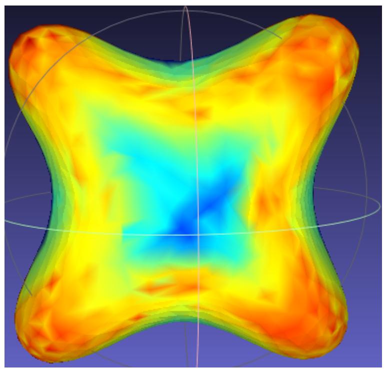
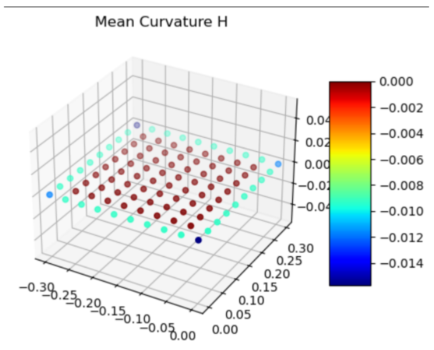

(b) Non-uniformed Curvature

Figure 6: MeshLab Visual results of $lilium_s.obj$

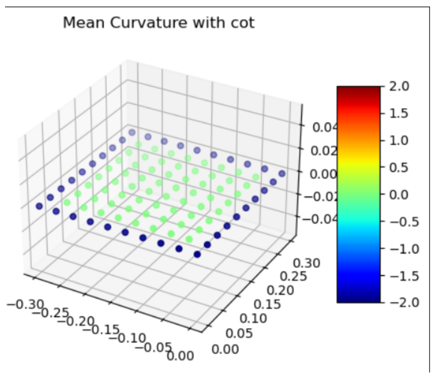
By comparing the results obtained from Uniform Curvature and Non-Uniform Curvature, we can clearly observe that the latter get much better results than the former. To be more specific, the curvature distribution of the latter is much smoother, and the noise is also significantly reduced, allowing us to clearly capture the key areas in the model. The former has many intense changes that were not expected, indicating that this curvature algorithm is not stable.

4.2.2 *plane.obj*

Visual results of Uniform Curvature calculated and Non-uniformed(Discrete Laplace-Beltrami) calculated are shown as follows(I clamped the result here, limiting the final output result to between -2 and 2 to prevent high noise. Some may argue that the comparison results may be unfair to the former algorithm, but it demonstrated that the latter's results are over ten times larger than the former's, thus making it more feasible to apply a clamping operation to achieve better outcomes.)

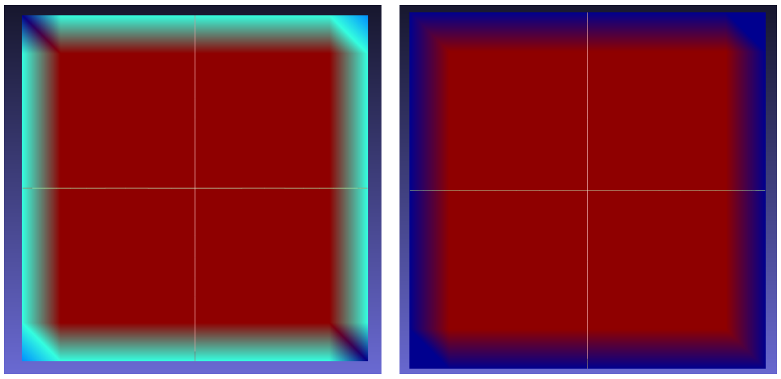


(a) Uniform Curvature



(b) Non-uniformed Curvature

Figure 7: Plot Results of $plane_s.ob$



(a) Uniform Curvature (b) Non-uniformed Curvature

Figure 8: MeshLab Visual results of $plane_s.ob$

By comparing the results obtained from Uniform Curvature and Non-Uniform Curvature, we can clearly observe that the latter get much better results than the former. Especially on the four edges of the plane, the former exhibits significant noise, corresponding to more than three different colors. In contrast, the latter is remarkably smooth, entirely devoid of noise, with the entire model comprising only two colors, corresponding to flat areas and edges respectively.

5 Model Analysis

5.1 Laplace-Beltrami Operator

The formula of Laplace-Beltrami Operator in the lecture slides is shown as follows:

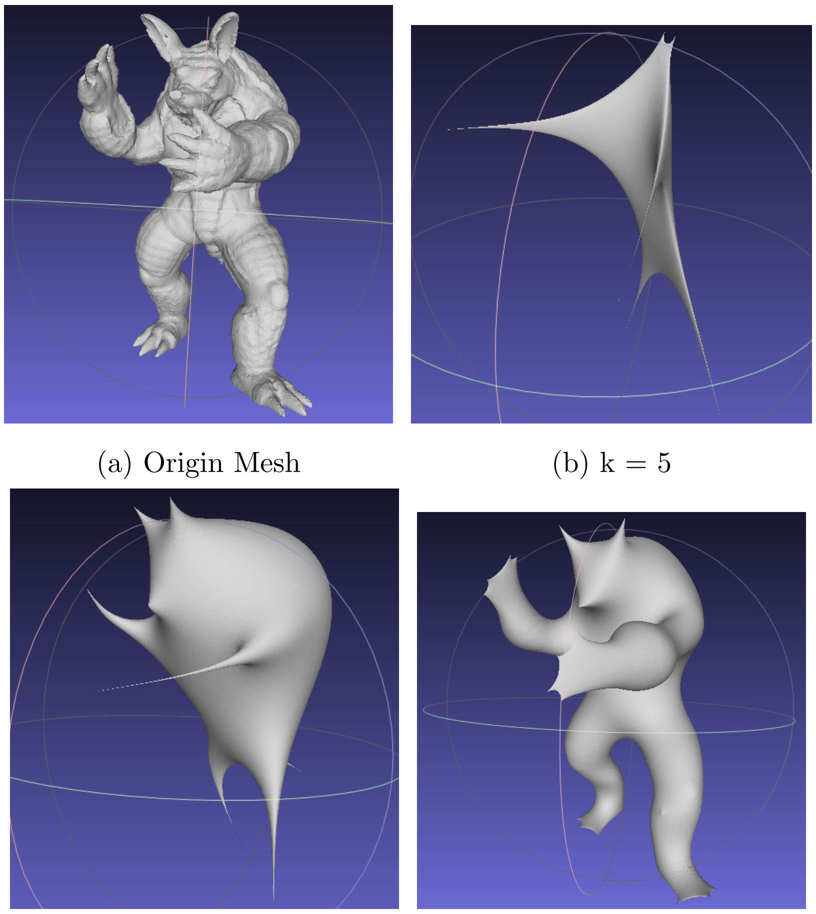
$$\mathbf{L} = \mathbf{M}^{-1}\mathbf{C} \in R^{n\times n} \tag{12}$$

where

$$\mathbf{C}_{ij} = \begin{cases} \cot\alpha_{ij} + \cot\beta_{ij}, & i \neq j, j \in \mathcal{N}_1(v_i) \\ -\sum_{v_j\in\mathcal{N}_1(v_i)}(\cot\alpha_{ij} + \cot\beta_{ij}) & i = j \\ 0 & \text{otherwise} \end{cases}$$
$$\mathbf{M}^{-1} = \operatorname{diag}\left(\ldots, \frac{1}{2A_i}, \ldots\right) \tag{13}$$

5.2 Reconstruction

I then performs eigenvalue decomposition on the Laplace-Beltrami Operator obtained above, selects the first k smallest eigenvectors which represent the fundamental shapes and structures within the mesh, allowing for effective reconstruction and analysis while reducing noise and less relevant high-frequency details. And then I uses these for vertex reconstruction. The reconsturction result are shown as follows:



(a) Origin Mesh

(b) k = 5

(c) k = 15

(d) k = 50

Figure 9: Reconstruction Result of *armadillo.obj*

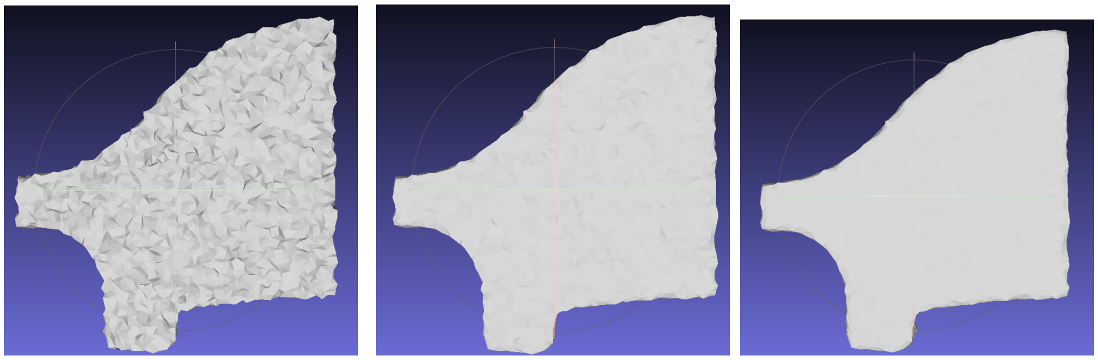
From the results of the reconstruction above, we can see that as k increases, the results of the reconstruction become increasingly closer to the original mesh. By the time k reaches 50, the original mesh's general shape has already been largely restored.

6 Explicit Laplacian Mesh Smoothing

First, I Implement explicit Laplacian mesh smoothing which corresponds to the following equation:

$$\mathbf{P}^{(t+1)} = (\mathbf{I} + \lambda \mathbf{L})\mathbf{P}^{(t)} \tag{14}$$

where L is the Laplacian Operator we compute above and P^t represents the current vertices and $p^t + 1$ represents the vertices after smoothing:



(a) The Origin Mesh (b) $lamda = 0.0005$ (c) $lamda = 0.001$

Figure 10: Explicit Smoothing Results of $fandisk_ns.obj$(iteration = 10)

If my step size is set too large the mesh will begin to deform.

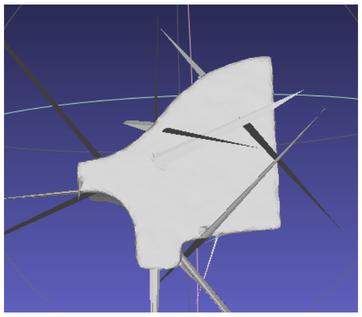


Figure 11: step size is too set too large

7 Implicit Laplacian Mesh Smoothing

7.1 Solution

For this section, I Implement implicit Laplacian mesh smoothing which corresponds to the following equation:

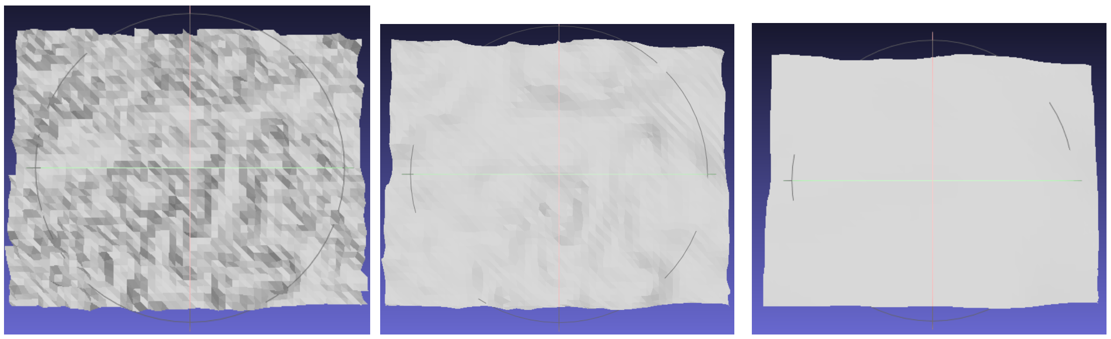
$$(\mathbf{I} - \lambda\mathbf{L})\mathbf{P}^{(t+1)} = \mathbf{P}^{(t)} \tag{15}$$

7.2 Visualization and Comparison

Visual results of comparison between Explicit Laplacian Mesh Smoothing and Implicit Laplacian Mesh Smoothing are shown as follows

7.2.1 $plane_n s.obj$

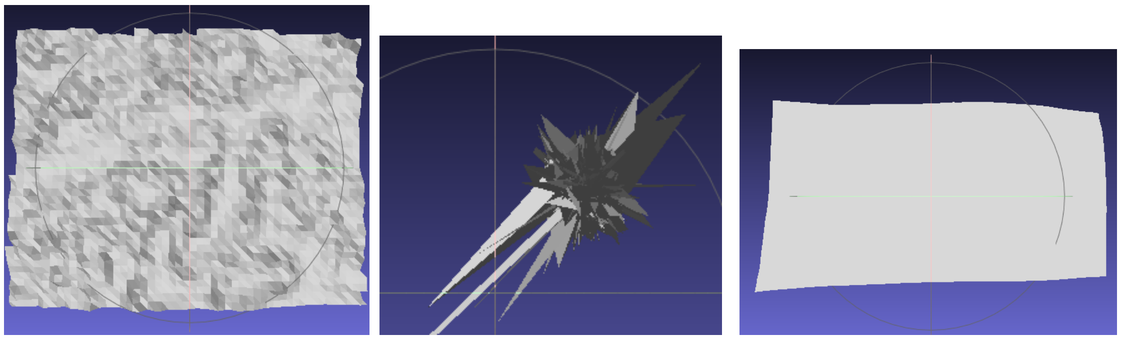
The following figures are the comparison results between explicit smoothing and implicit smoothing under iteration = 5, $lamda$ = 0.00001



(a) Origin Mesh (b) Explicit Laplacian (c) Implicit Laplacian

Figure 12: Visual results of $plane_n s.obj$ Under Smaller Step Size

The following figures are the comparison results between explicit smoothing and implicit smoothing under iteration = 5, $lamda$ = 0.0018

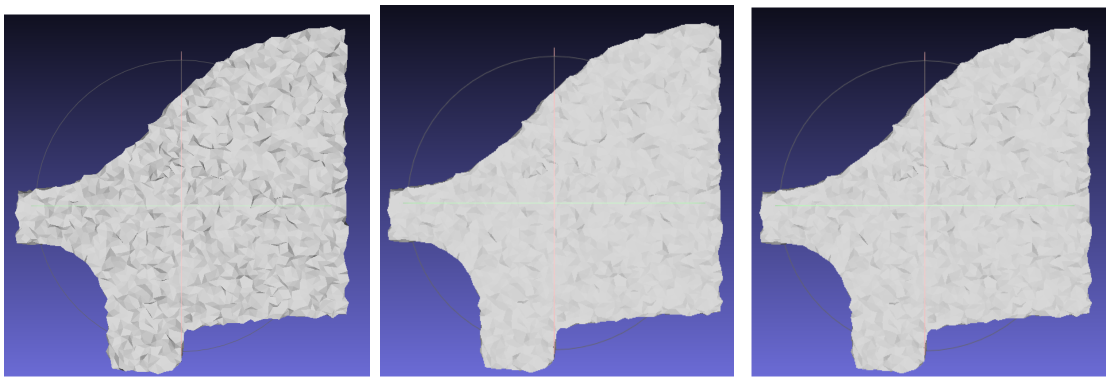


(a) Origin Mesh (b) Explicit Laplacian (c) Implicit Laplacian

Figure 13: Visual Results of $plane_n s.obj$ Under Bigger Step Size

7.2.2 $fandisk_n s.obj$

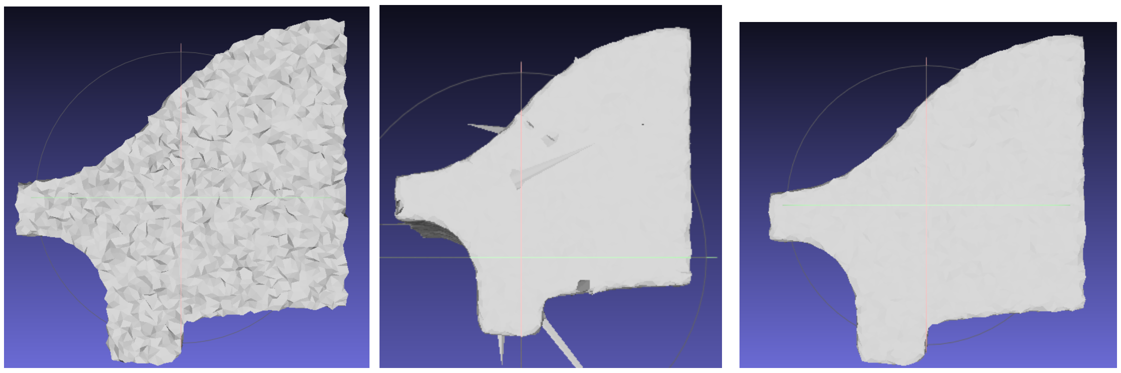
The following figures are the comparison results between explicit smoothing and implicit smoothing under iteration = 5, $lamda$ = 0.0005



(a) Origin Mesh (b) Explicit Laplacian (c) Implicit Laplacian

Figure 14: Visual results of $fandisk_ns.obj$ Under Smaller Step Size

The following figures are the comparison results between explicit smoothing and implicit smoothing under iteration = 5, $lamda$ = 0.0018



(a) Origin Mesh (b) Explicit Laplacian (c) Implicit Laplacian

Figure 15: Visual Results of $fandisk_ns.obj$ Under Bigger Step Size

7.2.3 Analysis

From the visual results above, we can observe that within a certain range, the larger the stepSize, the better the smoothing effect of the algorithm. Similarly, the greater the number of iterations, the better the smoothing effect. When stepSize is within a certain range, the Implicit algorithm improves the results only slightly over the Explicit algorithm. However, when stepSize is relatively large, the results from the Explicit algorithm show significant distortion, while the Implicit algorithm still achieves satisfying results. This phenomenon indicates that the latter algorithm is more stable and less prone to excessive smoothing leading to distortion.

8 Evaluation of Laplacian Mesh Denoising

8.1 Test Design

My test was designed as follows: I started with a model and added varying degrees of noise to it. Then, I used the results of Implicit Laplacian Mesh Denoising to smooth the noisy mesh. Finally, I evaluated the performance of my Laplacian Mesh Denoising by comparing the errors between the noisy mesh and the original mesh with the errors between the smoothed mesh and the original mesh, and by examining the changes from the noisy to the smoothed mesh.

I further evaluated and observed when the method works and when it fails by repeating the above test with different models to gain a more comprehensive understanding.

8.2 Result and Anaysis

8.2.1 Test Result of $fandisk_n s.obj$

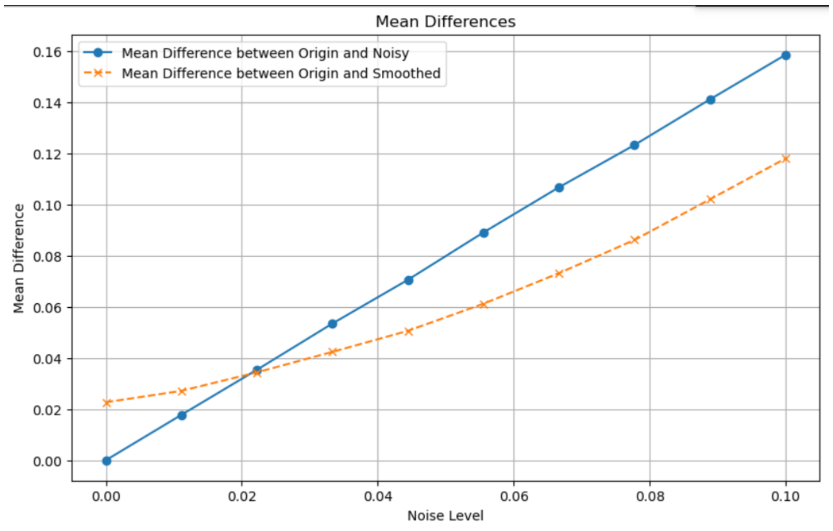
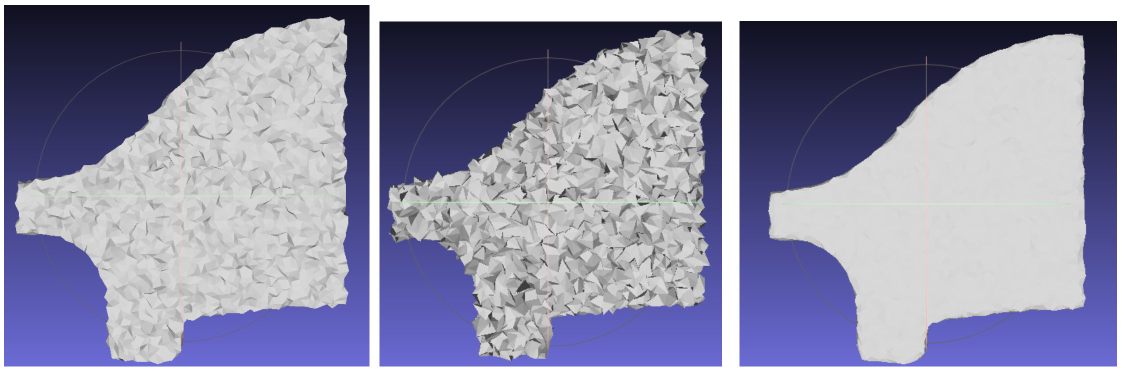


Figure 16: Mean Error Comparison



(a) Origin Mesh (b) Noisy Mesh (c) Smoothed Mesh

Figure 17: Visual Results of $fandisk_ns.obj$ Under Noisy and Smoothed

8.2.2 Test Result of *cube.obj*

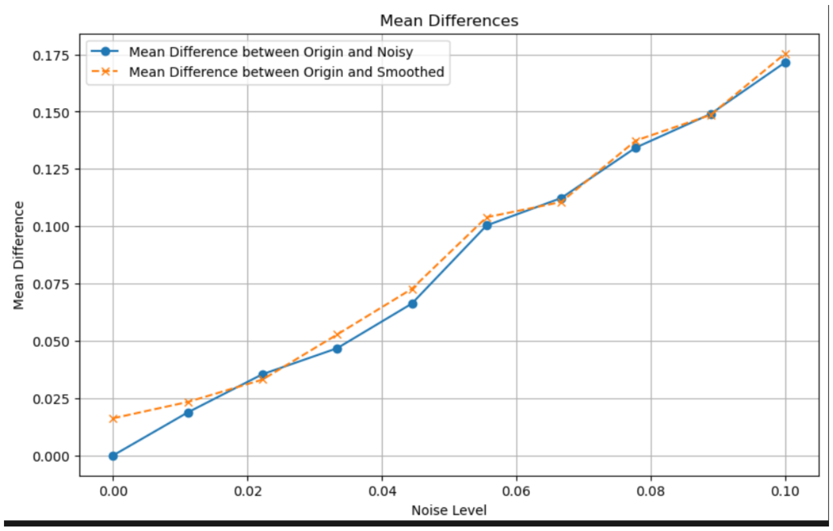
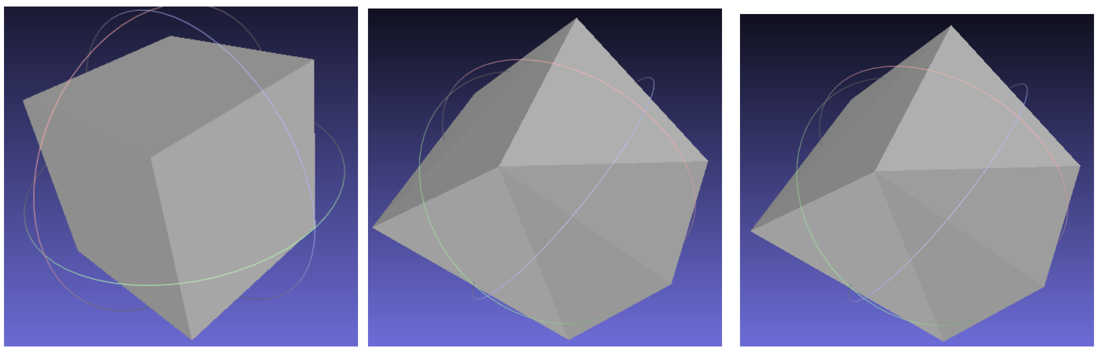


Figure 18: Mean Error Comparison



(a) Origin Mesh (b) Noisy Mesh (c) Smoothed Mesh

Figure 19: Visual Results of $fandisk_ns.obj$ Under Noisy and Smoothed

8.2.3 Analysis

Analyzing the results from the two sets of tests, we observe that our Implicit Laplacian Mesh Smoothing performs well when the models have a higher number of vertices and are inherently less smooth. However, when the models have fewer vertices and are already quite smooth, our algorithm does not yield as good results.

References