Acquisition and Processing of 3D Geometry

Coursework 2

Binglun Wang ucabbw5@ucl.ac.uk

March 20, 2023

Mesh denoising is a very important topic, and this report briefly introduces related methods and discusses some experimental findings. In the first three sections, we will explore some basic concepts. Section 1 discusses the relationship between Laplacian value and mean curvature based on uniform discretization and discusses Gaussian curvature. Section 2 introduces the First and Second Fundamental forms and normal curvature through an example of a parametric surface function. Section 3 introduces the Laplacian value and mean curvature obtained based on Discrete Laplace-Beltrami. Section 4 extracts the main features from a spectral analysis perspective and smooths the mesh through feature reconstruction. Section 5 discusses an explicit diffusion flow-based smooth mesh method, section 6 discusses an implicit diffusion flow-based smooth mesh method, and section 7 discusses the impact of increasing noise on denoising methods. All experimental codes can be found at the end of the report.

1 Uniform Laplace

1.1 Method

By way of uniform discretisation, the Laplace value of a point vi is:

$$\Delta_{\text{uni}} f(v_i) := \frac{1}{|\mathcal{N}_1(v_i)|} \sum_{v_j \in \mathcal{N}_1(v_i)} (f(v_j) - f(v_i)) \tag{1}$$

where $\mathcal{N}_1(v_i)$ is the number of neighbors of v_i

We can directly use equation 1 to calculate each vertex's laplacian value.

Also, we can build a Laplacian operator of uniform discretisation:

$$\Delta_{\text{uni}} = \begin{bmatrix} -1 & \frac{1}{|\mathcal{N}_1(v_1)|} * e_{1,2} & \cdots & \frac{1}{|\mathcal{N}_1(v_1)|} * e_{1,n} \\ \frac{1}{|\mathcal{N}_1(v_2)|} * e_{2,1} & -1 & \cdots & \frac{1}{|\mathcal{N}_1(v_2)|} * e_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{|\mathcal{N}_1(v_n)|} * e_{n,1} & \frac{1}{|\mathcal{N}_1(v_n)|} * e_{n,2} & \cdots & -1 \end{bmatrix} \tag{2}$$

where $e_{i,j} = 1$ if an edge between in i and j. Otherwise, $e_{i,j} = 0$

$$\Delta_{\text{uni}} f(v_i) = -2H\mathbf{n} \tag{3}$$

In this coursework, we only focus on the **absolute** value of mean curvature. So,

$$H = \frac{1}{2} \|\Delta_{\text{uni}} f(v_i)\| \tag{4}$$

Where H is the mean curvature, $\mathbf{n}$ is the normal of vertex v_i.

Gaussian curvature K at a vertex v_i.

$$K = \left(2\pi - \sum_j \theta_j\right) / A_i \tag{5}$$

Where A_i is the area around the vertex v_i, $\sum_j \theta_j$ is the sum of angles around vertex v_i

1.2 Implementation

The space complexity of the Laplacain operator is $O(n^2)$, where n is the number of points in the mesh. So when n is very large, it is easy to run out of space. However, the graph of mesh is usually sparse. So, the Laplacian operator is usually sparse. So, I used the scipy.sparse to build the Laplacian matrix. Also, I use numpy vectorisation to build the matrix to make processing fast (See code at the end of report).

In Gaussian curvature. I iterate through each face to calculate the angles and areas corresponding to each vertex. Each face has a corresponding 3 angles, and I add the angles of each face to the corresponding vertices. So end up each vertex have the sum of angles around it. For the area A_i, I used the Barycentric cells method, which is easily calculated and has good estimates. When we connect edge midpoints and triangle barycenters, each vertex area can be represented by the blue area Figure 1. So I add face areas to the corresponding three vertices. The final result divided by 3 is the area corresponding to each vertex. Since we can use trimesh.face areas in the assignment requirements, I called it directly in my implementation and for the angle, I used the dot product of vectors to calculate the angles (See code at the end of report).

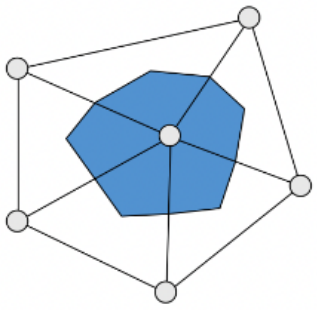


Figure 1: Barycentric cells area

1.3 Experiments and Discussion

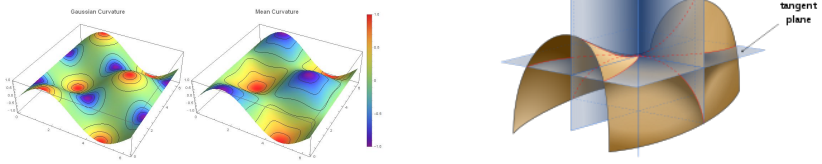
The principal curvatures at vertex p, denoted k_1 and k_2, are the maximum and minimum values of this curvature, like Figure 2 b. Here are two formulas:

The mean of two principal curvatures is the mean curvature, H.

$$H = \frac{k_1 + k_2}{2} \tag{6}$$

The product $k_1 k_2$ of the two principal curvatures is the Gaussian curvature, K:

$$K = k_1 k_2 \tag{7}$$



(a) Mean curvature and Gaussian curvature [1]

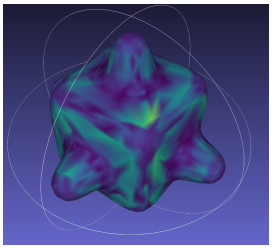
(b) principal curvatures [2]

Figure 2: Gaussian curvature results, are visualised by setting all mean curvature above 3 to 3 and those below -3 to -3 and normalisation.

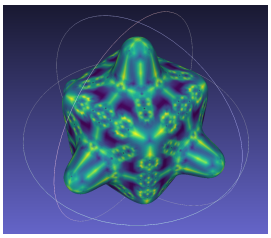
As can be seen in Figure 2, the uniform mean curvature has negative values at the convex points of the surface, positive values at the concave points, and near zero values at the saddle points. But in this coursework, we only focus on absolut value. So each concave-convex vertex should have high mean curvature.

Comparing a and b in Figure 3, We can see that when the data size is small, due to the influence of fine texture details, the variation of the results obtained based on uniform mean curvature is discontinuous and does not show the values of high mean curvature on convex and concave surfaces. When the data size of the model is very large, the mean curvature is more continuous, but some strange high values of mean curvature were obtained based on uniform mean curvature because the vertices are not uniform. This phenomenon is more evident in Figure 3c, where it can be seen that the vertices circled in red are surrounded by non-uniformly distributed vertices, and each vertex has a different contribution to the Laplacian value at that point. Also, for complex textures such as those of cows and rabbits in Figure 3 d, the method based on uniform mean curvature for 2,000 to 3,000 points is still insufficient to achieve good results.

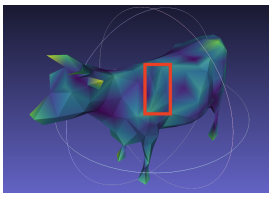
The corresponding Gaussian curvature results are as expected, again the convex-concave points show a bright colour (high values) the saddle points show a dark colour (low values). For some detailed textures, such as those on the bumpy cube, the gaussian curvature does not show up when the number of points is low, but the general texture is correct and does not like in Figure 3 a.



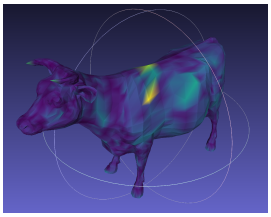
(a) bumpy cube with 1502 points



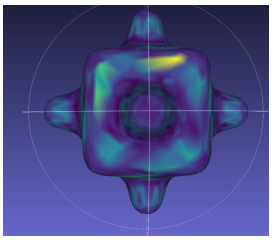
(b) bumpy cube with 19970 points



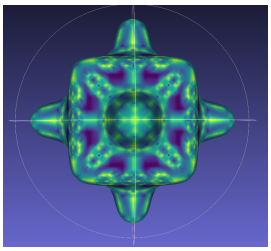
(c) cow with 304 points



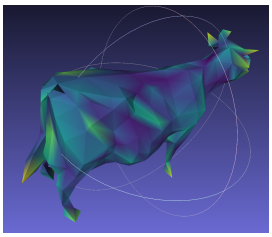
(d) cow with 2759 points



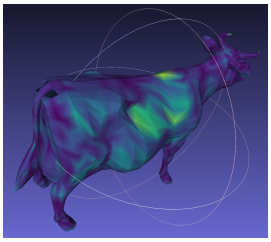
(e) bumpy cube with 1502 points



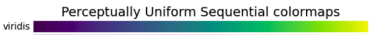
(f) bumpy cube with 19970 points



(g) cow with 304 points



(h) cow with 2759 points



(i) Lower values are darker colours, higher values are brighter colours

Figure 3: Uniform mean curvature results, are visualised by setting all mean curvature above 3 to 3 and those below -3 to -3 and normalisation. The same object results are shown from different views in the same columns in the figure.

2 First and Second Fundamental forms

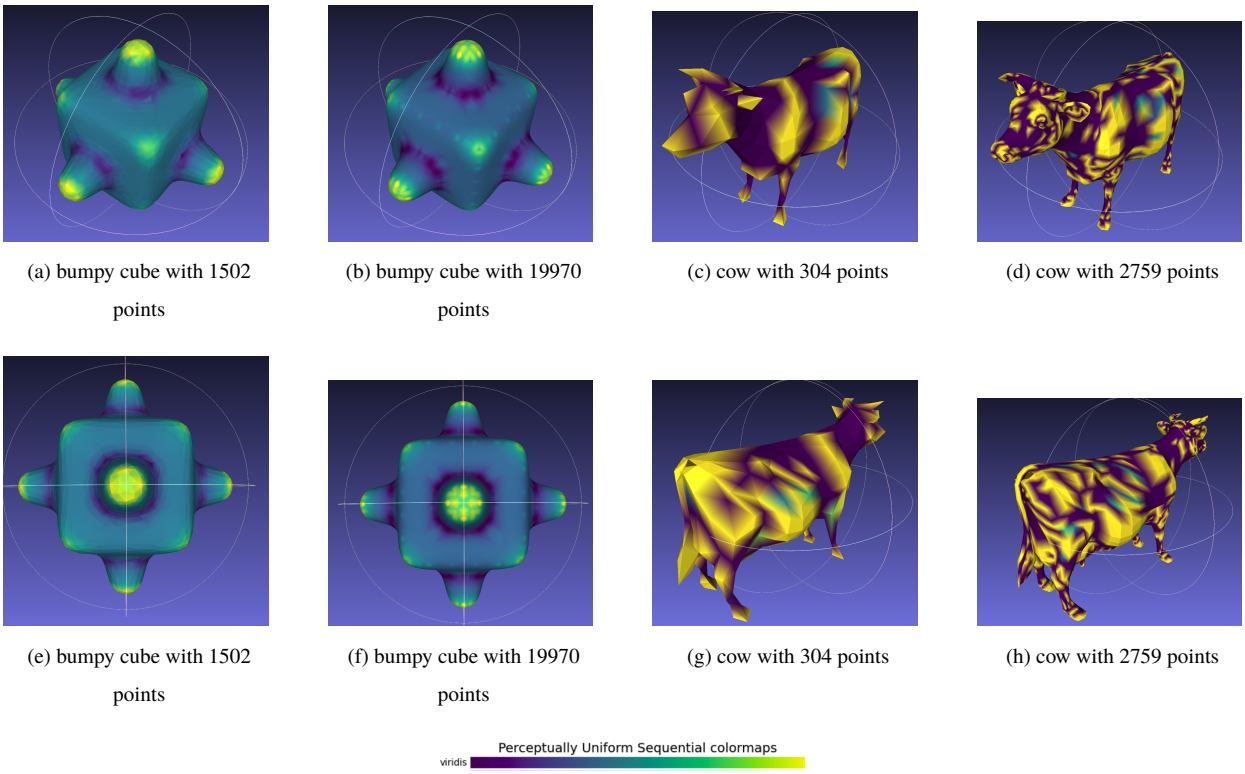
$$p(u,v) = \begin{pmatrix} a\cos(u)\sin(v) \\ b\sin(u)\sin(v) \\ c\cos(v) \end{pmatrix} \tag{8}$$

$$\begin{aligned} X_u = \frac{\partial p(u,v)}{\partial u} &= \begin{pmatrix} -a\sin(v)\sin(u) \\ b\sin(v)\cos(u) \\ 0 \end{pmatrix} \\ X_v = \frac{\partial p(u,v)}{\partial v} &= \begin{pmatrix} a\cos(u)\cos(v) \\ b\sin(u)\cos(v) \\ -c\sin(v) \end{pmatrix} \end{aligned} \tag{9}$$

First fundamental form:

$$\begin{Vmatrix} E & F \\ F & G \end{Vmatrix} = \begin{Vmatrix} X_u^T X_u & X_u^T X_v \\ X_v^T X_u & X_v^T X_v \end{Vmatrix} \tag{10}$$

Where,



(a) bumpy cube with 1502 points

(b) bumpy cube with 19970 points

(c) cow with 304 points

(d) cow with 2759 points

(e) bumpy cube with 1502 points

(f) bumpy cube with 19970 points

(g) cow with 304 points

(h) cow with 2759 points

(i) Lower values are darker colours, higher values are brighter colours

Figure 4: Gaussian curvature results, are visualised by setting all mean curvature above 3 to 3 and those below -3 to -3 and normalisation. The same object results are shown from different views in the same columns in the figure.

$$
\begin{aligned}
E &= X_u^T X_u = a^2 \sin(u)^2 \sin(v)^2 + b^2 \sin(v)^2 \cos(u)^2 \\
&= (a^2 \sin(u)^2 + b^2 \cos(u)^2) \sin(v)^2 \\
F &= X_u^T X_v = X_v^T X_u = -a^2 \sin(u) \sin(v) \cos(u) \cos(v) + b^2 \sin(u) \sin(v) \cos(u) \cos(v) \\
&= (b^2 - a^2) \sin(u) \sin(v) \cos(u) \cos(v) \\
G &= X_v^T X_v = a^2 \cos(u)^2 \cos(v)^2 + b^2 \sin(u)^2 \cos(v)^2 + c^2 \sin(v)^2
\end{aligned}
\tag{11}
$$

Second fundamental form:

$$
\begin{Vmatrix} e & f \\ f & g \end{Vmatrix} = \begin{Vmatrix} X_{uu}^T n & X_{uv}^T n \\ X_{uv}^T n & X_{vv}^T n \end{Vmatrix}
\tag{12}
$$

Where,

$$
\begin{aligned}
X_{uu} = \frac{\partial^2 p(u,v)}{\partial u^2} &= \begin{pmatrix} -a\sin(v)\cos(u) \\ -b\sin(v)\sin(u) \\ 0 \end{pmatrix} \\
X_{vv} = \frac{\partial^2 p(u,v)}{\partial v^2} &= \begin{pmatrix} -a\cos(u)\sin(v) \\ -b\sin(u)\sin(v) \\ -c\cos(v) \end{pmatrix} \\
X_{uv} = \frac{\partial^2 p(u,v)}{\partial u \partial v} &= \begin{pmatrix} -a\cos(v)\sin(u) \\ b\cos(v)\cos(u) \\ 0 \end{pmatrix}
\end{aligned} \tag{13}
$$

$$
n = \frac{X_u \otimes X_v}{||X_u \otimes X_v||} \tag{14}
$$

Where $\otimes$ is the cross product.

$$
\begin{aligned}
X_u \otimes X_v &= \begin{pmatrix} -bc\sin(v)^2\cos(u) \\ -ac\sin(v)^2\sin(u) \\ -ab\sin(v)\cos(v)(\sin^2(u)+\cos^2(u)) \end{pmatrix} \\
&= \begin{pmatrix} -bc\sin(v)^2\cos(u) \\ -ac\sin(v)^2\sin(u) \\ -ab\sin(v)\cos(v) \end{pmatrix}
\end{aligned} \tag{15}
$$

So,

$$
\begin{aligned}
e/||X_u \otimes X_v|| &= abc\sin(v)^3\cos(u)^2 + abc\sin(v)^3\sin(u)^2 \\
&= abc\sin(v)^3\left(\sin(u)^2+\cos(u)^2\right) \\
&= abc\sin(v)^3 \\
e &= \frac{abc\sin(v)^3}{||X_u \otimes X_v||} \\
f/||X_u \otimes X_v|| &= abc\cos(v)\sin(v)^2\cos(u)\sin(u) - abc\cos(v)\sin(v)^2\cos(u)\sin(u) \\
&= 0 \\
f &= 0 \\
g/||X_u \otimes X_v|| &= abc\sin(v)^3\cos(u)^2 + abc\sin(v)^3\sin(u)^2 + abc\sin(v)\cos(v)^2 \\
&= abc\sin(v)^3 + abc\sin(v)\cos(v)^2 \\
&= abc\sin(v)(\sin(v)^2+\cos(v)^2) \\
&= abc\sin(v) \\
g &= \frac{abc\sin(v)}{||X_u \otimes X_v||}
\end{aligned} \tag{16}
$$

where,

$$||X_u \otimes X_v|| = \sqrt{(-bc\sin(v)^2\cos(u))^2 + (-ac\sin(v)^2\sin(u))^2 + (-ab\sin(v)\cos(v))^2} \tag{17}$$

Then we need to calculate the normal curvature function for point $(a, 0, 0)$

Take $(a, 0, 0) = p(u, v)$

$$\begin{cases} a\cos(u)\sin(v) = a \\ b\sin(u)\sin(v) = 0 \\ c\cos(v) = 0 \end{cases}$$

Because $u \in [0, 2\pi), v \in [0, \pi)$

So, $u = 0, v = \frac{\pi}{2}$

$$\begin{aligned} E &= (a^2\sin(u)^2 + b^2\cos(u)^2)\sin(v)^2 \\ &= b^2 \\ F &= (b^2 - a^2)\sin(u)\sin(v)\cos(u)\cos(v) \\ &= 0 \\ G &= a^2\cos(u)^2\cos(v)^2 + b^2\sin(u)^2\cos(v)^2 + c^2\sin(v)^2 \\ &= c^2 \end{aligned} \tag{18}$$

$$\begin{aligned} ||X_u \otimes X_v|| &= \sqrt{(-bc\sin(v)^2\cos(u))^2 + (-ac\sin(v)^2\sin(u))^2 + (-ab\sin(v)\cos(v))^2} \\ &= bc \\ e &= \frac{abc\sin(v)^3}{||X_u \otimes X_v||} \\ &= a \\ f &= 0 \\ g &= \frac{abc\sin(v)}{||X_u \otimes X_v||} \\ &= a \end{aligned} \tag{19}$$

At $(a, 0, 0)$ the normal curvature is:

$$\begin{aligned} \kappa_n(\bar{\mathbf{t}}) = \frac{\bar{\mathbf{t}}^T\mathbf{I}\bar{\mathbf{t}}}{\bar{\mathbf{t}}^T\mathbf{II}\bar{\mathbf{t}}} &= \frac{eA^2 + 2fAB + gB^2}{EA^2 + 2FAB + GB^2} \\ &= \frac{a(A^2 + B^2)}{b^2A^2 + c^2B^2} \end{aligned} \tag{20}$$

$$\mathbf{t} = A\mathbf{x}_u + B\mathbf{x}_v$$

$$\bar{\mathbf{t}} = (A, B)$$

where AB are our function variables.

3 Non-uniform (Discrete Laplace-Beltrami)

3.1 Method

By way of discrete Laplace-Beltrami, the Laplace value of a point v_i is:

$$\Delta_S f(v_i) := \frac{1}{2A_i} \sum_{v_j \in \mathcal{N}_1(v_i)} (\cot \alpha_{ij} + \cot \beta_{ij}) (f(v_j) - f(v_i)) \tag{21}$$

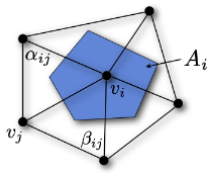


Figure 5: alpha and beta

where cot is the cotangent function.

Also, we can build a Laplacian operator of discrete Laplace-Beltrami:

$$\begin{aligned} \mathbf{L} &= \mathbf{M}^{-1}\mathbf{C} \\ \mathbf{L}, \mathbf{M}^{-1}, \mathbf{C} &\in \mathbf{R}^{n \times n} \end{aligned} \tag{22}$$

where,

$$\begin{aligned} \mathbf{C}_{ij} &= \begin{cases} \cot \alpha_{ij} + \cot \beta_{ij}, & i \neq j, j \in \mathcal{N}_1(v_i) \\ -\sum_{v_j \in \mathcal{N}_1(v_i)} (\cot \alpha_{ij} + \cot \beta_{ij}) & i = j \\ 0 & \text{otherwise} \end{cases} \\ \mathbf{M}^{-1} &= \operatorname{diag}\left(\ldots, \frac{1}{2A_i}, \ldots \right) \end{aligned} \tag{23}$$

3.2 Implementation

As mentioned in the first section, I used the same method to obtain the area. For angles, we can observe that in a triangle mesh, if we iterate all three angles of each triangle face, we will exactly iterate two angles corresponding to each edge once. Using this observation, I traverse the angles of each face and find the corresponding vertices. However, for a non-continuous mesh, such as a mesh with a corner vertex j in Figure 6, only have edges like i, j that only have one corresponding angle, this method may encounter the wrong Laplacian value, which will be discussed in detail later.

3.3 Experiments and Discussion

Compared to the first section, this method's mean curvature Figure 7 is very consistent with expectations. Both convex and concave points have high values of mean curvature using this method.

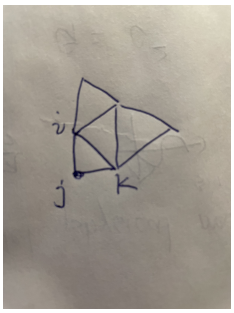
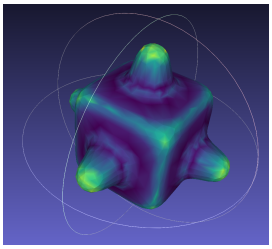
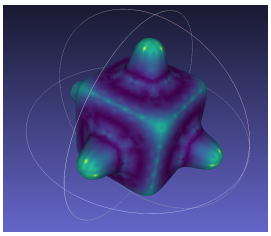


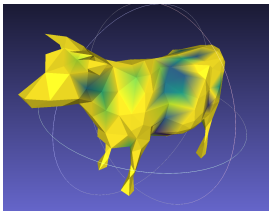
Figure 6: wrong example



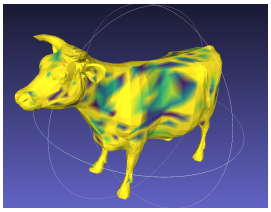
(a) bumpy cube with 1502 points



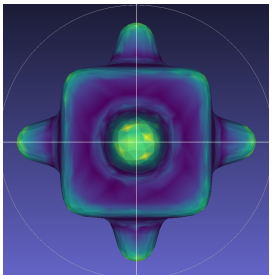
(b) bumpy cube with 19970 points



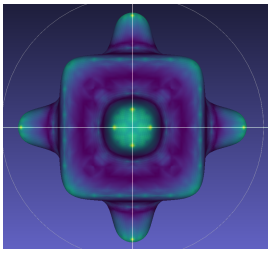
(c) cow with 304 points



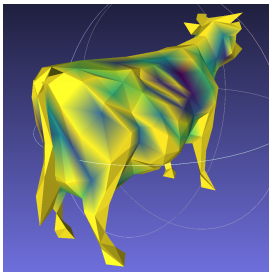
(d) cow with 2759 points



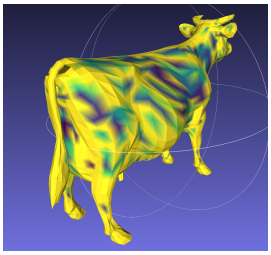
(e) bumpy cube with 1502 points



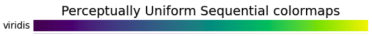
(f) bumpy cube with 19970 points



(g) cow with 304 points



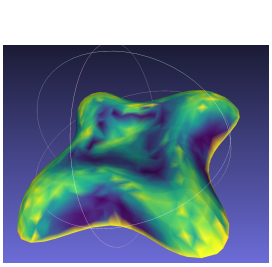
(h) cow with 2759 points



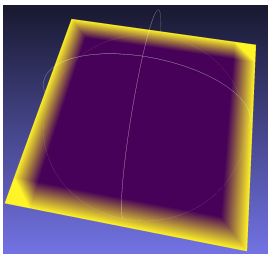
(i) Lower values are darker colours, higher values are brighter colours

Figure 7: Discrete Laplace-Beltrami results, are visualised by setting all mean curvature above 3 to 3 and those below -3 to -3 and normalisation. The same object results are shown from different views in the same columns in the figure.

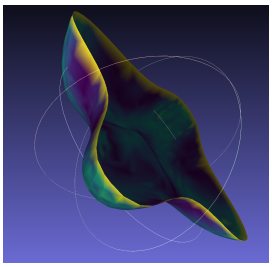
"non-continuous" mesh boundary: In Figure 8 ab, for these two meshes, we can observe that the mean curvature along their boundaries has abnormally large values. Especially for the plane mesh, the mean curvature along its boundary should be zero. The reason for this is that these meshes are not continuous, and the boundary vertices only have neighbours on the inner side, which leads to a large mean curvature. One solution is shown in the Figure 8 c, we can calculate the mean curvature of boundary points by adding some non-existent points based on symmetry. I would like to emphasize that the purpose of this report is not to solve all problems, so here I only discuss a possible solution.



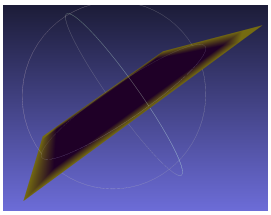
(a) lilium with 786 points



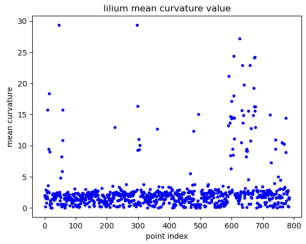
(b) plane with 100 points



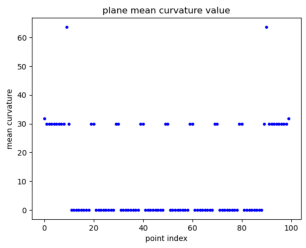
(c) different view with a



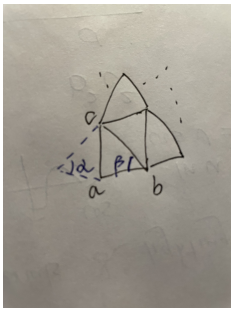
(d) different view with b



(e) lilium mean curvature value



(f) plane mean curvature value



(g) discussed solution

Figure 8: Discrete Laplace-Beltrami results, are visualised by setting all mean curvature above 3 to 3 and those below -3 to -3 and normalisation.

4 Modal analysis

4.1 Method

In the Fourier domain, we can observe that taking the second derivative corresponding eigenvalues and eigenvectors of the Laplace operator. The absolute values of the smaller eigenvalues correspond to the low-frequency information on the mesh. So we can select the top smaller absolute eigenvalues corresponding eigenvectors as the top main component of mesh.

$$\Delta\left(e^{2\pi \mathrm{i}\omega x}\right)=\frac{\mathrm{d}^2}{\mathrm{d}x^2}e^{2\pi \mathrm{i}\omega x}=-(2\pi\omega)^2e^{2\pi \mathrm{i}\omega x} \tag{24}$$

The Laplace operator can be very large, and we typically need to use sparse matrix computations. For sparse matrices, computations are faster if the matrix is symmetric. However, the Laplace operator is not a symmetric matrix. But $M^{-}1, C$ are two symmetric matrices. So we can use a trick:

$$
\begin{aligned}
\Delta\varphi_i &= \lambda_i\varphi_i \\
M^{-1}CM^{-\frac{1}{2}}M^{\frac{1}{2}}\varphi_i &= \lambda_i\varphi_i \\
M^{-\frac{1}{2}}CM^{-\frac{1}{2}}M^{\frac{1}{2}}\varphi_i &= \lambda_i M^{\frac{1}{2}}\varphi_i \\
D\alpha_i &= \lambda_i\alpha_i
\end{aligned}
\tag{25}
$$

Where D is the symmetric matrix

$$
\begin{aligned}
D &= M^{-\frac{1}{2}}CM^{-\frac{1}{2}} \\
\varphi_i &= M^{-\frac{1}{2}}\alpha_i
\end{aligned}
\tag{26}
$$

Then we can use eigenvectors to reconstruct mesh.

$$
\begin{array}{ccc}
\mathbf{x} := [x_1, \ldots, x_n] & \mathbf{y} := [y_1, \ldots, y_n] & \mathbf{z} := [z_1, \ldots, z_n] \\
\mathbf{x} \leftarrow \sum_{i=1}^{k} \left(\mathbf{x}^T\varphi_i\right)\varphi_i & \mathbf{y} \leftarrow \sum_{i=1}^{k} \left(\mathbf{y}^T\varphi_i\right)\varphi_i & \mathbf{z} \leftarrow \sum_{i=1}^{k} \left(\mathbf{z}^T\varphi_i\right)\varphi_i
\end{array}
\tag{27}
$$

4.2 Implementation

I based on the Laplace matrix obtained in Section 3, and the above formula, using scipy sparse to get eigenvectors. Taking advantage of the speed advantage of vectorization, I implemented the reconstruction see code at the end).

4.3 Experiment and discussion

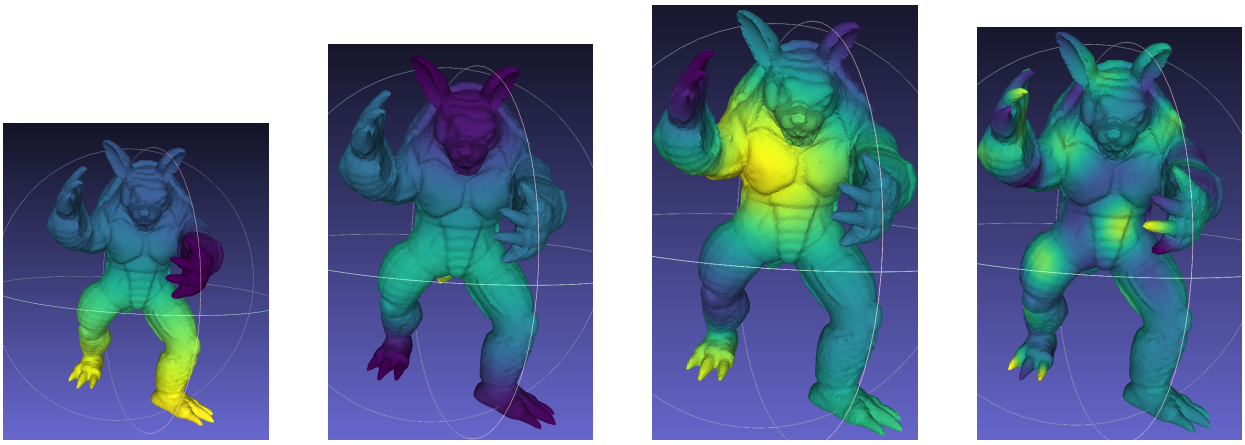
From Figure 9, it can be observed that eigenvectors corresponding to larger absolute eigenvalues contain more high-frequency information.

Using the k eigenvectors corresponding to the smallest absolute eigenvalues, I reconstructed the model. From Figure 10, it can be seen that as k increases, the accuracy of the reconstructed model improves at first, but then the model becomes more and more distorted. Initially, as k increases, the reconstruction effect gets better, which is normal because higher frequency information contains more details and texture of the model. However, when k is higher, the model becomes more and more distorted. The first reason is that the noise is mainly present in the high-frequency information. As k increases, the eigenvectors contain more noise information, which leads to more and more noise in the reconstruction. Another possible reason is that the area estimation in constructing the matrix M is not accurate enough, which results in a more inaccurate estimation of high-frequency information.

5 Explicit Laplacian mesh smoothing

5.1 Method and Implementation

To make a mesh smooth, each node on the mesh can be moved towards the centre of its neighbours. Based on this general idea, an explicit method can be formulated as follows:

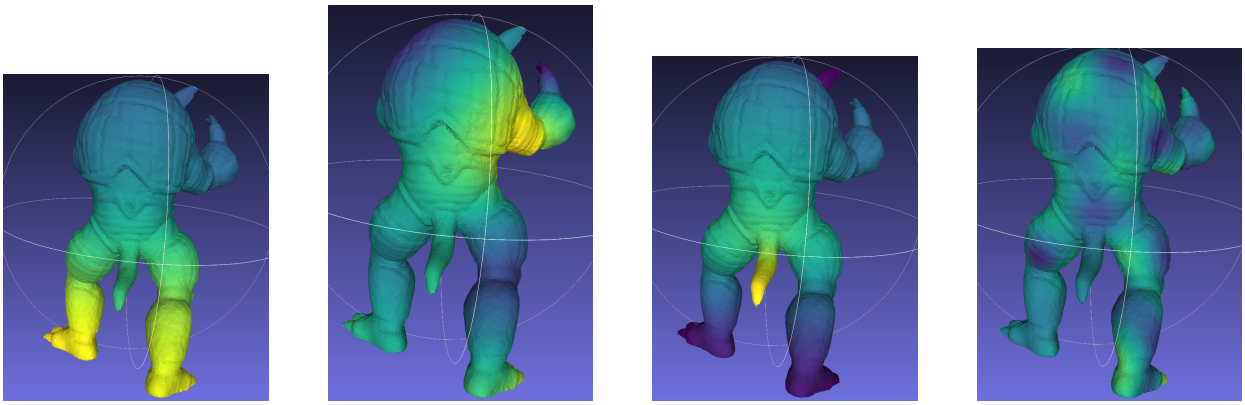


(a) the 2-nd smallest absolute eigenvalue

(b) the 5-th smallest absolute eigenvalue

(c) the 10-th smallest absolute eigenvalue

(d) the 100-th smallest absolute eigenvalue



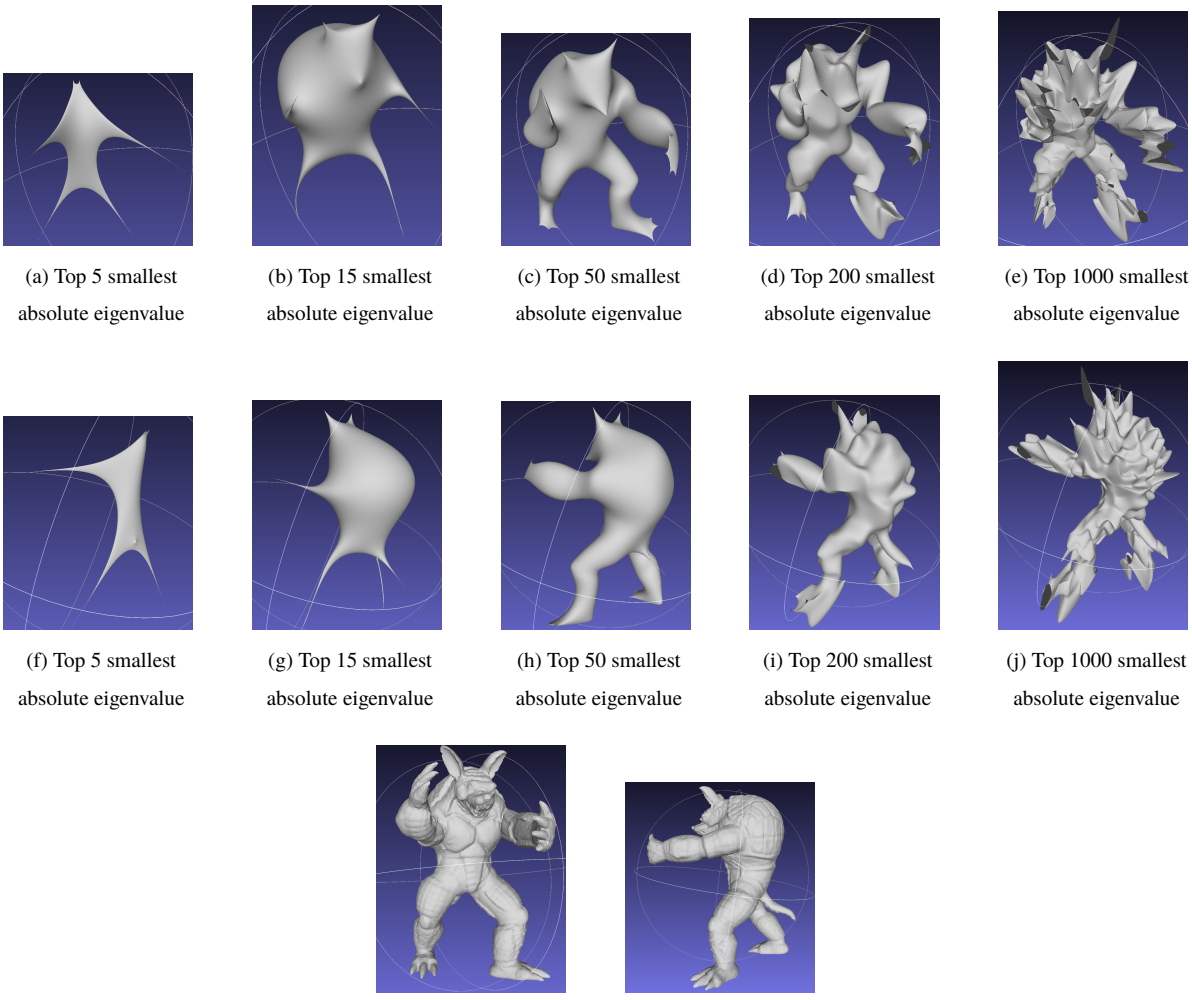
(e) the 2-nd smallest absolute eigenvalue

(f) the 5-th smallest absolute eigenvalue

(g) the 10-th smallest absolute eigenvalue

(h) the 100-th smallest absolute eigenvalue

Figure 9: Eigenvectors visualistion on the mesh. The same object results are shown from different views in the same columns in the figure.



(a) Top 5 smallest absolute eigenvalue

(b) Top 15 smallest absolute eigenvalue

(c) Top 50 smallest absolute eigenvalue

(d) Top 200 smallest absolute eigenvalue

(e) Top 1000 smallest absolute eigenvalue

(f) Top 5 smallest absolute eigenvalue

(g) Top 15 smallest absolute eigenvalue

(h) Top 50 smallest absolute eigenvalue

(i) Top 200 smallest absolute eigenvalue

(j) Top 1000 smallest absolute eigenvalue

(k) Original model

(l) Original model

Figure 10: Reconstruction results. The same object results are shown from different views in the same columns in the figure.

$$\mathbf{P}^{(t+1)} = (\mathbf{I} + \lambda \mathbf{L})\mathbf{P}^{(t)} \tag{28}$$

Based on the results of Sections 1 and 3, it can be observed that the laplacian values obtained using the Discrete Laplace-Beltrami method are more accurate. Therefore, in this section, the L matrix is based on the Discrete Laplace-Beltrami method. If the uniform method is used, there may be vertex drift issues, which can lead to distortion problems, as shown in Figure 11.

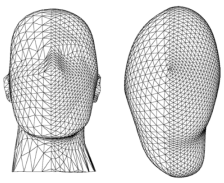


Figure 11: vertex drift problem [3]

5.2 Experiment and discussion

From the Figure 12 Figure 13 below, it can be seen that a good choice of lambda is like the one in figures fgh, which makes the model smoother without causing “huge changes” issues like figures bcd during the iteration process, and does not make the update iteration too slow, as in figures jkl, where changes are not very noticeable within 300 rounds. Because the update of each point in explicit method is local, the problem of "huge changes" when lambda is set to a large value comes from the example shown in Figure o. When lambda is set to a very small value, it can cause very small updates for each iteration. Additionally, as seen from figure mn, when an appropriate lambda value is chosen, the model becomes smoother, but the scale of the model will become smaller as the number of iterations increases.

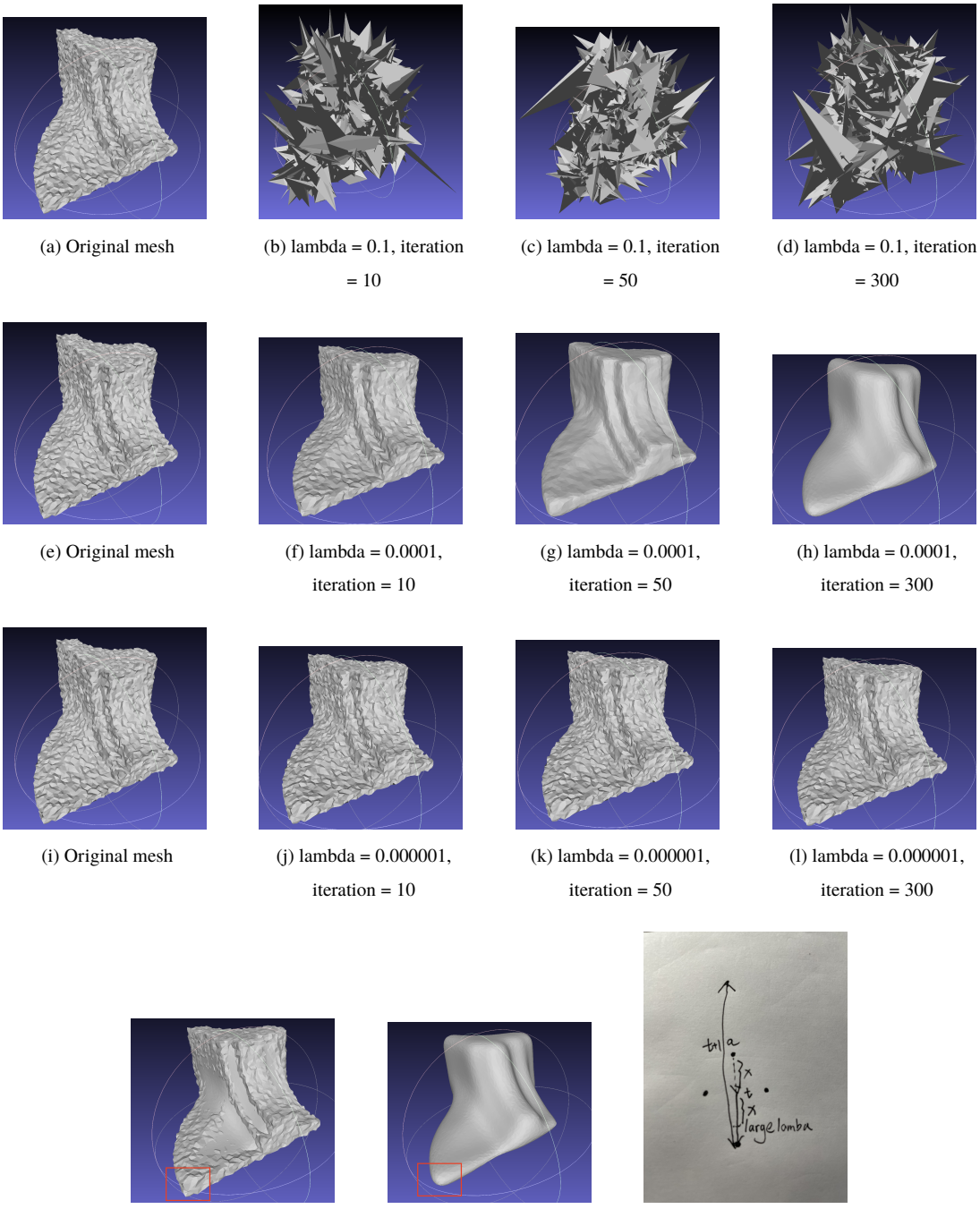
In addition, as can be seen from Figure 14, for non-closed meshes, this method also presents many problems at the boundaries of the mesh. As discussed in Section 3, a solution to avoid this problem is that we can calculate the laplacian value of boundary points by adding some non-existent points based on symmetry. And we also can see for different meshes, we need to choose different lambda values, which is very inconvenient in practical applications.

6 Implicit Laplacian mesh smoothing

6.1 Method and Implementation

Unlike the explicit method that relies on local considerations, the implicit method [3] takes into account global considerations and sets the unknown value LP_{t+1} as the updated value:

$$\begin{aligned} \mathbf{P}^{(t+1)} &= \mathbf{P}^{(t)} + \lambda \mathbf{L}\mathbf{P}^{(t+1)} \\ (\mathbf{I} - \lambda \mathbf{L})\mathbf{P}^{(t+1)} &= \mathbf{P}^{(t)} \end{aligned} \tag{29}$$



(a) Original mesh

(b) lambda = 0.1, iteration = 10

(c) lambda = 0.1, iteration = 50

(d) lambda = 0.1, iteration = 300

(e) Original mesh

(f) lambda = 0.0001, iteration = 10

(g) lambda = 0.0001, iteration = 50

(h) lambda = 0.0001, iteration = 300

(i) Original mesh

(j) lambda = 0.000001, iteration = 10

(k) lambda = 0.000001, iteration = 50

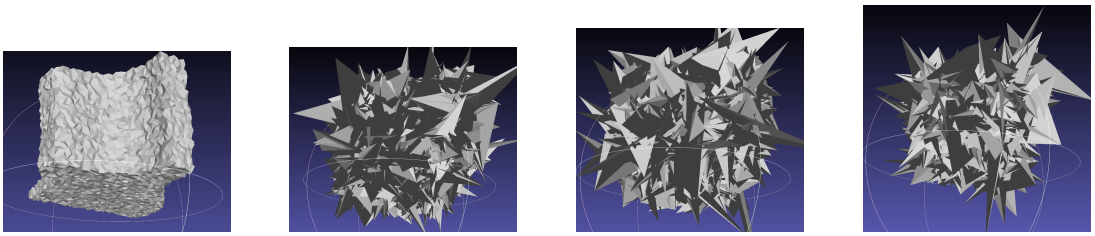
(l) lambda = 0.000001, iteration = 300

(m) lambda = 0.0001, compare iteration = 10 and 300

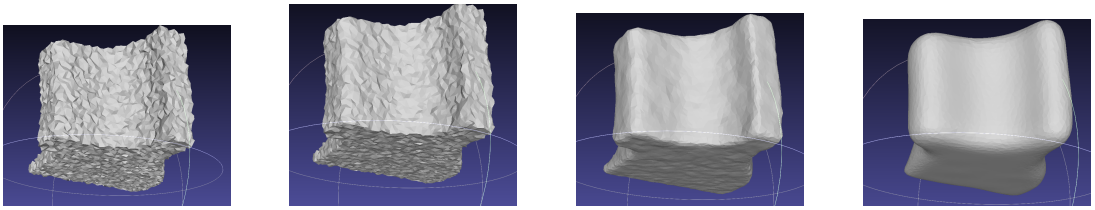
(n) lambda = 0.0001, compare iteration = 10 and 300

(o) large lambda problem example in 2D

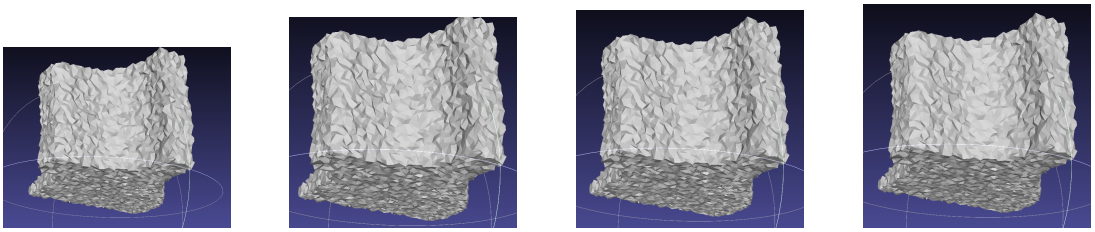
Figure 12: Explicit smooth results with different lambda and iterations



(a) Original mesh (b) lambda = 0.1, iteration = 10 (c) lambda = 0.1, iteration = 50 (d) lambda = 0.1, iteration = 300

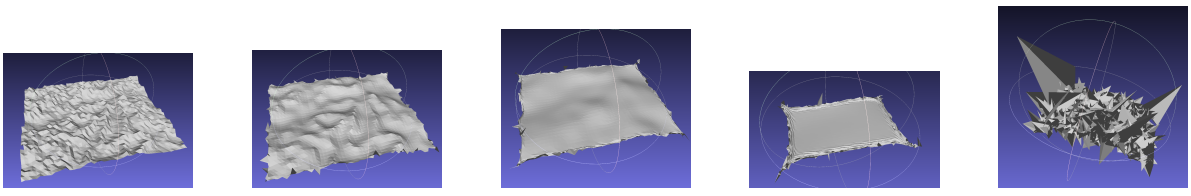


(e) Original mesh (f) lambda = 0.0001, iteration = 10 (g) lambda = 0.0001, iteration = 50 (h) lambda = 0.0001, iteration = 300



(i) Original mesh (j) lambda = 0.000001, iteration = 10 (k) lambda = 0.000001, iteration = 50 (l) lambda = 0.000001, iteration = 300

Figure 13: (different view) Explicit smooth results with different lambda and iterations



(a) Original mesh (b) lambda = 0.000001, iteration = 10 (c) lambda = 0.000001, iteration = 50 (d) lambda = 0.000001, iteration = 300 (e) lambda = 0.0001, iteration = 10

Figure 14: Explicit smooth results with different lambda and iterations

As discussed in the previous section, in this experiment I used the Discrete Laplace-Beltrami method to obtain the Laplacian matrix.

6.2 Experiment and discussion

Advantages:

As shown in Figures Figure 15 and Figure 16, the implicit method still requires choosing an appropriate lambda and iterations to control the smoothness. However, unlike the explicit method discussed in the previous section, the implicit method does not suffer from the "huge changes" issue. Therefore, the implicit method can choose a larger lambda to achieve better results in fewer iterations.

In addition, because the implicit method takes into account global solving, it has better-smoothing results. Especially for non-closed meshes as shown in Figure 17, the boundary of a plane does not produce strange results like the explicit method.

Disadvantages:

1. The implicit method requires solving Ax=b, which can result in invalid values at high iteration times like Figure 15 d.

2. The implicit method still changes the scale of the mesh. Sun et al. [4] found "even we can overcome this problem of shrinkage by rescaling the mesh to preserve its volume. Again, however, distortion of prominent mesh features occurs." This distortion problem can also see Figure 17 f

3. Although the implicit method does not suffer from the "huge changes" issue of the explicit method and can choose a larger lambda for fewer iterations and easier parameter selection, it still requires parameter selection and is not an end-to-end method.

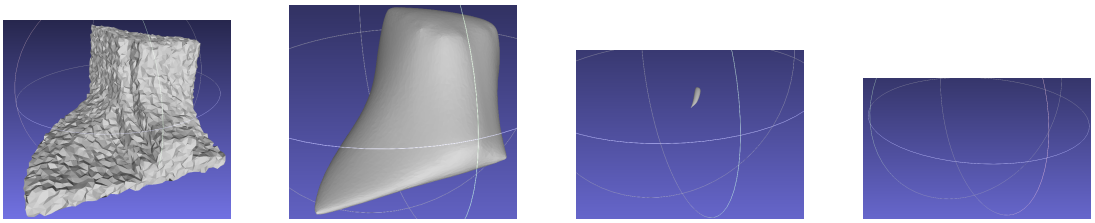
7 Evaluate the performance of Laplacian mesh denoising

7.1 Method and Implementation

In this experiment, I add Gaussian noise to each point of M models, $M' = M + n$, where $n \sim \mathcal{N}(0,\, I \cdot \alpha D)$. α is a hyperparameter in $[0, 0.1]$. I is the Identity matrix with the same shape of M_2. D is the diagonal length of the bounding box of model M_2.

According to the previous experiments, the explicit smooth method usually performs worse than the implicit method and is more difficult to select the lambda parameter, requiring longer iteration time. The method based on spectrum analysis is also not good at reconstructing the texture details of the model Figure 20. Therefore, I chose to use the implicit method from the previous section to experiment with the bunny Figure 18 model at different levels adding Gaussian noise.

Since a large lambda may cause over-smoothing and a small lambda requires more iterations to achieve a smooth effect, the result has monotonicity from visual observation. Therefore, I used binary search to adjust the lambda parameter and finally chose lambda = 0.000005 for the experiment.

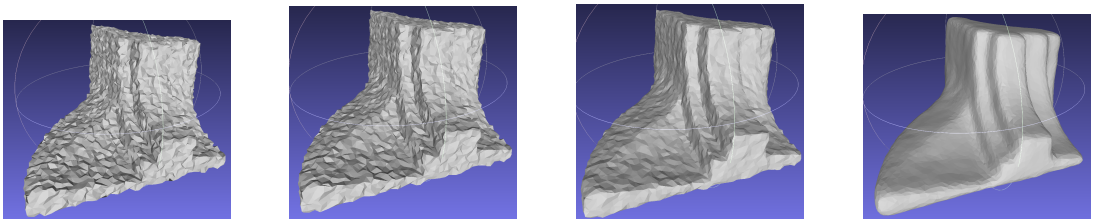


(a) Original mesh

(b) lambda = 1, iteration = 1

(c) lambda = 1, iteration = 3

(d) lambda = 1, iteration = 10, nan results

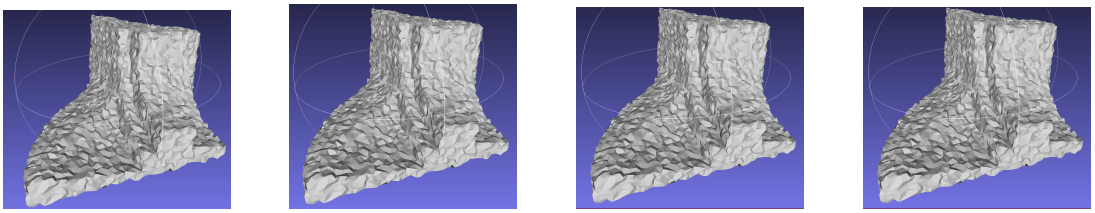


(e) Original mesh

(f) lambda = 0.001, iteration = 1

(g) lambda = 0.001, iteration = 3

(h) lambda = 0.001, iteration = 10



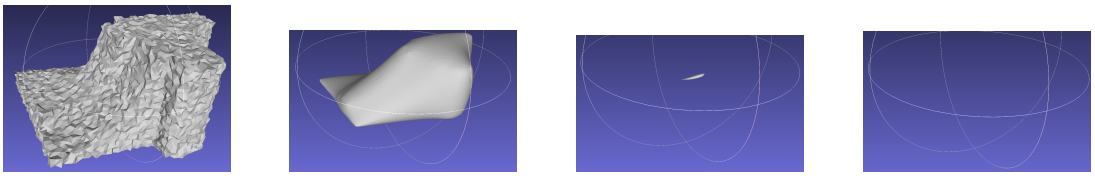
(i) Original mesh

(j) lambda = 0.000001, iteration = 1

(k) lambda = 0.000001, iteration = 3

(l) lambda = 0.000001, iteration = 10

Figure 15: Implicit smooth results with different lambda and iterations

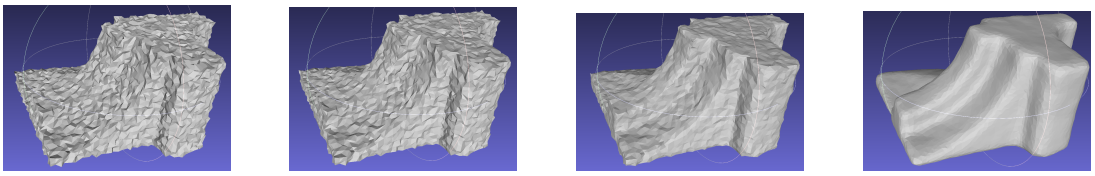


(a) Original mesh

(b) lambda = 1, iteration = 1

(c) lambda = 1, iteration = 3

(d) lambda = 1, iteration = 10, nan results

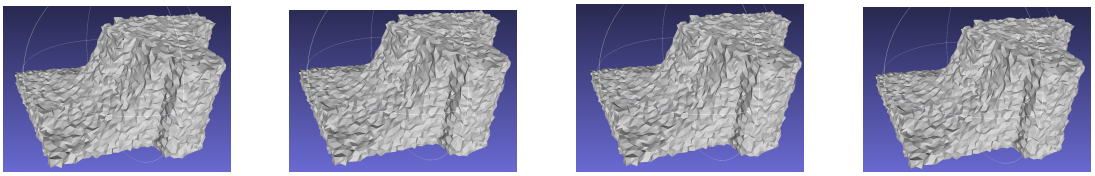


(e) Original mesh

(f) lambda = 0.001, iteration = 1

(g) lambda = 0.001, iteration = 3

(h) lambda = 0.001, iteration = 10



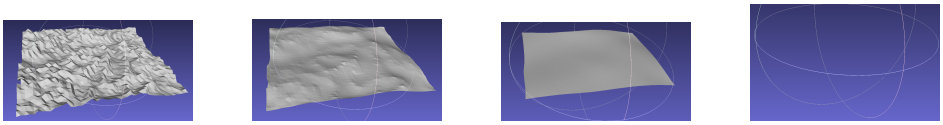
(i) Original mesh

(j) lambda = 0.000001, iteration = 1

(k) lambda = 0.000001, iteration = 3

(l) lambda = 0.000001, iteration = 10

Figure 16: (Different view) Implicit smooth results with different lambda and iterations

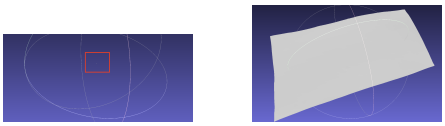


(a) Original mesh

(b) lambda = 0.0001, iteration = 1

(c) lambda = 0.0001, iteration = 3

(d) lambda = 0.0001, iteration = 10, nan reuslts



(e) lambda = 1, iteration = 1, scale problem (same scale with figures abcd)

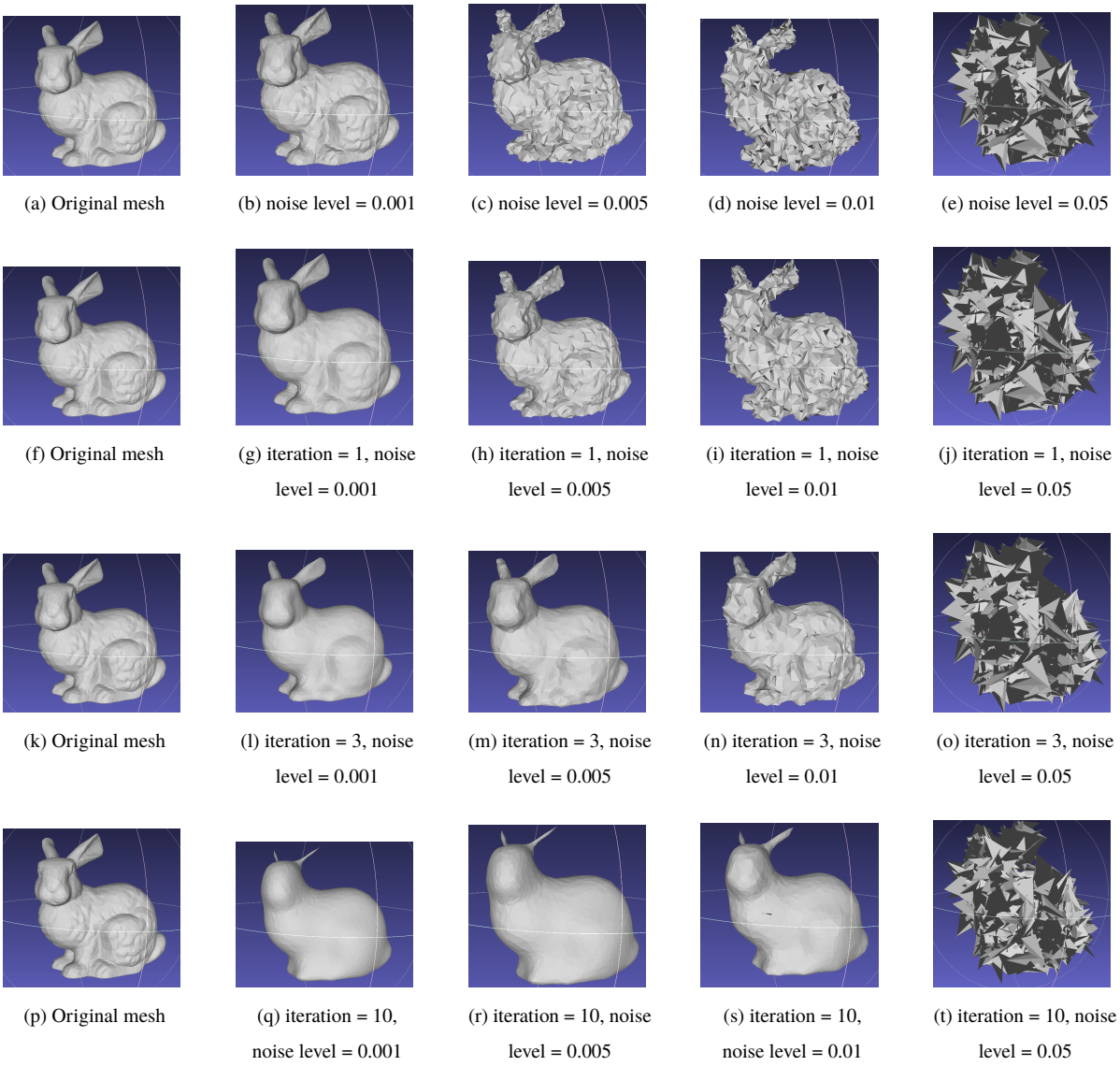
(f) lambda = 1, iteration = 1, zoom up

Figure 17: Explicit smooth results with different lambda and iterations

Figure 18: bunny

7.2 Experiment and discussion

From the Figure 19 below, it can be seen that for small noise, implicit mesh smooth methods achieve good results, but require appropriate iteration selection. As the noise gradually increases, the implicit mesh smooth method can still remove some noise, but the result is not very good. When the noise increases to the point where it affects the main features of the mesh, the implicit mesh smooth method will be unable to remove the noise.



(a) Original mesh (b) noise level = 0.001 (c) noise level = 0.005 (d) noise level = 0.01 (e) noise level = 0.05

(f) Original mesh (g) iteration = 1, noise level = 0.001 (h) iteration = 1, noise level = 0.005 (i) iteration = 1, noise level = 0.01 (j) iteration = 1, noise level = 0.05

(k) Original mesh (l) iteration = 3, noise level = 0.001 (m) iteration = 3, noise level = 0.005 (n) iteration = 3, noise level = 0.01 (o) iteration = 3, noise level = 0.05

(p) Original mesh (q) iteration = 10, noise level = 0.001 (r) iteration = 10, noise level = 0.005 (s) iteration = 10, noise level = 0.01 (t) iteration = 10, noise level = 0.05

Figure 19: Explicit smooth results with different noise level and iterations

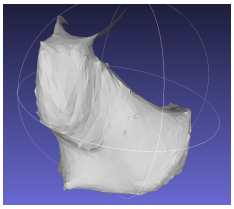


Figure 20: noise level = 0.05, eigenvalue reconstruction

References

[1] eldo. (2014) Comparison with mean curvature. [Online]. Available: https://mathematica.stackexchange.com/questions/61409/computing-gaussian-curvature

[2] L. Ferraz and X. Binefa, "A new non-redundant scale invariant interest point detector." in *VISAPP (1)*, 2009, pp. 277–280.

[3] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 1999, pp. 317–324.

[4] X. Sun, P. L. Rosin, R. Martin, and F. Langbein, "Fast and effective feature-preserving mesh denoising," *IEEE transactions on visualization and computer graphics*, vol. 13, no. 5, pp. 925–938, 2007.

In [18]:
```
import sys,os
RES_PATH = '../resources'
LIB_PATH = '../python_lib'

if not os.path.exists(RES_PATH):
    print( 'cannot find \resources\, please update RES_PATH')
    exit(1)
else:
    print('found resources')

# append path
sys.path.append(LIB_PATH)
import numpy as np
import igl
import trimesh
import open3d
import pyrender
import matplotlib
import matplotlib.pyplot as plt
from geo_tools import rd_helper
from sklearn.neighbors import KDTree
from trimesh import creation, transformations
from mpl_toolkits.mplot3d import proj3d
import scipy.sparse
from scipy.sparse import lil_matrix as lil_matrix
from scipy.sparse import spdiags
import matplotlib as mpl
import matplotlib.cm as cm
```

```
found resources
```

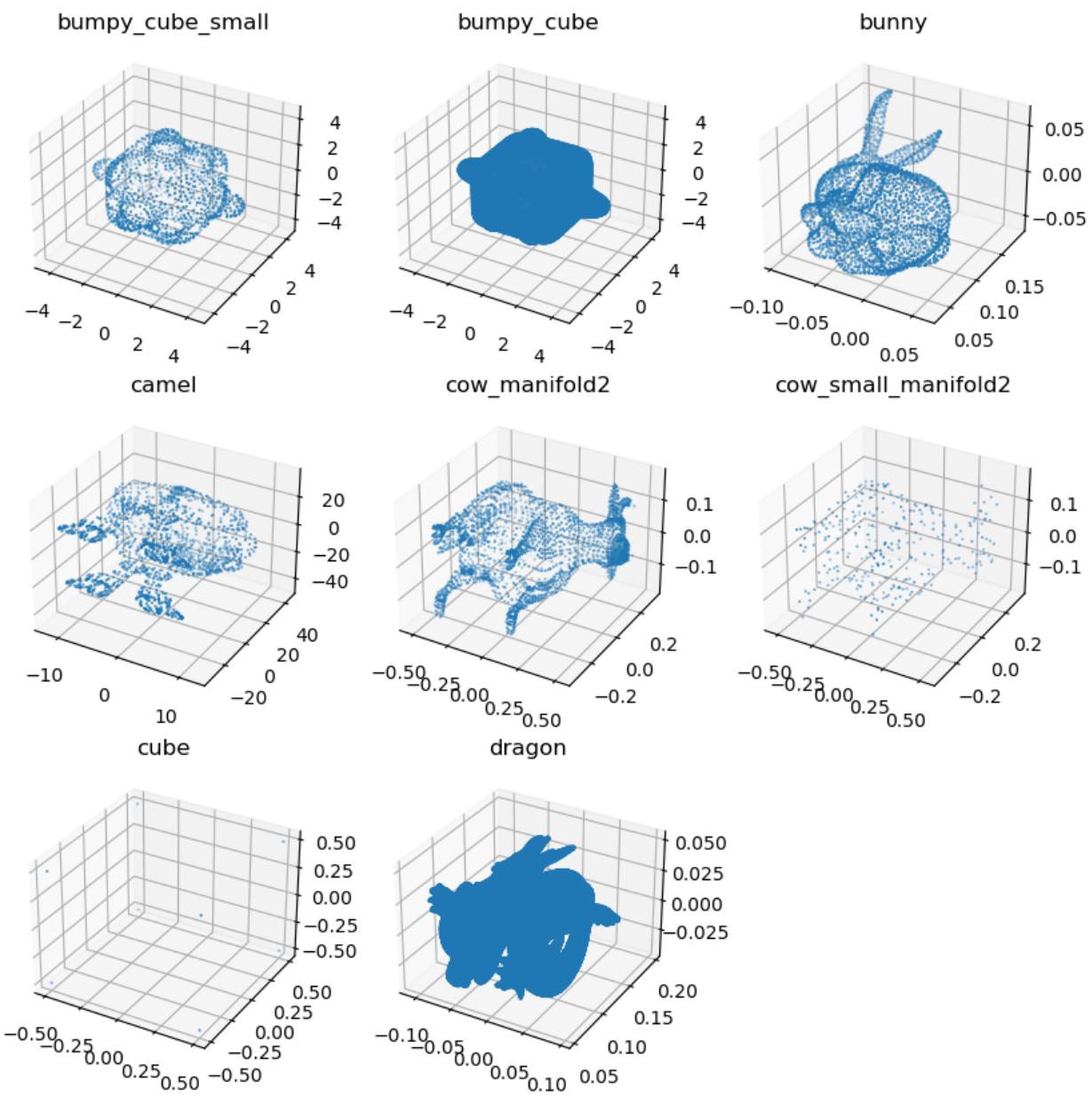
In [207…
```
bumpy_cube_small = trimesh.load('./meshes/bumpy-cube-small.obj')
bumpy_cube =  trimesh.load('./meshes/bumpy-cube.obj')
bunny = trimesh.load('./meshes/bunny.obj')
camel = trimesh.load('./meshes/camel.obj')
cow_manifold2 = trimesh.load('./meshes/cow_manifold2.obj')
cow_small_manifold2 = trimesh.load('./meshes/cow_small_manifold2.obj')
cube = trimesh.load('./meshes/cube.obj')
dragon = trimesh.load('./meshes/dragon.obj')
```

In [208…
```
MODEL_dict = {'bumpy_cube_small' : bumpy_cube_small, 'bumpy_cube' : bumpy_cube, 'bunny': bunny,
              'camel': camel, 'cow_manifold2': cow_manifold2, 'cow_small_manifold2':cow_small_manifold2,
              'cube': cube, 'dragon':dragon}
```

In [21]:
```
print(MODEL_dict.keys())
```

```
dict_keys(['bumpy_cube_small', 'bumpy_cube', 'bunny', 'camel', 'cow_manifold2', 'cow_small_manifold2', 'cube', 'drago
n'])
```

In [22]:
```
fig = plt.figure(figsize = (10.0, 10.0))
N_model = len(MODEL_dict)
for idx, key in enumerate(MODEL_dict):
    model = MODEL_dict[key]
    ax = fig.add_subplot(3, (N_model - 1) // 3 + 1, idx + 1, projection='3d')
    input = model.vertices
    ax.scatter(input[:,0], input[:,1], -input[:,2], marker='.', s=1)
    ax.set_title(key)
```



1. Uniform Laplace

1) mean curvature

In [23]:
```
def build_uniform_Laplace_operator(model):
    N = model.vertices.shape[0]
    Lap_operator = lil_matrix((N, N))
    for idx in range(N):
        neighbors = np.array(model.vertex_neighbors[idx])
        Lap_operator[idx, neighbors] = 1 / neighbors.shape[0]
    I = np.arange(N)
    Lap_operator[I, I] = -1
    return Lap_operator

def get_uniform_disc_lap(model):
    Lapvalue = np.zeros_like(model.vertices)
    for i in range(model.vertices.shape[0]):
        neighbors = np.array(model.vertex_neighbors[i])
        Lapvalue[i] = np.sum(model.vertices[i] - model.vertices[neighbors], axis = 0) / neighbors.shape[0]
    return Lapvalue

def uniform_mean_curvature(model, method = 'operator'):
    if method == 'operator':
        Lap_operator = build_uniform_Laplace_operator(model)
        Lap_value = Lap_operator @ model.vertices
    elif method == 'discretization':
        Lap_value = get_uniform_disc_lap(model)
    Mean_curvature = 0.5 * np.linalg.norm(Lap_value, axis = 1)
    return Mean_curvature
```

In [24]:
```
MODEL = bumpy_cube_small
```

In [25]:
```
Mean_curvature_Lap_operator = uniform_mean_curvature(MODEL, method = 'operator')
Mean_curvature_uniform_discretization = uniform_mean_curvature(MODEL, method = 'discretization')
```

In [26]:
```
print(np.linalg.norm(Mean_curvature_Lap_operator - Mean_curvature_uniform_discretization))
```

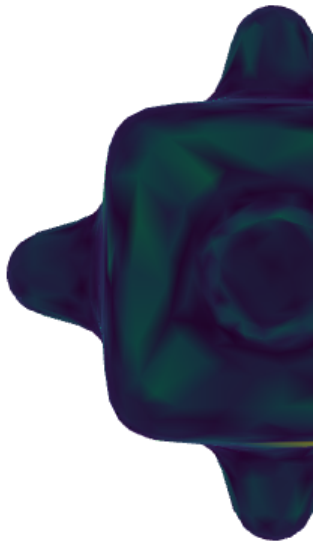
```
3.1619569254907614e-15
```

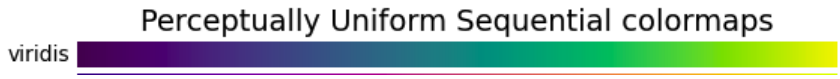
In [187…
```
def visulaize(model, color, file_name = 'tmp.obj', Norm = True, cut_value = None):
    data_color = color.copy()
    if cut_value is not None:
        data_color[data_color > cut_value[1]] = cut_value[1]
        data_color[data_color < cut_value[0]] = cut_value[0]


    vmin, vmax = np.min(data_color), np.max(data_color)
    norm = mpl.colors.Normalize(vmin = vmin, vmax = vmax)
    cmap = cm.viridis
    if Norm == True:
        m = cm.ScalarMappable(norm = norm, cmap=cmap)
    else:
        m = cm.ScalarMappable(cmap=cmap)
    vis_data = m.to_rgba(data_color)
    model.visual.vertex_colors = vis_data
    z = model.export(file_name)
```

In [28]:
```
visulaize(MODEL, Mean_curvature_Lap_operator, file_name = 'Q1_small_cube.obj', cut_value = [-3, 3])
```

In [29]:
```
MODEL.show()
```

Out[29]:





In [30]:
```
print(MODEL_dict.keys())
```

```
dict_keys(['bumpy_cube_small', 'bumpy_cube', 'bunny', 'camel', 'cow_manifold2', 'cow_small_manifold2', 'cube', 'drago
n'])
```

In [17]:
```
for idx, key in enumerate(MODEL_dict):
    MODEL = MODEL_dict[key]
    Mean_curvature = uniform_mean_curvature(MODEL)
    visulaize(MODEL, Mean_curvature, file_name = './Q1/MC_' + key + '.obj', cut_value = [-3, 3])
```

2) Gaussian curvature

In [209…
```
def get_angles(a, b):
    return np.arccos(np.sum(a * b) / np.linalg.norm(a) / np.linalg.norm(b))

def get_othervertices(j):
    if j == 0:
        a, b = 1, 2
    elif j == 1:
        a, b = 0, 2
    else:
        a, b = 0, 1
    return a, b

def get_gaussian_curvature(model):
    # From assignment, we can use third-party libraries to access
    # neighboring vertices/edges/faces/face area is allowed
```

```
    # But without angles. So we need to compute angles.
    face_areas = model.area_faces
    faces = model.faces
    vertices = model.vertices
    N = model.vertices.shape[0]

    vetrices_angles = np.zeros(N)
    vetrices_ares = np.zeros(N)

    for i in range(faces.shape[0]):
        for j in range(3):
            p_index = faces[i, j]
            a, b = get_othervertices(j)
            a, b = faces[i, a], faces[i, b]
            vetrices_angles[p_index] += get_angles(vertices[a, :] - vertices[p_index, :], vertices[b,:] - vertices[p_in
            vetrices_ares[p_index] += face_areas[i]

    Gaussian_curvature = (2 * np.pi - vetrices_angles) / (vetrices_ares / 3.0)

    return Gaussian_curvature
```

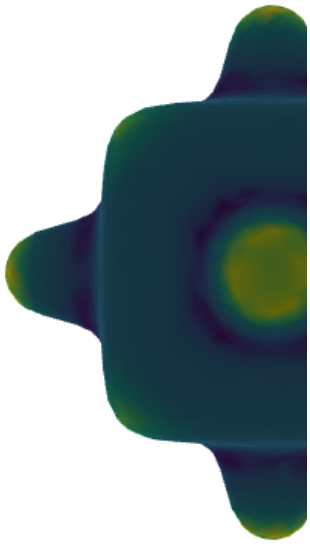
In [210…
```
MODEL = bumpy_cube_small
```

In [211…
```
Gaussian_curvature = get_gaussian_curvature(MODEL)
```

In [212…
```
visulaize(MODEL, Gaussian_curvature, file_name = 'Q1_small_cube.obj', cut_value = [-3, 3])
```

In [213…
```
MODEL.show()
```

Out[213]:



In [214…
```
print(MODEL_dict.keys())
```

```
dict_keys(['bumpy_cube_small', 'bumpy_cube', 'bunny', 'camel', 'cow_manifold2', 'cow_small_manifold2', 'cube', 'drago
n'])
```

In [215…
```
for idx, key in enumerate(MODEL_dict):
    MODEL = MODEL_dict[key]
    gaussian_curvature = get_gaussian_curvature(MODEL)
    visulaize(MODEL, gaussian_curvature, file_name = './Q1/GC_' + key + '.obj', cut_value = [-3, 3])
```

2. First and Second Fundamental forms:

$$p(u,v) = \begin{pmatrix} a\cos(u)\sin(v) \\ b\sin(u)\sin(v) \\ c\cos(v) \end{pmatrix} \tag{1}$$

$$\begin{aligned} X_u &= \frac{\partial p(u,v)}{\partial u} = \begin{pmatrix} -a\sin(v)\sin(u) \\ b\sin(v)\cos(u) \\ 0 \end{pmatrix} \\ X_v &= \frac{\partial p(u,v)}{\partial v} = \begin{pmatrix} a\cos(u)\cos(v) \\ b\sin(u)\cos(v) \\ -c\sin(v) \end{pmatrix} \end{aligned} \tag{2}$$

First fundamental form:

$$\begin{Vmatrix} E & F \\ F & G \end{Vmatrix} = \begin{Vmatrix} X_u^T X_u & X_u^T X_v \\ X_v^T X_u & X_v^T X_v \end{Vmatrix} \tag{3}$$

Where,

$$\begin{aligned} E &= X_u^T X_u = a^2 \sin(u)^2 \sin(v)^2 + b^2 \sin(v)^2 \cos(u)^2 \\ &= (a^2 \sin(u)^2 + b^2 \cos(u)^2) \sin(v)^2 \\ F &= X_u^T X_v = X_v^T X_u = -a^2 \sin(u) \sin(v) \cos(u) \cos(v) + b^2 \sin(u) \sin(v) \cos(u) \cos(v) \\ &= (b^2 - a^2) \sin(u) \sin(v) \cos(u) \cos(v) \\ G &= X_v^T X_v = a^2 \cos(u)^2 \cos(v)^2 + b^2 \sin(u)^2 \cos(v)^2 + c^2 \sin(v)^2 \end{aligned} \tag{4}$$

Second fundamental form:

$$\begin{Vmatrix} e & f \\ f & g \end{Vmatrix} = \begin{Vmatrix} X_{uu}^T n & X_{uv}^T n \\ X_{uv}^T n & X_{vv}^T n \end{Vmatrix} \tag{5}$$

Where,

$$\begin{aligned} X_{uu} = \frac{\partial^2 p(u,v)}{\partial u^2} &= \begin{pmatrix} -a \sin(v) \cos(u) \\ -b \sin(v) \sin(u) \\ 0 \end{pmatrix} \\ X_{vv} = \frac{\partial^2 p(u,v)}{\partial v^2} &= \begin{pmatrix} -a \cos(u) \sin(v) \\ -b \sin(u) \sin(v) \\ -c \cos(v) \end{pmatrix} \\ X_{uv} = \frac{\partial^2 p(u,v)}{\partial u \partial v} &= \begin{pmatrix} -a \cos(v) \sin(u) \\ b \cos(v) \cos(u) \\ 0 \end{pmatrix} \end{aligned} \tag{6}$$

$$n = \frac{X_u \otimes X_v}{||X_u \otimes X_v||} \tag{7}$$

Where $\otimes$ is the cross product.

$$\begin{aligned} X_u \otimes X_v &= \begin{pmatrix} -bc \sin(v)^2 \cos(u) \\ -ac \sin(v)^2 \sin(u) \\ -ab \sin(v) \cos(v) (\sin^2(u) + \cos^2(u)) \end{pmatrix} \\ &= \begin{pmatrix} -bc \sin(v)^2 \cos(u) \\ -ac \sin(v)^2 \sin(u) \\ -ab \sin(v) \cos(v) \end{pmatrix} \end{aligned} \tag{8}$$

So,

$$\begin{aligned} e/||X_u \otimes X_v|| &= abc \sin(v)^3 \cos(u)^2 + abc \sin(v)^3 \sin(u)^2 \\ &= abc \sin(v)^3 \left(\sin(u)^2 + \cos(u)^2\right) \\ &= abc \sin(v)^3 \\ e &= \frac{abc \sin(v)^3}{||X_u \otimes X_v||} \\ f/||X_u \otimes X_v|| &= abc \cos(v) \sin(v)^2 \cos(u) \sin(u) - abc \cos(v) \sin(v)^2 \cos(u) \sin(u) \\ &= 0 \\ f &= 0 \\ g/||X_u \otimes X_v|| &= abc \sin(v)^3 \cos(u)^2 + abc \sin(v)^3 \sin(u)^2 + abc \sin(v) \cos(v)^2 \\ &= abc \sin(v)^3 + abc \sin(v) \cos(v)^2 \\ &= abc \sin(v) (\sin(v)^2 + \cos(v)^2) \\ &= abc \sin(v) \\ g &= \frac{abc \sin(v)}{||X_u \otimes X_v||} \end{aligned} \tag{9}$$

where,

$$||X_u \otimes X_v|| = \sqrt{(-bc \sin(v)^2 \cos(u))^2 + (-ac \sin(v)^2 \sin(u))^2 + (-ab \sin(v) \cos(v))^2} \tag{10}$$

Then we need to calculate the normal curvature function for point $(a, 0, 0)$

Take $(a, 0, 0) = p(u, v)$

$$\begin{cases} a \cos(u) \sin(v) = a \\ b \sin(u) \sin(v) = 0 \\ c \cos(v) = 0 \end{cases}$$

Because $u \in [0, 2\pi), v \in [0, \pi)$

So, $u = 0, v = \frac{\pi}{2}$

$$
\begin{aligned}
E &= (a^2 \sin(u)^2 + b^2 \cos(u)^2) \sin(v)^2 \\
&= b^2 \\
F &= (b^2 - a^2) \sin(u) \sin(v) \cos(u) \cos(v) \\
&= 0 \\
G &= a^2 \cos(u)^2 \cos(v)^2 + b^2 \sin(u)^2 \cos(v)^2 + c^2 \sin(v)^2 \\
&= c^2
\end{aligned} \tag{11}
$$

$$
\begin{aligned}
||X_u \otimes X_v|| &= \sqrt{(-bc \sin(v)^2 \cos(u))^2 + (-ac \sin(v)^2 \sin(u))^2 + (-ab \sin(v) \cos(v))^2} \\
&= b^2 c^2 \\
e &= \frac{abc \sin(v)^3}{||X_u \otimes X_v||} \\
&= \frac{a}{bc} \\
f &= 0 \\
g &= \frac{abc \sin(v)}{||X_u \otimes X_v||} \\
&= \frac{a}{bc}
\end{aligned} \tag{12}
$$

At $(a, 0, 0)$ the normal curvature is:

$$
\begin{aligned}
\kappa_n(\bar{\mathbf{t}}) = \frac{\bar{\mathbf{t}}^T \mathbf{I} \bar{\mathbf{t}}}{\bar{\mathbf{t}}^T \mathbf{II} \bar{\mathbf{t}}} &= \frac{eA^2 + 2fAB + gB^2}{EA^2 + 2FAB + GB^2} \\
&= \frac{\frac{a}{bc}(A^2 + B^2)}{b^2 A^2 + c^2 B^2}
\end{aligned} \tag{13}
$$

$$
\begin{aligned}
\mathbf{t} &= A\mathbf{x}_u + B\mathbf{x}_v \\
\bar{\mathbf{t}} &= (A, B)
\end{aligned}
$$

where AB are our function variables.

3. Non-uniform (Discrete Laplace-Beltrami):

In [156…

```
def get_angles(a, b):
    return np.arccos(np.sum(a * b) / np.linalg.norm(a) / np.linalg.norm(b))

def get_othervertices(j):
    if j == 0:
        a, b = 1, 2
    elif j == 1:
        a, b = 0, 2
    else:
        a, b = 0, 1
    return a, b

def get_Laplace_Beltrami(model):

    face_angles = model.face_angles[:,:]
    faces = model.faces[:,:]
    face_areas = model.area_faces
    vertices = model.vertices
    N = model.vertices.shape[0]
    M = faces.shape[0]

    vetrices_ares = np.zeros(N)
    Cotangent_C = lil_matrix((N, N))
    for i in range(M):
        for j in range(3):
            a, b = get_othervertices(j)
            p_index = faces[i, j]
            a, b = faces[i, a], faces[i, b]
            angle_tmp = get_angles(vertices[a, :] - vertices[p_index, :], vertices[b,:] - vertices[p_index, :])
            angle_tmp2 = face_angles[i, j]
            angle_tmp = 1.0/np.tan(angle_tmp)
            Cotangent_C[a, b] += angle_tmp
            Cotangent_C[b, a] += angle_tmp
            vetrices_ares[p_index] += face_areas[i]

    I = np.arange(N)
    Cotangent_C[I, I]= -1 * np.sum(Cotangent_C, axis = 1)
    vetrices_ares_double = vetrices_ares / 3 * 2
    data = 1 / vetrices_ares_double
    diags = np.array([0])
    Cotangent_M_inv = spdiags(data, diags, N, N)
```

```
    Cotangent_M = spdiags(vetrices_ares_double, diags, N, N)
    Lap_B = Cotangent_M_inv @ Cotangent_C
    return Lap_B, Cotangent_M, Cotangent_M_inv, Cotangent_C
```

In [157…
```
def get_Lapb_mean_cuvature(model):
    Lap_B, _, _, _ = get_Laplace_Beltrami(model)
    Lap_B_value = Lap_B @ model.vertices
    Mean_curvature_Lap_B = 0.5 * np.linalg.norm(Lap_B_value, axis = 1)
    return Mean_curvature_Lap_B
```

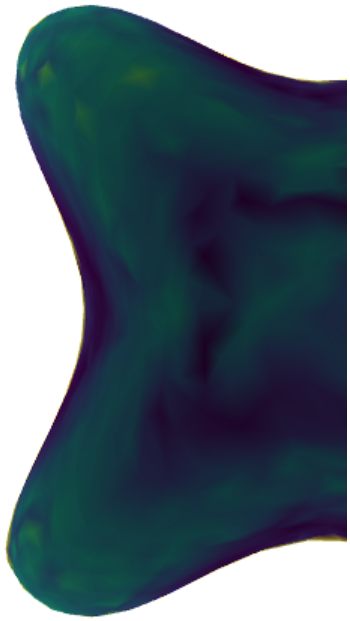
In [158…
```
lilium_s = trimesh.load('./meshes/curvatures/lilium_s.obj')
plane = trimesh.load('./meshes/curvatures/plane.obj')
MODEL_dict['lilium_s'] = lilium_s
MODEL_dict['plane'] = plane
```

In [173…
```
print(lilium_s.vertices.shape, plane.vertices.shape)
```

```
(786, 3) (100, 3)
```

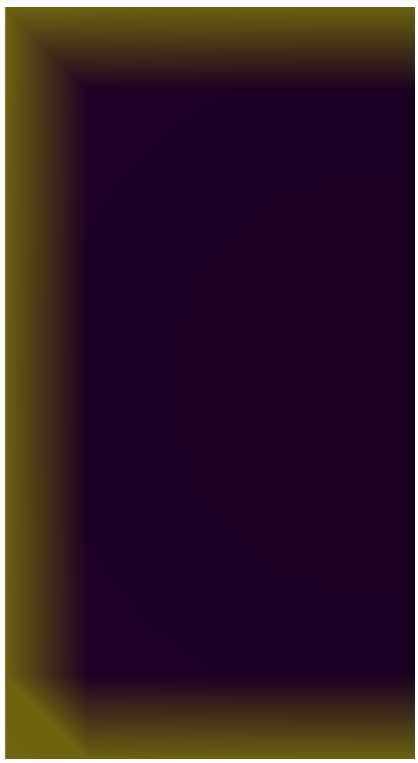
In [159…
```
mean_curvature_lilium_s = get_Lapb_mean_cuvature(lilium_s)
visulaize(lilium_s, mean_curvature_lilium_s, cut_value = [-5, 5])
lilium_s.show()
```

Out[159]:



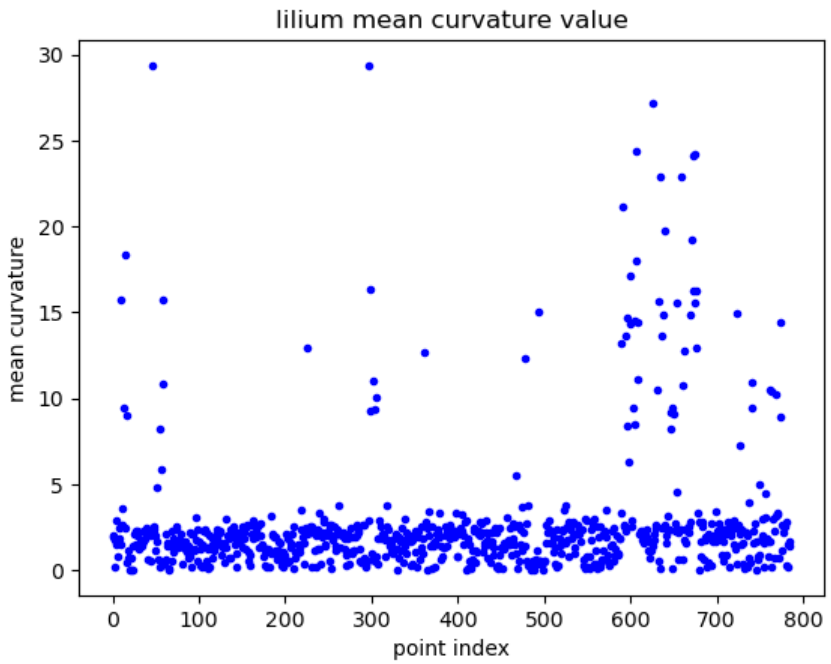
In [160…
```
mean_curvature_plane = get_Lapb_mean_cuvature(plane)
visulaize(plane, mean_curvature_plane, cut_value = [-5, 5])
plane.show()
```

Out[160]:



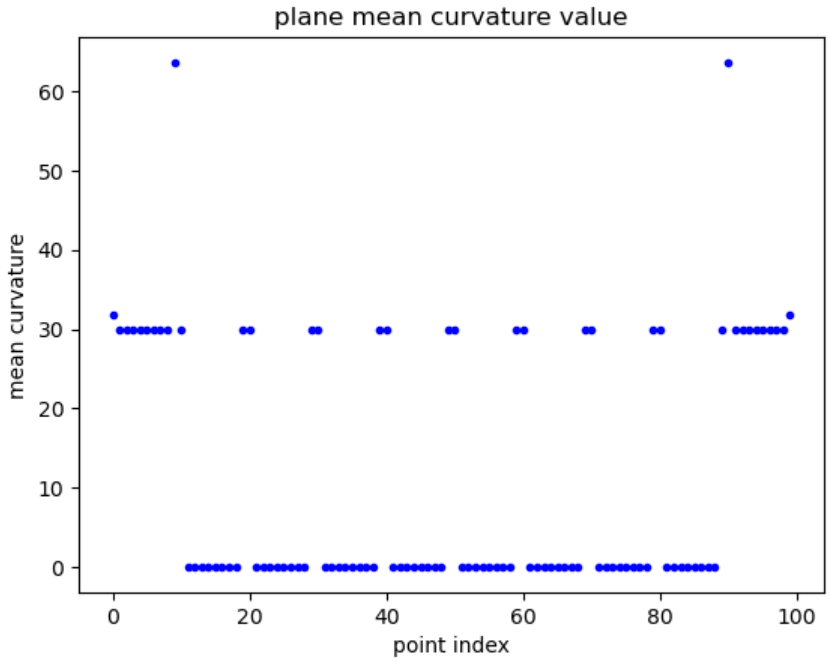
In [161…]
```
plt.plot(mean_curvature_lilium_s, 'b.')
plt.ylabel('mean curvature')
plt.xlabel('point index')
plt.title('lilium mean curvature value')
```

Out[161]:
```
Text(0.5, 1.0, 'lilium mean curvature value')
```



In [162…]
```
plt.plot(mean_curvature_plane, 'b.')
plt.ylabel('mean curvature')
plt.xlabel('point index')
plt.title('plane mean curvature value')
```

Out[162]:
```
Text(0.5, 1.0, 'plane mean curvature value')
```



In [104…]
```
print(MODEL_dict.keys())
```

```
dict_keys(['bumpy_cube_small', 'bumpy_cube', 'bunny', 'camel', 'cow_manifold2', 'cow_small_manifold2', 'cube', 'drago
n', 'lilium_s', 'plane'])
```

In [105…]
```
for idx, key in enumerate(MODEL_dict):
    MODEL = MODEL_dict[key]
    gaussian_curvature = get_Lapb_mean_cuvature(MODEL)
    visulaize(MODEL, gaussian_curvature, file_name = './Q3/' + key + '.obj', cut_value = [-5, 5])
```

4. Modal analysis

In [165…]
```
armadillo = trimesh.load("./meshes/armadillo.obj")
N = armadillo.vertices.shape[0]
print(N)
```

```
25193
```

In [167…
```
from scipy.sparse.linalg import eigs
def get_eigen_vec(model, k):
    Lap_B, Cotangent_M, Cotangent_M_inv, Cotangent_C  = get_Laplace_Beltrami(model)
    Cotangent_M_inv_2 = scipy.sparse.csr_matrix.sqrt(Cotangent_M_inv)
    D = Cotangent_M_inv_2 @ Cotangent_C @ Cotangent_M_inv_2
    eigenvals, eigenvecs = eigs(D, k = k, which = 'SM', ncv = k*2+1)
    eigenvecs = Cotangent_M_inv_2 @ eigenvecs
    return eigenvecs
```

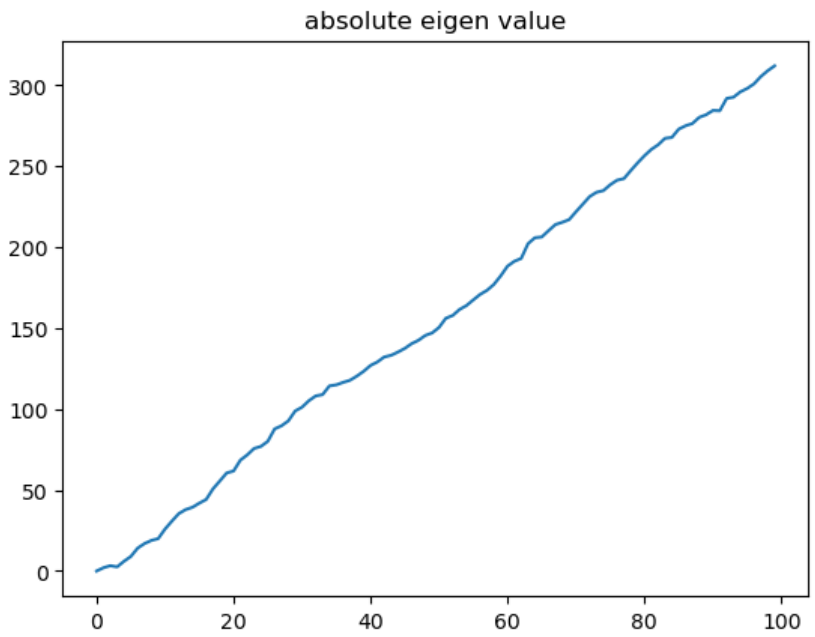
In [168…
```
eigen_list = get_eigen_vec(model = armadillo, k = 1000)
```

In [200…
```
from scipy.sparse.linalg import eigs
def get_eigen_vec2(model, k):
    Lap_B, Cotangent_M, Cotangent_M_inv, Cotangent_C  = get_Laplace_Beltrami(model)
    Cotangent_M_inv_2 = scipy.sparse.csr_matrix.sqrt(Cotangent_M_inv)
    D = Cotangent_M_inv_2 @ Cotangent_C @ Cotangent_M_inv_2
    eigenvals, eigenvecs = eigs(D, k = k, which = 'SM', ncv = k * 2 + 1)
    eigenvecs = Cotangent_M_inv_2 @ eigenvecs
    return eigenvals, eigenvecs
```

In [201…
```
eigenvalues, eigen_list2 = get_eigen_vec2(model = armadillo, k = 100)
```

In [205…
```
plt.plot(np.abs(eigenvalues))
plt.title('absolute eigen value')
```

Out[205]:
```
Text(0.5, 1.0, 'absolute eigen value')
```



In [169…
```
print(eigen_list.shape)
```

```
(25193, 1000)
```

In [216…
```
armadillo_tmp = armadillo.copy()
color = eigen_list.real[:, 100].copy()
visulaize(armadillo_tmp, color)
armadillo_tmp.show()
```

Out[216]:

In [206…
```
single_klist = [1, 2, 3, 4, 5, 10, 100]
for k in single_klist:
    key = str(k)
    color = eigen_list.real[:, k].copy()
    visulaize(armadillo_tmp, color, file_name = './Q4/single_' + key + '.obj')
```

In [174…
```
def reconstruction(model, eigen_vecs, k):
    vertices = model.vertices
    res_model = model.copy()
    M = eigen_vecs.shape[1]

    res_vertices = np.zeros_like(vertices)
    for i in range(k):
        res_vertices[:, 0] += vertices[:, 0] @ eigen_vecs[:, i] * eigen_vecs[:, i]
        res_vertices[:, 1] += vertices[:, 1] @ eigen_vecs[:, i] * eigen_vecs[:, i]
        res_vertices[:, 2] += vertices[:, 2] @ eigen_vecs[:, i] * eigen_vecs[:, i]
    res_model.vertices = res_vertices
    return res_model
```

In [175…
```
K_list = [5, 15, 50, 100, 200, 300, 500, 1000]
```

In [176…
```
for k in K_list:
    recon_ = reconstruction(armadillo, eigen_list.real, k = k)
    z = recon_.export('./Q4/' + str(k) + '.obj')
```

5. Explicit Laplacian mesh smoothing

In [217…
```
fandisk_ns = trimesh.load("./meshes/smoothing/fandisk_ns.obj")
plane_ns = trimesh.load("./meshes/smoothing/plane_ns.obj")
```

In [221…
```
def explicit_smoothing(model, LAMBDA, max_iter = 100):
    smooth_model = model.copy()
    for i in range(max_iter):
        pre_vertices = smooth_model.vertices
        Lap_B, _, __, _ = get_Laplace_Beltrami(smooth_model)
        smooth_model.vertices = pre_vertices + LAMBDA * Lap_B @ pre_vertices
    return smooth_model
```

In [224…
```
LAMBDA_list = [0.1, 0.0001, 0.000001]
max_iter_list = [10, 50, 300]
```

In [225…
```
for idx0, LAMBDA in enumerate(LAMBDA_list):
    for idx1, max_iter in enumerate(max_iter_list):
        smooth_fandisk_ns = explicit_smoothing(fandisk_ns, LAMBDA, max_iter = max_iter)
        key = 'fan_' + str(idx0) + str(idx1)
        z = smooth_fandisk_ns.export('./Q5/' + key + '.obj')
```

In [226…
```
LAMBDA_list = [0.1, 0.0001, 0.000001]
max_iter_list = [10, 50, 300]
```

In [227…
```
for idx0, LAMBDA in enumerate(LAMBDA_list):
    for idx1, max_iter in enumerate(max_iter_list):
        smooth_plane_ns = explicit_smoothing(plane_ns, LAMBDA, max_iter = max_iter)
```

```
        key = 'plane_' + str(idx0) + str(idx1)
        z = smooth_plane_ns.export('./Q5/' + key + '.obj')
```

6. Implicit Laplacian mesh smoothing

In [239…
```
def Implicit_smoothing(model, LAMBDA, max_iter = 10):
    smooth_model = model.copy()
    for i in range(max_iter):
        pre_vertices = smooth_model.vertices
        Lap_B, Cotangent_M, Cotangent_M_inv, Cotangent_C  = get_Laplace_Beltrami(smooth_model)
        A = Cotangent_M - LAMBDA * Cotangent_C
        b = Cotangent_M @ pre_vertices
        smooth_model.vertices = scipy.sparse.linalg.spsolve(A, b)
    return smooth_model
```

In [243…
```
LAMBDA_list = [1, 0.001, 0.000001]
max_iter_list = [1, 3, 10]
```

In [244…
```
for idx0, LAMBDA in enumerate(LAMBDA_list):
    for idx1, max_iter in enumerate(max_iter_list):
        smooth_fandisk_ns = Implicit_smoothing(fandisk_ns, LAMBDA, max_iter = max_iter)
        key = 'fan_' + str(idx0) + str(idx1)
        z = smooth_fandisk_ns.export('./Q6/' + key + '.obj')
```

```
/var/folders/w4/rzpqz_713433wplp923_s5cr0000gn/T/ipykernel_18083/2610599413.py:31: RuntimeWarning: divide by zero encou
ntered in double_scalars
  angle_tmp = 1.0/np.tan(angle_tmp)
/var/folders/w4/rzpqz_713433wplp923_s5cr0000gn/T/ipykernel_18083/3363808345.py:2: RuntimeWarning: invalid value encount
ered in arccos
  return np.arccos(np.sum(a * b) / np.linalg.norm(a) / np.linalg.norm(b))
```

In [245…
```
LAMBDA_list = [1, 0.0001, 0.00000001]
max_iter_list = [1, 3, 10]
```

In [246…
```
for idx0, LAMBDA in enumerate(LAMBDA_list):
    for idx1, max_iter in enumerate(max_iter_list):
        smooth_plane_ns = Implicit_smoothing(plane_ns, LAMBDA, max_iter = max_iter)
        key = 'plane_' + str(idx0) + str(idx1)
        z = smooth_plane_ns.export('./Q6/' + key + '.obj')
```

```
/var/folders/w4/rzpqz_713433wplp923_s5cr0000gn/T/ipykernel_18083/3363808345.py:2: RuntimeWarning: invalid value encount
ered in double_scalars
  return np.arccos(np.sum(a * b) / np.linalg.norm(a) / np.linalg.norm(b))
/var/folders/w4/rzpqz_713433wplp923_s5cr0000gn/T/ipykernel_18083/2610599413.py:31: RuntimeWarning: divide by zero encou
ntered in double_scalars
  angle_tmp = 1.0/np.tan(angle_tmp)
/var/folders/w4/rzpqz_713433wplp923_s5cr0000gn/T/ipykernel_18083/3363808345.py:2: RuntimeWarning: invalid value encount
ered in arccos
  return np.arccos(np.sum(a * b) / np.linalg.norm(a) / np.linalg.norm(b))
/var/folders/w4/rzpqz_713433wplp923_s5cr0000gn/T/ipykernel_18083/2610599413.py:39: RuntimeWarning: divide by zero encou
ntered in divide
  data = 1 / vetrices_ares_double
```

7. Evaluate the performance of Laplacian meshd enoising

In [249…
```
bunny = trimesh.load('./meshes/bunny.obj')
bunny.show()
```

Out[249]:

In [250…
```python
def add_gaussian_noise(model, noise_rate):
    vertices = model.vertices.copy()
    noise_model = model.copy()
    x_len = np.amax(vertices[:,0]) - np.amin(vertices[:,0])
    y_len = np.amax(vertices[:,1]) - np.amin(vertices[:,1])
    z_len = np.amax(vertices[:,2]) - np.amin(vertices[:,2])
    diag_len_bounding_box = np.sqrt(x_len * x_len + y_len * y_len + z_len * z_len)
    # Add noise
    std = noise_rate * diag_len_bounding_box
    vertices = vertices + np.random.normal(0, std, vertices.shape)
    noise_model.vertices = vertices
    return noise_model
```

In [254…
```python
noise_rate = 0.005
noise_bunny = add_gaussian_noise(bunny, noise_rate)
noise_bunny.show()
```

Out[254]:

In [276…
```python
noise_rate_list = [0.001, 0.005, 0.01, 0.05, 0.1]
LAMBDA = 0.000005
max_iter_list = [1, 3, 10]
```

In [277…
```python
for idx0, noise_rate in enumerate(noise_rate_list):
    noise_bunny = add_gaussian_noise(bunny, noise_rate)
    key = 'noise_' + str(idx0)
    z = noise_bunny.export('./Q7/' + key + '.obj')
    for idx1, max_iter in enumerate(max_iter_list):
```

```
        smooth_bunny = Implicit_smoothing(noise_bunny, LAMBDA, max_iter = max_iter)
        key = 'bunny_' + str(idx0) + str(idx1)
        z = smooth_bunny.export('./Q7/' + key + '.obj')
```

In [278…
```
noise_rate = 0.05
noise_bunny = add_gaussian_noise(bunny, noise_rate)
```

In [279…
```
eigen_list = get_eigen_vec(model = noise_bunny, k = 30)
```

In [280…
```
K_list = [5, 15, 30]
```

In [281…
```
for k in K_list:
    recon_ = reconstruction(noise_bunny, eigen_list.real, k = k)
    z = recon_.export('./Q7/' + str(k) + '.obj')
```