

# Digital 3D Geometry Processing

## Exercise 5 – Surface Normals, Curvature

Handout date: 26.03.2019

Submission deadline: 02.04.2019, 13:00 h

### What to 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:

- Hand in **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. Indicate what fraction of the total workload each project member contributed.
- 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 ILIAS before the submission deadline. Late submissions will receive 0 points! The total points of this homework is 10.

### Goal

The aim of the this exercise is to make yourself familiar with the halfedge based mesh data structure. Moreover, you will get an idea on the difference of different normals and how to compute curvatures with discrete operators.

## 1 Computing Vertex Normals

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:

$$\mathbf{n}(T) = \frac{(\mathbf{x}_j - \mathbf{x}_i) \times (\mathbf{x}_k - \mathbf{x}_i)}{\|(\mathbf{x}_j - \mathbf{x}_i) \times (\mathbf{x}_k - \mathbf{x}_i)\|}. \quad (1)$$

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:

$$\mathbf{n}(x_i) = \frac{\sum_{T \in \mathcal{N}_1(x_i)} \alpha_T \mathbf{n}(T)}{\left\| \sum_{T \in \mathcal{N}_1(x_i)} \alpha_T \mathbf{n}(T) \right\|} \quad (2)$$

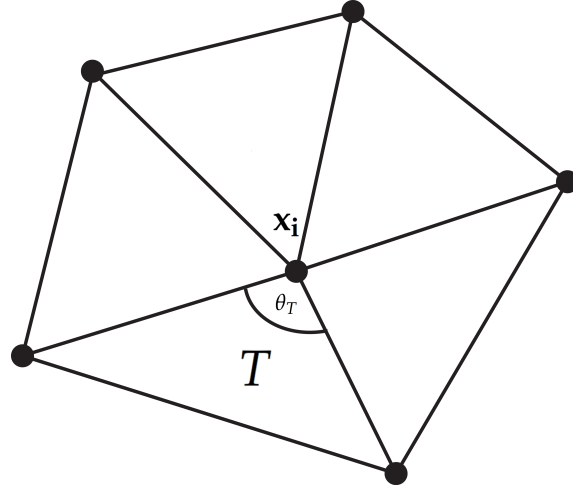


Figure 1: Incident triangle angle for normal weights.

where  $\alpha_T$  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`.

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 combobox 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).

## 2.1 Uniform Laplace Operator

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 the length of the computed vector  $L_U(v)$  in vertex property `vertex_curvature_`. Store the minimal Laplacian value in the `min_curvature_`.

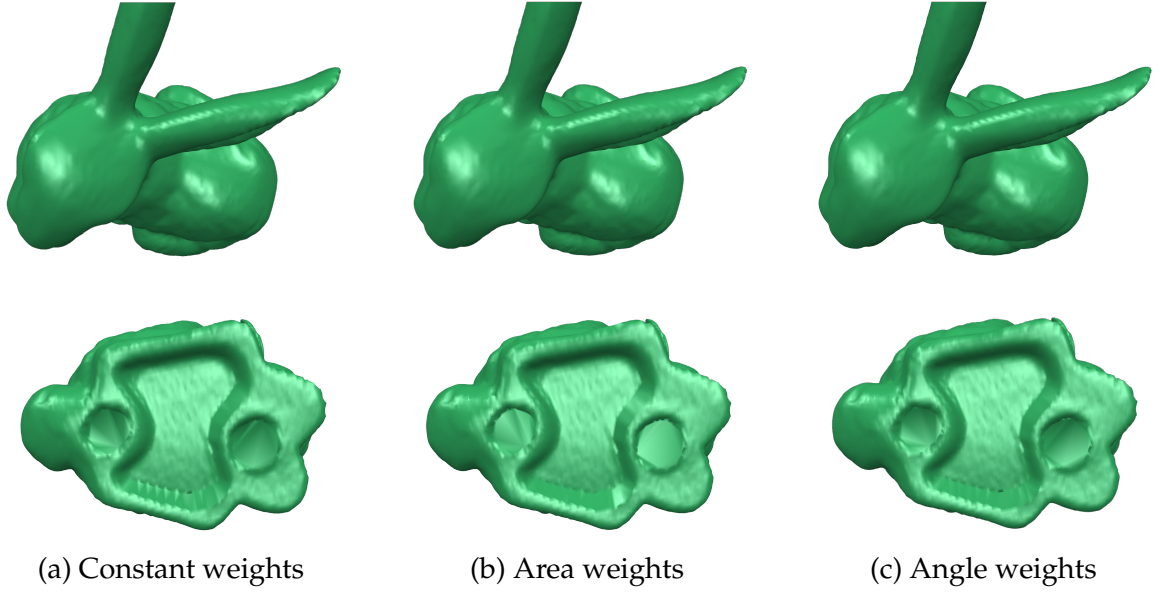


Figure 2: Difference in rendering when computing the normals with different weights.

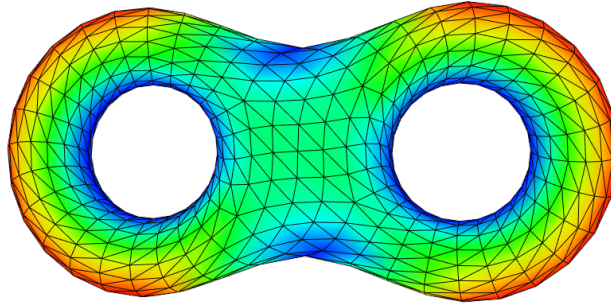


Figure 3: The uniform Laplacian operator at each vertex.

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.

## 2.2 Laplace-Beltrami Curvature

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:

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

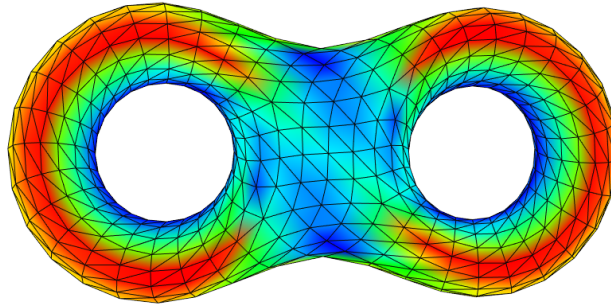
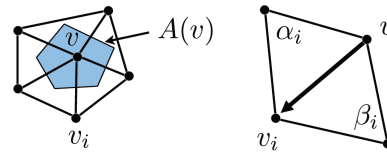


Figure 4: The Laplace-Beltrami approximation of the mean curvature at each vertex.

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

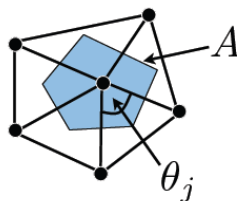


to understand how and which weights are computed. Use the stored weights values to implement the mean curvature approximation using the Laplace-Beltrami operator. The `calc_mean_curvature()` function in the `Curvature.cc` file has to fill the `vertex_curvature_` property with the mean curvature approximation 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 combobox 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.

## 2.3 Gaussian Curvature

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.

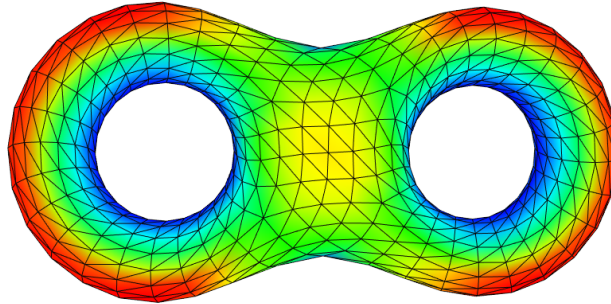


Figure 5: Approximation of the Gaussian curvature at each vertex.

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`.