Digital 3D Geometry Processing
Exercise 4 – Surface Normals, Curvature

February 15, 2024

Note

Hand in a .zip compressed file renamed to `Exercise`*n*`-GroupMemberNames.zip` where *n* is the number of the current exercise sheet. It should contain:

- **Only** the files you changed (headers and source). It is up to you to make sure that all files that you have changed are in the zip.
- A `readme.txt` file containing a description on how you solved each exercise (use the same numbers and titles) and the encountered problems.
- Other files that are required by your `readme.txt` file. For example, if you mention some screenshot images in `readme.txt`, these images need to be submitted too.
- Submit your solutions to Gradescope before the submission deadline.

Theory Question (2 pts)

Recall our Gaussian curvature discretization $K(v) = 2\pi - \sum_j \theta_j$ **(Note: without area normalization)** for interior vertices $v \in V_{int}$ of a traingle mesh $M = (V, E, F)$. Let us also define a boundary geodesic curvature discretization at a boundary vertex $b \in V_{bdy}$:

$$k_g(b) = \pi - \sum_j \theta_j, \tag{1}$$

where the second term again gives the interior angle sum over neighboring triangles. When considered together, there is a simple discrete Gauss-Bonnet formula, Eq. (2), that holds for any triangle mesh $M = (V = V_{int} \sqcup V_{bdy}, E, F)$:

$$\sum_{v \in V_{int}} K(v) + \sum_{b \in V_{bdy}} k_g(b) = 2\pi\chi(M) = 2\pi(2 - 2g - n) \tag{2}$$

Show explicitly, by calculating and summing Gaussian curvatures at vertices, that this holds for the following two cases:

a. The regular tetrahedron that is the surface of the convex hull of:

$$\{(0,0,0),(1,1,0),(1,0,1),(0,1,1)\}.$$

b. The surface of the unit cube: $\{(x,y,z) \in \mathbb{R}^3 \mid 0 \leq x,y,z \leq 1\}$ **minus** the bottom face $\{(x,y,z) \in \mathbb{R}^3 \mid 0 \leq x,y \leq 1, z = 0\}$.

For the last example, feel free to choose any triangulation.

Extra Credit Question (+2 pts)

Prove in the case of triangle meshes without boundary ($n = 0$) that the Gauss-Bonnet formula, Eq. (2), is implied by Euler's Polyhedron Formula, Eq. (3), below:

$$|V| - |E| + |F| = \chi(M) = 2 - 2g. \tag{3}$$

(Hint: use some mesh combinatorics, and remember the angle sum of a triangle is π).

Extra Extra Credit Question (+1 pt)

Prove in the case of triangle meshes **with** boundary ($n > 0$) that the Gauss-Bonnet formula, Eq. (2), is implied by Euler's Polyhedron Formula, Eq. (3), below:

$$|V| - |E| + |F| = \chi(M) = 2 - 2g - n. \tag{4}$$

(Hint: same basic idea as above, but the combinatorics get messier and trickier).

Coding Part

The aim of the this exercise is to familiarize yourself with the halfedge-based mesh data structure. Moreover, you will get an idea of how the various normals are different from each other and how to compute curvatures with discrete operators. You will need to modify `Curvature.cc` (should already have), and use the plugin on meshes in `dgp-exercise4.zip`. Please unzip and place the `.off` and `.obj` files in a folder on your machine.

Computing Vertex Normals (5pt)

Normal vectors for individual triangles $T = (\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$ can be computed as the normalized cross-product of two triangle edges:

$$\boldsymbol{n}(T) = \frac{(\mathbf{x_j} - \mathbf{x_i}) \times (\mathbf{x_k} - \mathbf{x_i})}{\left\|(\mathbf{x_j} - \mathbf{x_i}) \times (\mathbf{x_k} - \mathbf{x_i})\right\|}. \tag{5}$$

Computing vertex normals as spatial averages of normal vectors in a local one-ring neighborhood leads to a normalized weighted average of the (constant) normal vectors of incident triangles:

$$\boldsymbol{n}(x_i) = \frac{\sum_{T \in \mathcal{N}_1(x_i)} \alpha_T \, \boldsymbol{n}(T)}{\left\|\sum_{T \in \mathcal{N}_1(x_i)} \alpha_T \, \boldsymbol{n}(T)\right\|} \tag{6}$$

where α_T are weights. In this exercise you will compute vertex normals with three most frequently used types of weights.

- Consider the weights are constant $\alpha_T = 1$. Implement the `compute_normals_with_constant_weights()` function in file `Curvature.cc`.
- Let the weighting be based on triangle area, i.e., $\alpha_T = |T|$. Exploit the relationship between vector cross-product and triangle area to simplify the implementation. Implement the `compute_normals_by_area_weights()` function in file `Curvature.cc`.
- Consider weighting by incident triangle angles $\alpha_T = \theta_T$ (see Figure 1). The involved trigonometric functions make this method computationally more expensive, but it gives superior results in general. Implement the `compute_normals_with_angle_weights()` function in file `Curvature.cc`.

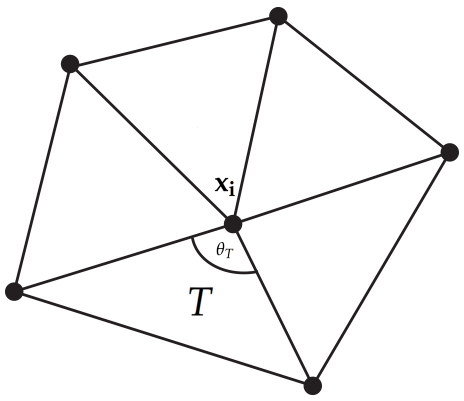
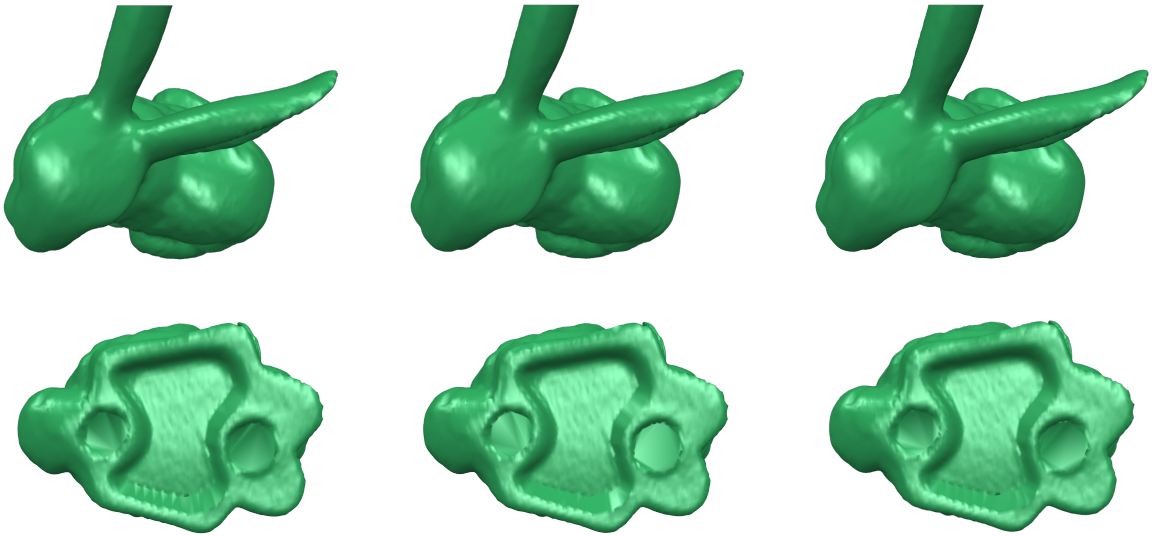


Figure 1: Incident triangle angle for normal weights.

You need to compute normals for all vertices and store them in vertex property `vertex_normal_`. To visualize, you need to choose the corresponding normal type in the combox and click on the `Show Normal` button. Observe the difference in the rendering when the normals are computed with three different versions for weights (see Figure 2 for example).



(a) Constant weights (b) Area weights (c) Angle weights

Figure 2: Difference in rendering when computing the normals with different weights.

2.1 Uniform Laplace Operator (3pt)

The uniform Laplace operator approximates the Laplacian of the discretized surface using the centroid of the one-ring neighborhood. For a vertex v let us denote the N neighbor vertices with v_i. The uniform Laplace approximation is

$$L_U(v) = \frac{1}{N}\sum_{i}^{|N|}(v_i - v)$$

Implement the uniform Laplace operator in the function `calc_uniform_laplacian()` in the `Curvature.cc` file. Store $(L_U(v) \cdot \boldsymbol{n})/2$ in vertex property `vertex_curvature_`. For the vertex normals, please use the constant weight normals (you may have to run the normal calculation again to set this). Store the minimal Laplacian value in the `min_curvature_` and the maximal Laplacian value in `max_curvature_`. To display the per-vertex Laplacian operator, choose `Uniform Laplacian` in the combox and then click on the `Show Curvature` button. The minimal and maximal Laplacian value will be displayed on the standard output. You should get the result similar to Figure 3.

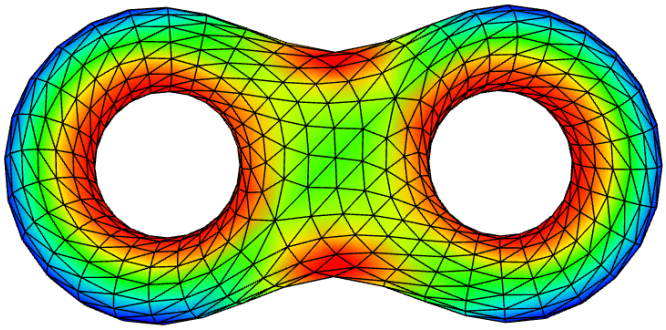


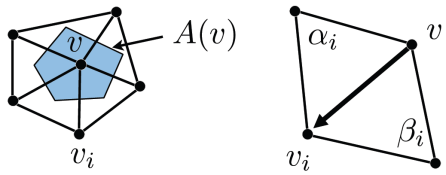
Figure 3: The uniform Laplacian operator at each vertex.

2.2 Laplace-Beltrami Curvature (3pt)

The discretization of the uniform Laplacian does not depend on vertex coordinates and therefore does not take into account the geometry of the mesh. To obtain a mean curvature approximation we need to introduce weights regarding the geometry. The Laplace-Beltrami operator uses the following weights for the neighbor vertices:

$$L_B(v) = \frac{1}{2A}\sum_{i}^{|N|}(\cot\alpha_i + \cot\beta_i)(v_i - v)$$

See the lecture slides and the picture on the right for explanation about this formula. Again, the half length of the Laplace-Beltrami approximation gives an approximation of the mean curvature. Study the `calc_weights()` function to understand how and which weights are computed. Use the stored weights values to implement the mean curvature approximation using the Laplace-Beltrami operator. The `calc_mean_curvature()` function in the `Curvature.cc` file has to fill the `vertex_curvature_` property with the mean curvature approximation values. Store the minimal curvature value in `min_curvature_` and the maximal curvature value in `max_curvature_`. To display the per-vertex Laplacian operator, choose `Laplace-Beltrami` in the combox and then click on the `Show Curvature` button. The minimal and maximal curvature value will be displayed on the standard output. You should get the result similar to Figure 4.



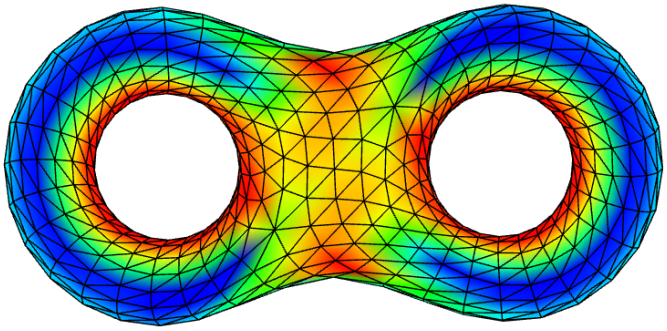
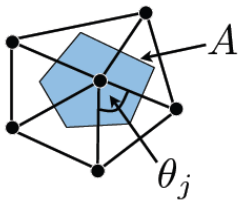


Figure 4: The Laplace-Beltrami approximation of the mean curvature at each vertex.

2.3 Gaussian Curvature (3pt)

In the lecture you have been presented an easy way to approximate the Gaussian curvature on a triangle mesh. The formula uses the sum of the angles around a vertex and the same associated area which is used in the Laplace-Beltrami operator:

$$G = (2\pi - \sum_j \theta_j) / A$$



Implement the `calc_gauss_curvature()` function in the `Curvature.cc` file so that it stores the Gaussian curvature approximations in the `vertex_curvature_` vertex property. Note that the `vertex_weight_` property already stores $\frac{1}{2A}$ value for every vertex, you do not need to calculate A again. Store the minimal curvature value in `min_curvature_` and the maximal curvature value in `max_curvature_`. For the "eight" mesh you should get a Gaussian curvature approximation like on Figure 5.

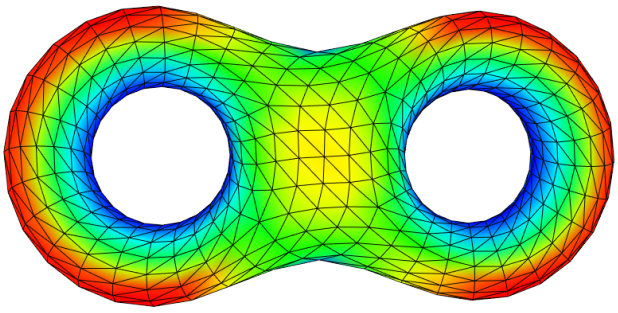


Figure 5: Approximation of the Gaussian curvature at each vertex.

The blue color corresponds to the minimal value and the red color corresponds to the maximal value of the current mesh. Explore the curvature of different given meshes. In addition you are given a small sphere and a 10 times bigger sphere. Observe what happens with the Uniform Laplacian and the Laplace-Beltrami operator on the spheres of different sizes (*hint: check on the maximal and minimal values*). Compare the results and comment on the difference. Write down your findings in the `readme.txt`.