COMP0119 : Acquisition and processing of 3d geometry

Oluwatosin Alabi

February 2020

General information

Question attempted/completed

I completed Questions 1, 2, 3, 4, 5, and 6.

libraries used

Scipy, Numpy, pyrender, trimesh, scipy.spatial.cKDtree, matplotlib, scipy.sparse.linalg

Solutions

Question 1

To get the mean curvature of the mesh using uniform discretization, I first calculated the Laplace Beltrami operator (Δ) which is a vxv matrix (where v is the number of vertices in the mesh). For each vertice in the uniform Laplace Beltrami operator, I stored a "1" corresponding to its self and for each one-ring neighbour, I stored "$\frac{1}{N}$" (where N is the amount of one-ring neighbours). Next, I calculated the absolute value of the mean curvature (H) with equ. 1

$$H = \frac{\|\Delta x\|}{2} \tag{1}$$

Finally, I calculated the sign of the mean curvature by performing the dot product of H*n and n for each vertex. If the dot product was positive for the vertex, then the mean curvature is positive, but if the dot product was negative, the mean curvature would have a negative sign.
To calculate the gauss curvature, I first got the angle deficit of each vertex, then I calculated the area of the one ring neighbour around the vertex. I used the **barycentric area**. Then to get the gauss curvature (K), I used

$$K = \frac{angle\_deficit}{A} \text{ for all vertices}$$

Figure 1 shows the mean curvature

Question 2

$$\Delta_s f(v) = \frac{1}{2A(v)} \sum_{v_i \in N_i(v)} (cot\alpha_i + cot\beta_i)(f(v_i) - f(v)) \tag{2}$$

For this question, I used equ.2.

1. I first calculated the barycentric area of a vertex,
2. then for each one ring neigbour vertex, I calculated the "adjacent angles" (usually there are two adjacent angles except at boundary vertices, where there might only be one),
3. I took the cotan of the adjacent angles, and multiplied by the subtraction of the position of the one-ring-neighbour-vertex and the vertex been considered, then divided the product by 2*Area as shown in equ.2
4. Took the sum of the previous step over all the one ring neigbours

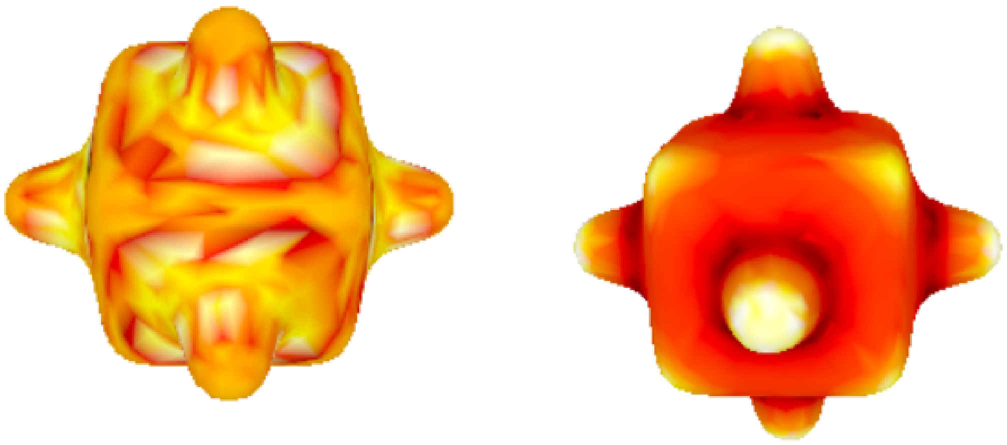

Figure 1: Mean curvature(left), Gauss curvature(right)

5. repeat step 1 to 4 for all vertices in the mesh

Next, I calculated the absolute value of the mean curvature (H) with equ. 1. Finally, I calculated the sign of the mean curvature by performing the dot product of H*n and n for each vertex. If the dot product was positive for the vertex, then the mean curvature is positive, but if the dot product was negative, the mean curvature would have a negative sign.
Comparing the mean curvature as estimated by cotan discretization to that estimated by the uniform discretization, the cotan discretization does a better job estimating the mean curvature.

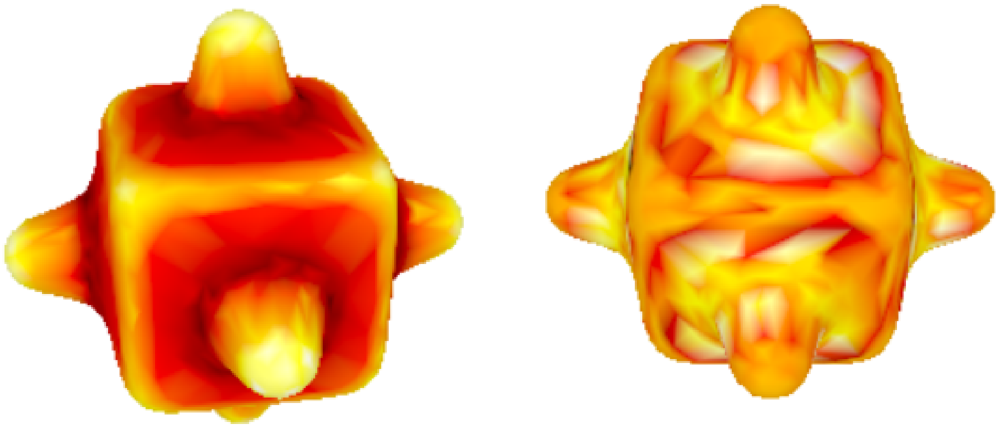

Figure 2: cotan discretization mean curvature(left), uniform discretization mean curvature(right)

Question 3

To compute the spectral analysis for k = 5,10,30 I followed the following steps

1. I first got the mass Matrix (M) and the C matrix using Equ. 3, 4 and 5

$$M = \begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_n \end{bmatrix} \tag{3}$$

2.
$$C_{ij} = \frac{1}{2}(cot\alpha_{ij} + cot\beta_{ij}) \tag{4}$$
$$C_{ii} = -\sum_{j \in N_1(i)} C_{ij} \tag{5}$$

3. obtain the smallest k eigen vectors(y_i) of $M^{-0.5}CM^{-0.5}$ (this is a sparse symmetric form hence we can use scipy.sparse.linalg.eigsh)
4. Get the original eigen vectors for Δ using the formula $\phi_i = M^{-0.5}y_i$
5. perform inner product between the positions of the mesh and eigen vectors gotten (ϕ_i) in the form of $\mathbf{X}M\phi$. **The results for k = 5,10,30 are shown on fig 3(a), 3(b), 3(c)**

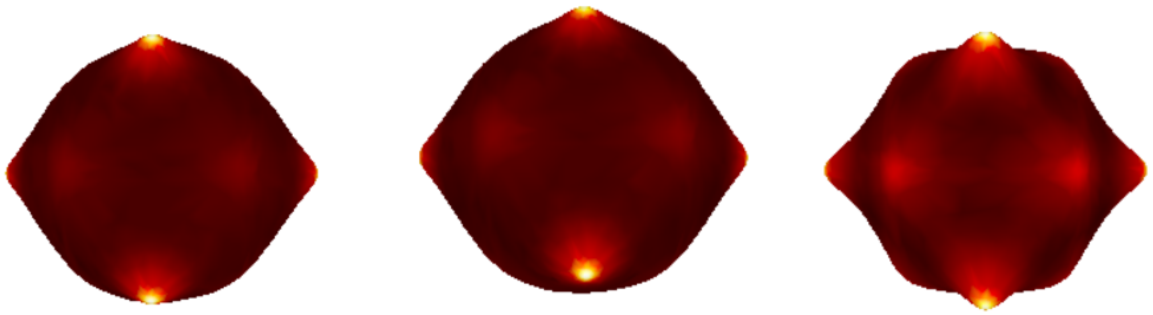

Figure 3: spectral analysis k = 5(left), K = 10 (middle), K= 30 (right)

Question 4

For both To implement explicit laplacian smoothening,

1. I first calculated the matrix form of the cotangent laplace beltrami L
2. Then I used equ. 6 to perform explicit smoothening on the mesh

$$P^{(t+1)} = (U + \lambda L)P^{(t)} \tag{6}$$

3. I performed the smoothening for a certain number of iterations

To compare between different types of smoothening, as well as to see the effect of lamda on the smoothening operation, I kept the amount of iterations constant(number of iterations=5)

As we can see from figure. , explicit smoothening is able to perform smoothening, but it does not totally remove all the noise present in this case. After tunning for a while, at 5 iterations the best lamda I got was $\lambda = 0.11e - 5$ but I noticed that for 5 iterations, above $= 0.5e - 5$, explicit was instead detrimental to the mesh.

Question 5

To perform implicit smoothening,

1. I first calculated the the mass matrix M and C
2. I calculated A = (M - lamda*C), and B = MP, which reduced, the equation below to

$$(M - \lambda C)P^{t+1} = MPt \qquad Ax = B$$

where x is the new mesh

3. then I used the conjugate gradient method from scipy.sparse.linalg.cg to solve for P^{t+1}

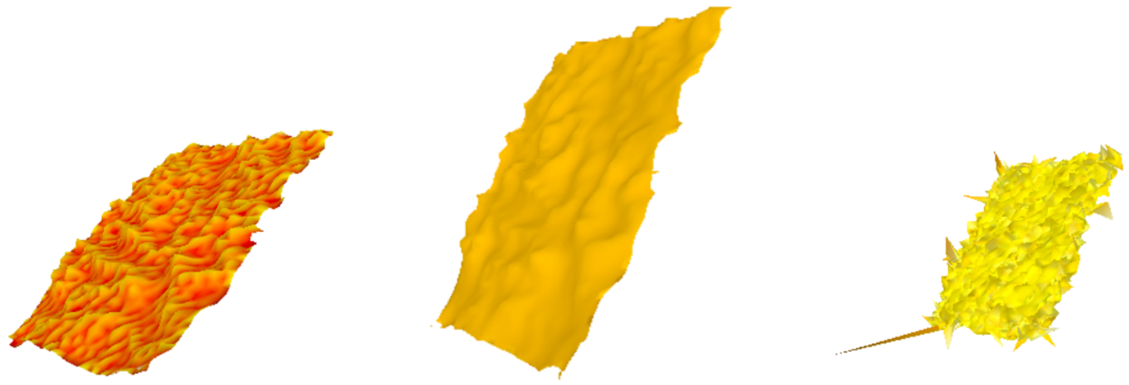

Figure 4: original noisy image (left), lamda = 0.11e-5 (middle), lamda= 2e-5 (right)

I tried various values for lamda I noticed that if lamda is too low (in the order of exp-6,) it required a lot more iteration for the mesh to be completely flattened(about 17 or more iterations). If lamda was just right (around exp-5), the implicit method removes noise perfectly and gives a flat mesh. if I set the value too high (around exp-3) I get a shrinked mesh.

Compared to the previous method explicit smoothening, the implicit smoothening method, is much more stable and the choice of lambda that gives a good result is more flexible. It also removes noise entirely for a range of lamda values, meanwhile, the explicit method was not able to remove noise entirely. The original mesh is 4(a), the smoothened mesh for various values of lambda are shown in fig. 5(a),5(b) and 5(c)

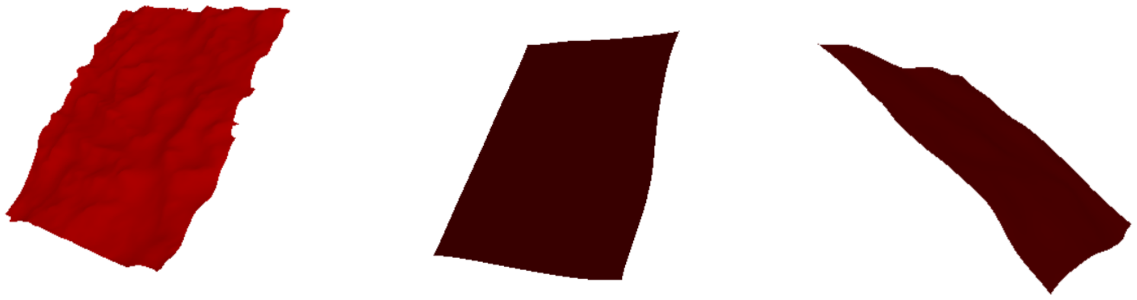

Figure 5: lamda = 1e-6 (left), lamda = 5e-5 (middle), lamda= 1e-5 (right)

Question 6

To test the effect of various noise levels on Laplacian smoothening, I added various levels of gaussian noise of mean zero to the mesh. I divided them into three levels, low levels which had a relatively low amount of noise, medium level which had medium mounts of noise and finally extreme level which totally distorted the shape of the mesh.

To do this, I used an array of random numbers that had a mean of zero and standard deviation equal to a noise level set by me. The shape of this array of values were the same as the shape of the mesh vertices. I then defined a constant k which I used, together with the bounding box dimensions to scale the noise to a value that can be added to the Mesh Vertices. The bounding box dimensions were gotten using trimesh.mesh.extents. Finally, I used implicit smoothening on the mesh, to check if I could eliminate the noise added.

$$boundedNoise = noise * (bounding_box_dimensions/K)$$

My results were as follows:

1. For the extreme case in which the noise was too much, laplacian smoothing was unable to recover back the mesh as shown in figure 8(b) and **??**
2. For medium levels of noise it was possible to get back the mesh with very little or no error 6(b) and **??**, 7(a) and 7(b)

From this we can conclude that laplacian smoothening is a relatively good method of smoothening. As it can handle to a large extent noise in a 3d scan

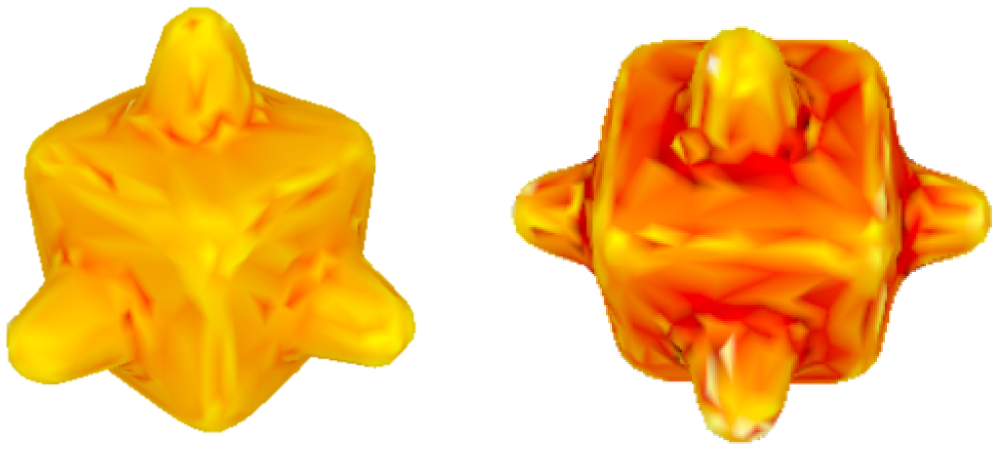
Figure 6: small level noisy mesh (left), smoothened mesh (right)

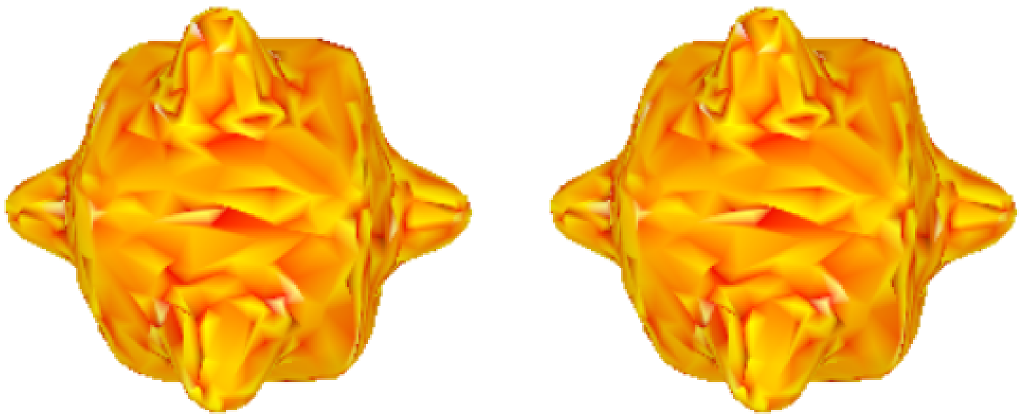
Figure 7: medium level noisy mesh (left), smoothened mesh (right)

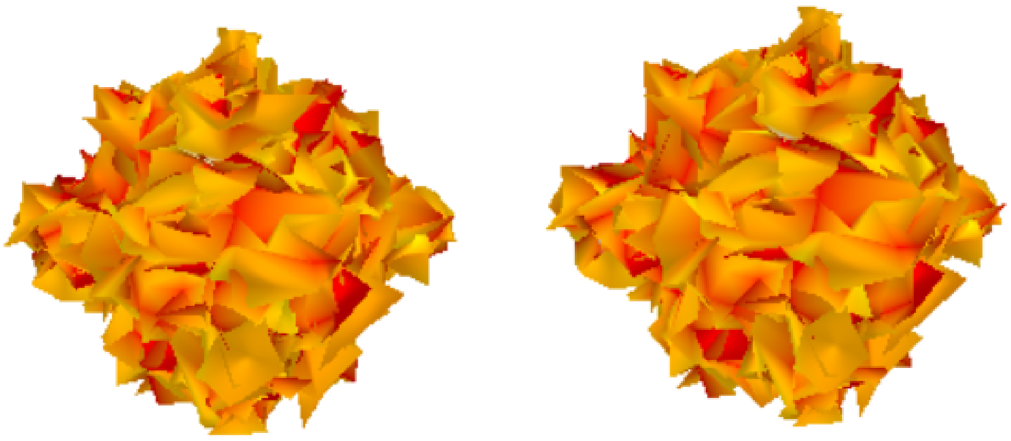

Figure 8: extreme level noisy mesh (left), smoothened mesh (right)