Acquisition and Processing of 3D
Assignment 2

Name: **Yinji Zhu** ID: **18062537** Email: **ucaby22@ucl.ac.uk**

March 24, 2019

Introduction

Noise is the key problem of handling range data or 3D scans [1]. In this case, denosing is an important thing needs to be done before processing dataset. In the assignment, Laplacian filtering is used to do mesh denosing, which requires to differentiate between object features and imperfections arising produced by random noise. Consequently, several methods such as uniform Laplace, non-uniform, mesh reconstructions, explicit Laplacian smoothing, and implicit Laplacian mesh smoothing need to be done in this assignment. All questions were written and the functions were placed in *mytools.cpp* and *mytools.h*.

Discrete Curvature and Spectral Meshes

Question 1

The first question is asked to calculate mean curvature H and Gaussian curvature K, which means that curvatures assign the same weighting to vertices and edges around. The absolute value of mean curvature H can be written as:

$$H = \frac{1}{2} \left\| \Delta_{\mathcal{S}} \mathbf{x} \right\|$$

where Δ and x represent Laplace-Beltrami and vertices respectively.

Knowing that the Laplace-Beltrami operator is done in uniform discretization, the function can be written as:

$$\Delta_{\text{uni}} \mathbf{x}_i := \frac{1}{|\mathcal{N}_1(v_i)|} \sum_{v_j \in \mathcal{N}_1(v_i)} (\mathbf{x}_j - \mathbf{x}_i) \approx -2H\mathbf{n}$$

where $\mathcal{N}_1(v_i)$ represents the valance of current v_i. Consequently, the current vertex has the weight -1 and each connectivity vertex has the weight $\frac{1}{\mathcal{N}_1(v_i)}$. As what can be seen from fig.1, v_j are adjacent points. In figure, five connected points exist to v_i, so weights of connectivity

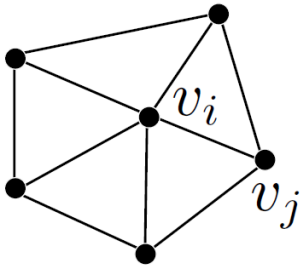


Figure 1: Connectivity vertices

vertices equal to 1/5 at the current v_i.
After the magnitude of H is obtained by the equation above, the sign of mean curvature needs to be judged. The sign of curvature is positive if the direction of surface gradient is the same with the surface normal. If the directions are opposite, the sign of curvature is negative, which meets the requirement of: $\Delta_{\text{uni}}\mathbf{x}_i := -2H\mathbf{n}$. The function *vertex_triangle_adjacency* in igl library is called in this part to obtain the adjacent vertices [2]. The number of valance can be obtained by getting *size()* of each row. Since Δ only has value when v_i and v_j are neighbors, most of the elements in Δ are zero. Consequently, Δ is set to a sparse matrix. Multiplying Δ with vertices, the magnitude of H can be obtained. The result of cow is shown below, which is set colors by *parula* in igl library:

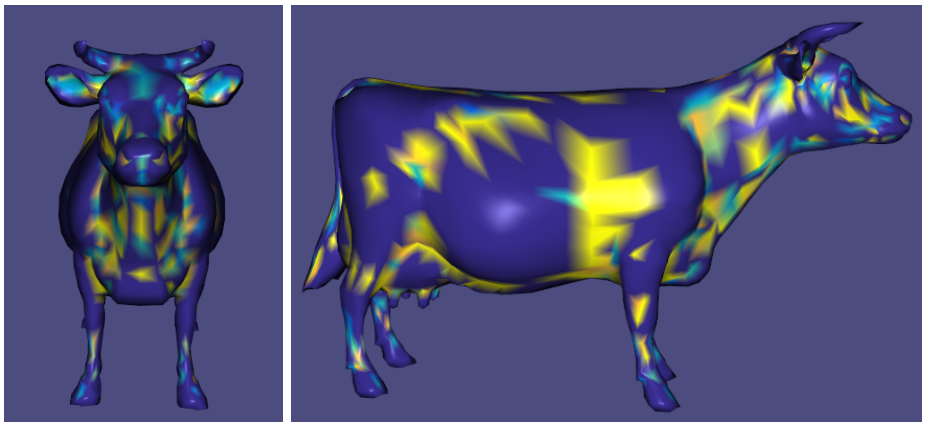
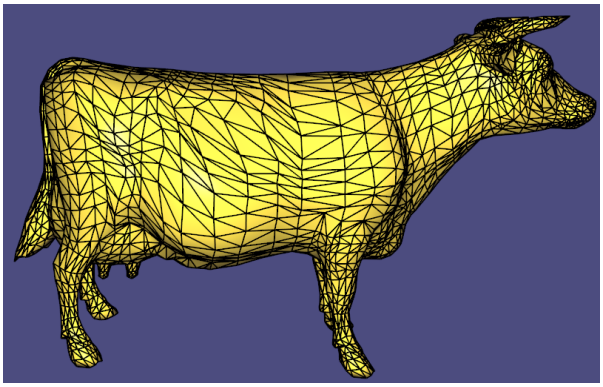


Figure 2: Output of uniform Laplace mean curvature

As what can be seen in fig.2, mean curvature in uniform Laplace is a quite inaccurate approximation of continuous curvature. In the places like cow horns and legs, mean curvature should have large changes since these places have large curvature, but the result does not meet the theory.

As can be seen in fig.3, triangulations are irregular. It will lead to non-zero H for planar meshes and tangential drift. Consequently, one possible way to fix this problem is to consider details of angles.

In this case, Gaussian curvatures are came up to show continuous curvature. Barycentric cells are chosen as the method to indicate area normalization A_i. Fig.4 shows the figure of barycentric cells.



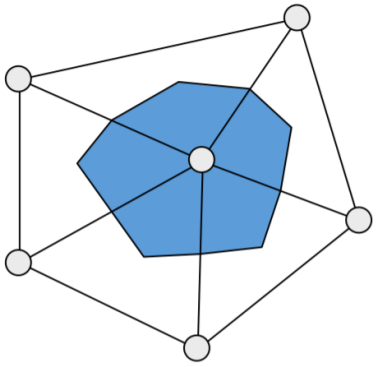


Figure 3: Barycentric cells

Figure 4: View of mesh

Barycentric cells connect edge midpoints and triangle barycenters. Therefore, blue area is the required area which represents $A(v_i)$. According to the definition of triangle centroids, the blue area equals to 1/3 area of the triangle, so $A(v_i)$ equals to 1/3 area of the total area which includes vertex v_i.

The formulation of Gaussian curvature can be written as:

$$K = \left(2\pi - \sum_j \theta_j\right) / A$$

Angles and areas can be calculated by vectors' dot product and cross product respectively. The result of cow is shown below.

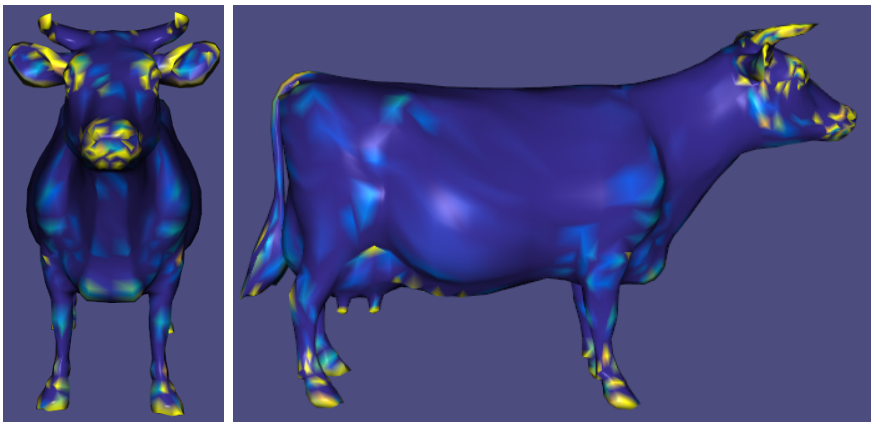


Figure 5: Output of discrete Laplace Gaussian curvature

The expression of Gaussian curvature is:

$$K = \kappa_1 \cdot \kappa_2$$

The expression shows that the Gaussian curvatures will have large values only if both of maximal and minimal curvatures are large enough. Hence, large Gaussian curvatures are only shown in the edges such as cow horns, nose, legs, etc. Comparing fig.5 with fig.2, discrete seems to show much better approximation of continuous curvature. However, the multiplication calculation

makes most of curvatures low, it is hard to tell the difference between other places. Consequently, it is still not the best way to approximate continuous curvature.

Question 2

In this part, vertex_triangle_adjacency in igl is used to calculate cotangent [2]. The equation of discrete Laplace-Beltrami in cotangent discretization can be written as:

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

where the cotangent matrix C can be written in the form of:

$$\mathbf{C}_{ij} = \begin{cases} \cot \alpha_{ij} + \cot \beta_{ij}, & i \neq j, j \in \mathcal{N}_1(v_i) \\ -\sum_{v_j \in \mathcal{N}_1(v_i)} (\cot \alpha_{ij} + \cot \beta_{ij}) & i = j \\ 0 & \text{otherwise} \end{cases}$$

Using the similar process comparing with uniform mean curvature and discrete Gaussian curvature, the result of cow can be seen in fig. 6.

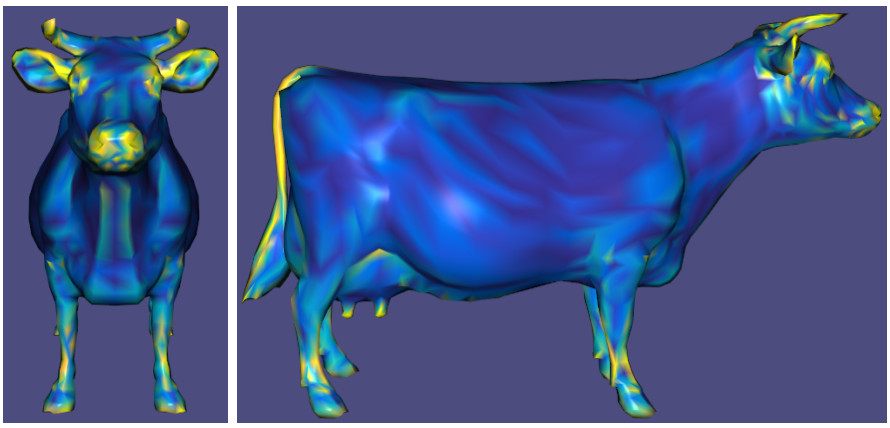


Figure 6: Output of cotangent Laplace mean curvature

Comparing with the algorithms in question, the quality of the estimated curvatures improves, which can see more accurate and gradual curvatures changes, for example, the tail of cow. Mean curvature shows medium value when only maximal is high. It shows high/small values when both of maximal and minimal are high/small. The cotangent Laplace mean curvature will not influence by irregular triangulations. The algorithm shows the changes of curvatures as well. In general, it is the best algorithm among the above three.

Question 3

In this question, Spectra library is used to calculate eigenvalues and eigenvectors [3].

Knowing that

$$\Delta \phi_i = \lambda_i \phi_i$$

Knowing that symmetric matrix has a much lower computation cost, the function can be re-written as:

$$M^{-1}C\phi_i = \lambda_i\phi_i$$

Then,

$$M^{-1/2}CM^{-1/2}M^{1/2}\phi_i = \lambda_i M^{1/2}\phi_i$$

Set $M^{1/2}\phi_i = y_i$ and $\tilde{\Delta} = M^{-1/2}CM^{-1/2}$, the function changes to:

$$\tilde{\Delta}y_i = \lambda_i y_i$$

Knowing that C and $M^{-1/2}$ are symmetric matrices, $\tilde{\Delta}$ is also a symmetric matrix. Consequently, the task changes to eigen decompose $\tilde{\Delta}$.

After eigen decomposition by extracting k smallest eigenvectors, the final projection can be written as [4]:

$$\tilde{x} = \sum_{i=1}^{k} < x, \phi_i >_m \phi_i$$

The results of cow using k = 5, 10, and 30 can be seen below:

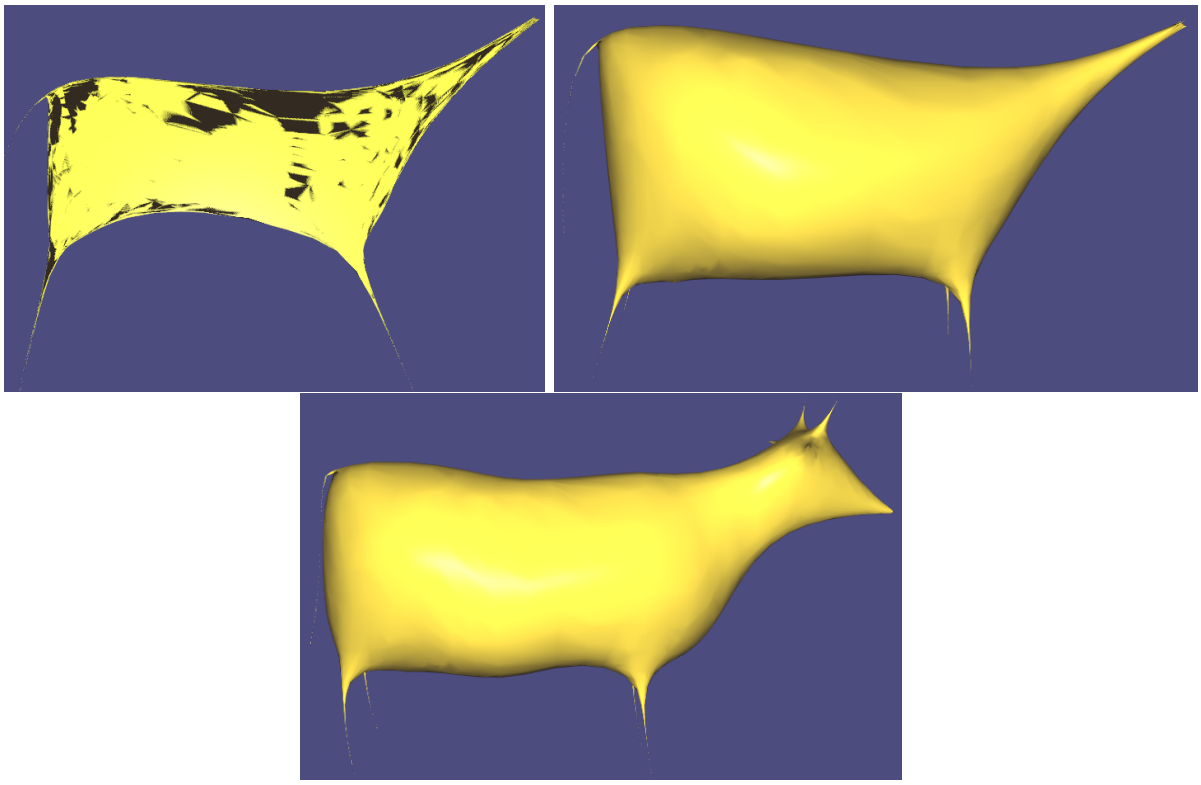


Figure 7: The results of cow using k = 5, 10, and 30 by eigen decomposition

Knowing that eigenvectors are natural vibrations and eigenvalues are natural frequencies, when computing more eigenvectors, it means that the graph captures more frequencies. Consequently, when the value of k increases, the result will show more features.

Question 5

The aim of this question is to implement explicit Laplacian mesh smoothing, using the equation:

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

where L is the multiplication of M^{-1} and C. In this equation, λ needs to be small enough to ensure stability.

The implementation uses bunny object, the left image in fig.8 is the original object and the right one is the result which set λ to 1×10^{-7} with 100 iterations.

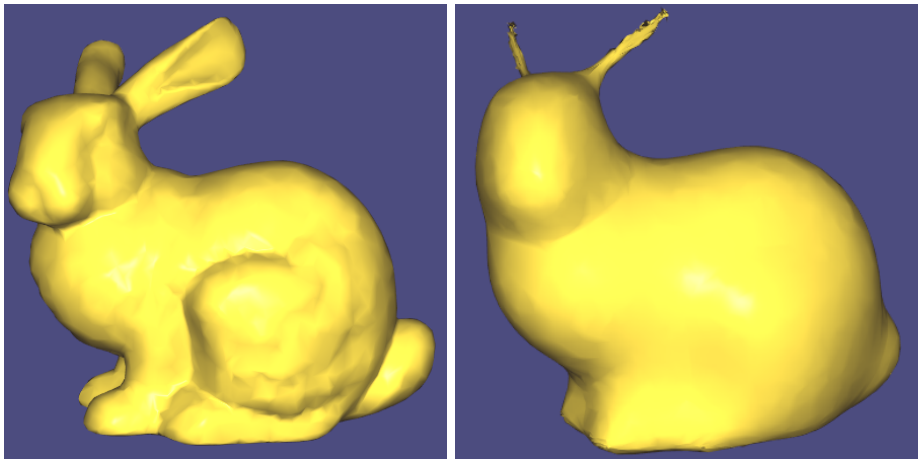


Figure 8: The results of explicit Laplician mesh smoothing before/ after

The choice of λ depends on mesh and structure. If the test object has thin edge, λ will be small. If not, λ will be large. λ less than 2×10^{-7} might be a good value to bunny object with small iteration. However, when the number of iterations increase, the image will generate zigzag as well. If want to achieve the same smoothing effect, smaller λ with more iterations have better effect than larger λ with less iterations. When the value of λ gradually increases and keep the same iteration, the overall shape can be smoother, but it will generate zigzag on the ear, which can be seen in the left image of fig.9. Consequently, a good λ step size should be a small value with large iteration.

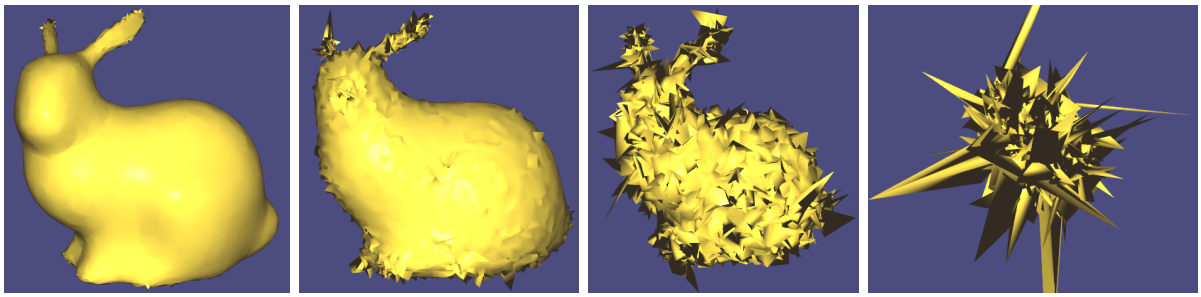


Figure 9: The results of explicit Laplician mesh smoothing when λ equals to 8, 30, 100, 1000 $\times 10^{-7}$

When the value of λ gradually increases to 8×10^{-7}, 3×10^{-6}, 1×10^{-5}, and 1×10^{-4}, the surface of bunny will generate more and more zigzags. When the step size is too large, the

shape of bunny cannot be recognized.

The figure of cube is also tried in this part. When λ is small, the figure of cube keeps the same. It is because the surface of cube is smooth enough. However, when λ increases to around

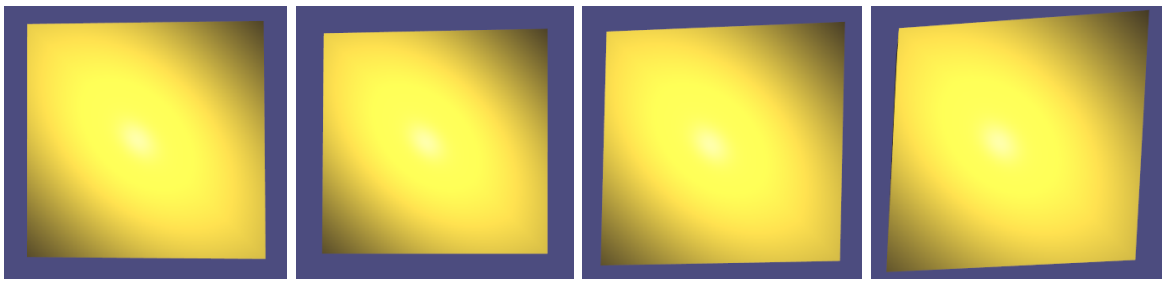


Figure 10: The results of explicit Laplician mesh smoothing when λ equals to 1, 10, 30, 50 $\times 10^{-3}$

1×10^{-3}, the cube starts to have large deformation. Comparing the value of λ between fig.9 and fig.10, it can be found that different objects need to choose different value of λ.

However, as what can be seen in right image of fig.8, when λ is small enough, it will still cause some zigzags. Consequently, the method of implicit Laplacian is required.

Question 6

The equation of implicit Laplacian mesh smoothing can be written as:

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

Since L is not a symmetric matrix, multiplying M, the equation can be re-written as:

$$(\mathbf{M} - \lambda \mathbf{L}_w)\,\mathbf{P}^{(t+1)} = \mathbf{M}\mathbf{P}^{(t)}$$

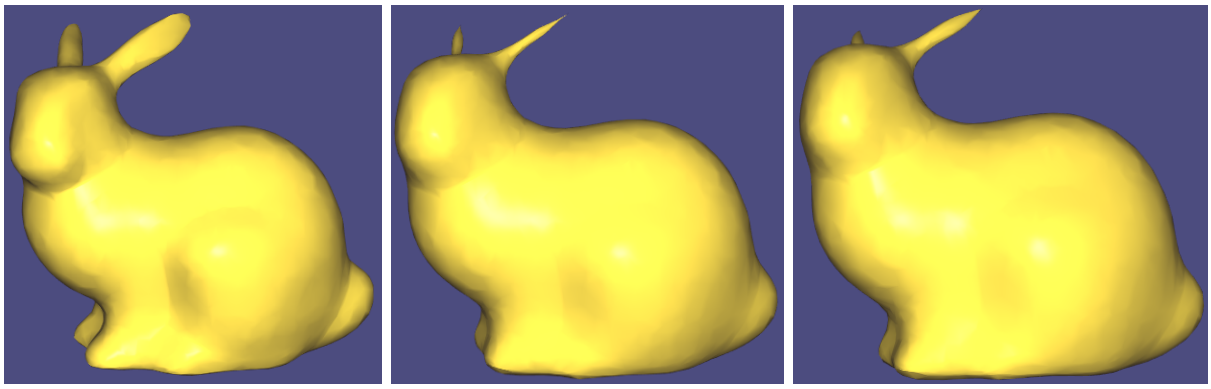


Figure 11: The results of implicit Laplician mesh smoothing (Left: $\lambda = 1 \times 10^{-6}$, iteration = 10 Middle: $\lambda = 1 \times 10^{-6}$, iteration = 20 Right: $\lambda = 2 \times 10^{-6}$, iteration = 10)

The division uses sparse Cholesky method called *SimplicialLLT* in Eigen library. Set λ to 1×10^{-6}, the bunny result after 10 and 20 iterations can be seen in left and middle images of

fig.11.

As what can be seen in the graph, the surface of the middle image is still smooth rather than generating zigzags. It proves that implicit Laplacian is much more stable comparing with explicit Laplacian, since this algorithm uses non linear solver optimizing the position of all vertices in the same time. Comparing the left image with the right one, when increasing the value of λ, the result is still smooth. Consequently, implicit integration is unconditionally stable. However, if the step size is too large, the result will have error. Comparing the value of λ step size, implicit Laplacian uses larger λ than explicit Laplacian which can increase the speed of converge. Moreover, implicit Laplacian needs less iterations than explicit Laplacian.

The changes of cube can be seen below:

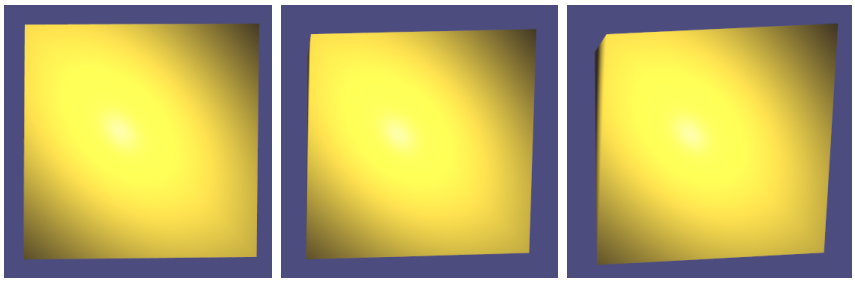


Figure 12: The results of implicit Laplician mesh smoothing when λ equals to 10, 30, 50 $\times 10^{-3}$

Comparing the graph with fig.10, it seems that the figures does not have large differences when object does not have many high frequencies.

Then, comparing the explicit and implicit effect of the object cow. In the left image, the explicit process generates zigzags in the cow's horns as well. However, in the right image, the implicit process makes a smoother cow without zigzags.

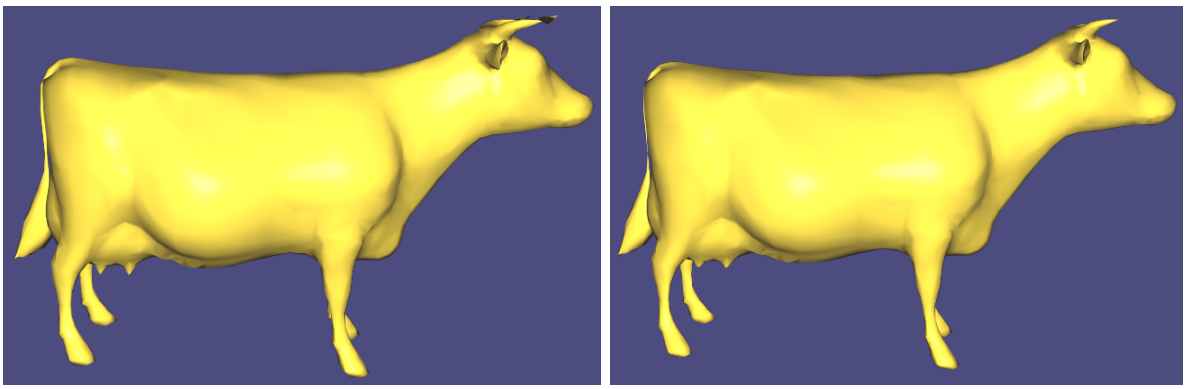


Figure 13: The explicit (left) and implicit (right) effect of the object cow

Comparing the whole objects shown above, implicit Laplacian has much less computational cost.

Question 7

Noise is generated by *default_random_engine* from std library to each point. The mean value is set to zero so a Gaussian white noise is created. After denosing by implicit Laplacian mesh, the value of error is set to the norm of two vertex vectors.

Changing the value of standard deviation in noise and setting step size λ to a fixed value $\lambda = 5 \times 10^{-7}$, the figure of error verse iteration can be seen below:

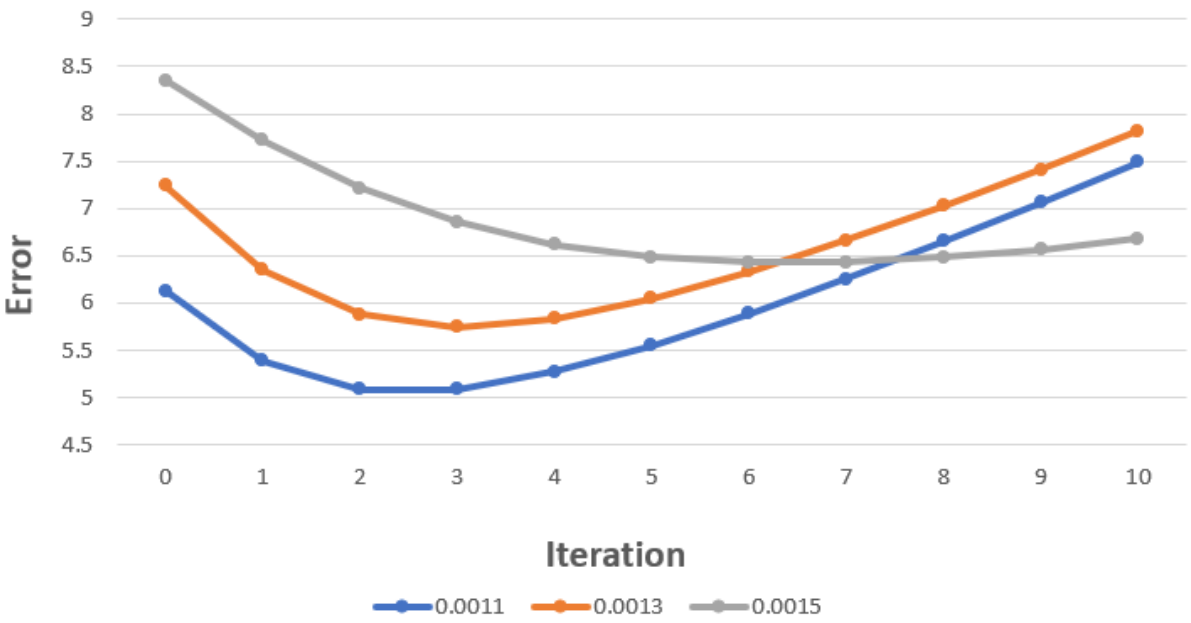


Figure 14: The plot of error verse iteration

As what can be seen from fig.14, the blue curve has the deviation of 0.0011, the orange curve has the deviation of 0.0013, and the grey curve has the deviation of 0.0015. The grey has the largest error at the beginning, since it has the larger standard deviation to generate larger noise. Then, errors decrease in all the three curves, since the implicit Laplacian step gets reduces the noise. If the object has smaller noise, the error will go to local minimum earlier, and then the error will increase. It is caused by implicit Laplacian step smooths high frequency features at the same time, which will lead to the difference between the original bunny with the bunny after denoising step. If more iterations are done, the three curves might have similar errors later, since they have the same step size.

Setting the value of standard deviation to 0.0013 with 3 iterations and 0.0015 with 7 iterations, since they have the lowest error, the figure of bunnies can be seen below:
Comparing the three images in fig.15, when the noise is higher, the bunny can keep less features after denosing.

In general, the Laplacian mesh denosing has excellent performance in flat region. However, in the region like edge which has high frequency, the performance of this method is not very well since noises are always in high frequency.

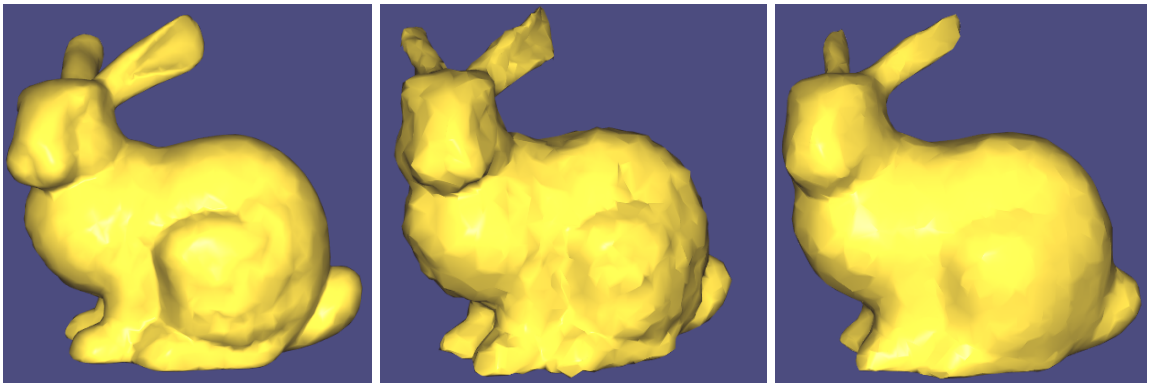


Figure 15: Denoised bunnies (Left: original Middle: $\lambda = 0.0013$ Right: $\lambda = 0.0015$)

References

[1] N. J. Mitra. (2019) Assignment 2. [Online]. Available: https://moodle-1819.ucl.ac.uk/pluginfile.php/370194/mod_assign/introattachment/0/assignment_2_0308.pdf?forcedownload=1

[2] D. Panozzo. (2014) vertex_triangle_adjacency.cpp. [Online]. Available: https://github.com/libigl/libigl/blob/master/include/igl/vertex_triangle_adjacency.cpp

[3] Spectra. (2019) Spectra c++ library for large scale eigenvalue problems. [Online]. Available: https://spectralib.org/

[4] N. J. Mitra. (2019) laplace_beltrami_and_eigenanalysis. [Online]. Available: https://moodle-1819.ucl.ac.uk/pluginfile.php/1425455/mod_resource/content/1/laplace_beltrami_and_eigenanalysis.pdf