Digital 3D Geometry Processing
Exercise 4 – Curvature, Smoothing and Feature Enhancement

Handout date: 19.10.2016

Submission deadline: 27.10.2016, 23:00 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.
- 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 Moodle before the submission deadline. Late submissions will receive 0 points!

Goal

In this exercise you will implement the following tasks:

- Computing the uniform Laplacian approximation of the mean curvature;
- Computing the Laplace-Beltrami approximation of the mean curvature;
- Approximating the Gaussian curvature;
- Surface smoothing using the uniform Laplace operator and the Laplace-Beltrami operator;
- Enhance surface features, using the difference between a mesh surface and its Laplace smoothing.

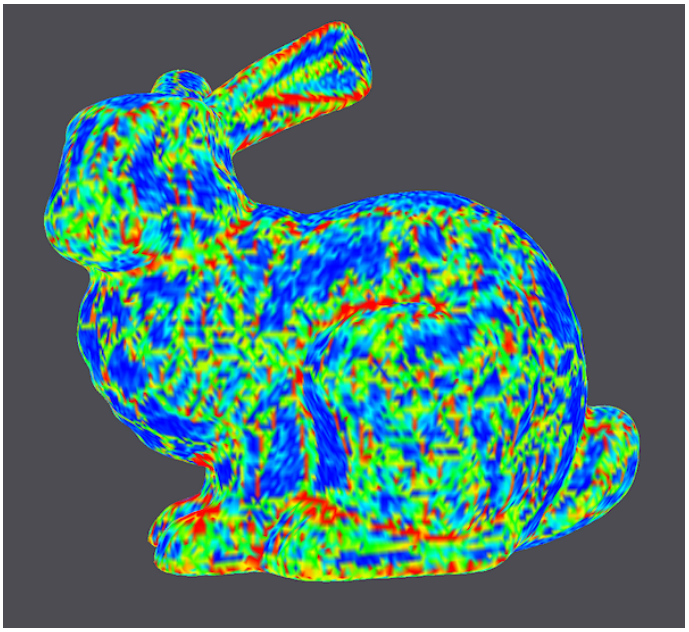


Figure 1: The uniform Laplacian approximation of the mean curvature at each vertex.

1.1 Uniform Laplace curvature

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

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

The half length of the vector L_U is an approximation of the mean curvature.
Implement the uniform Laplace approximation of the mean curvature in the function `calc_uniform_mean_curvature()` in the `viewer.cpp` file. Store the computed values in vertex properties `v_unicurvature`. To display the per-vertex mean curvature values click on the `Curvature -> Uniform Laplacian` button. You should get the result similar to Figure 1.

1.2 Laplace-Beltrami curvature

For irregular meshes the uniform Laplace smoothing moves vertices not only along the surface normal, but tangentially, as well. To create a smoothing which moves vertices only along surface normals one can use the Laplace-Beltrami operator. This operator uses the certain weights for the neighbor vertices:

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

$$L_B(v) = w \sum_{i}^{|N|} w_i (v_i - v)$$

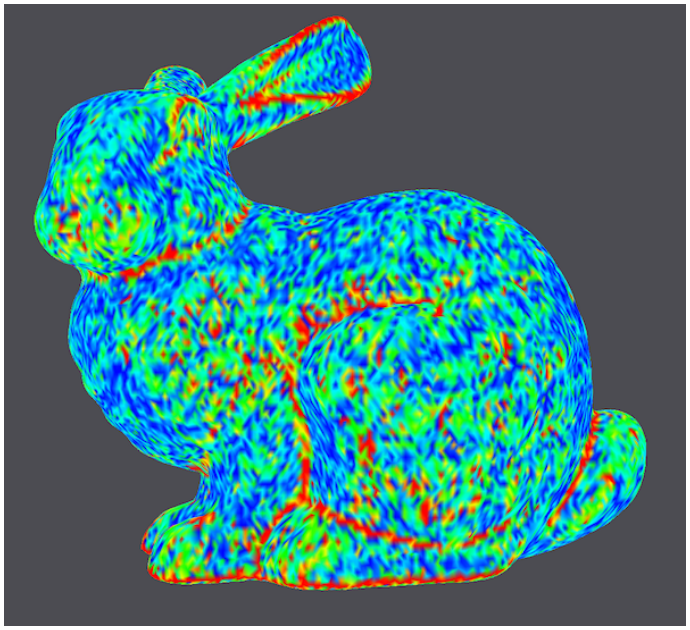
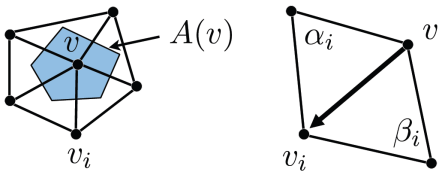


Figure 2: The Laplace-Beltrami approximation of the mean curvature at each vertex.



See the lecture slides and the above picture for explanation about this formula. Again, the half length of the Laplace 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 `viewer.cpp` file has to fill the `v_curvature` property with the mean curvature approximation values. To display the per-vertex mean curvature values click on the `Curvature -> Laplace-Beltrami` button. You should get the result similar to Figure 2.

1.3 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:

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

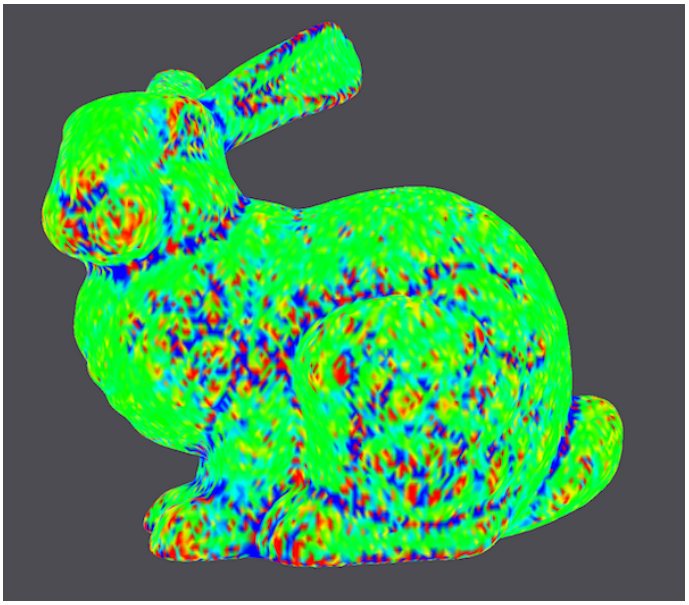
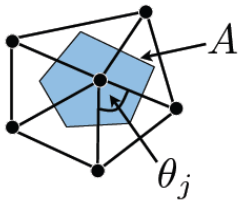


Figure 3: Approximation of the Gaussian curvature at each vertex.



Implement the `calc_gauss_curvature()` function in the `viewer.cpp` file so that it stores the Gaussian curvature approximations in the `v_gauss_curvature` vertex property. Note that the `v_weight` property already stores $\frac{1}{2A}$ value for every vertex, you do not need to calculate A again. For the bunny dataset you should get a Gaussian curvature approximation like on Figure 3.

2.1 Uniform Laplacian Smoothing

In uniform Laplacian smoothing, the position of each vertex is updated according to the uniform Laplace operator of the mesh at that vertex. Specifically, for a vertex v, its new position v' is computed as

$$v' = v + \frac{1}{2} L_U(v), \tag{1}$$

where L_U is the uniform Laplace operator:

$$L_U(v) = (\frac{1}{n} \sum_i v_i) - v,$$

with $\{v_i\}$ being the one-ring neighbors of v in the mesh, and n being the number of neighbors.
Within the framework, you need to implement uniform Laplacian smoothing by completing the member function `Viewer::uniform_smooth(...)`. The mesh is stored in the data member `mesh`, and you need to update its vertex positions according to the

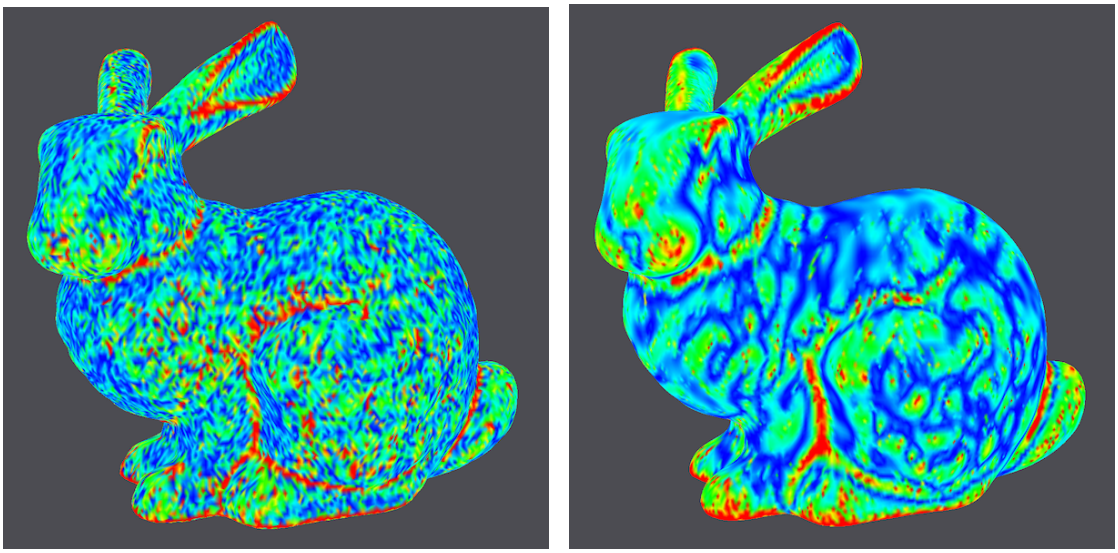


Figure 4: Mean curvature approximation of the bunny model, before and after 10 iterations of uniform Laplacian smoothing.

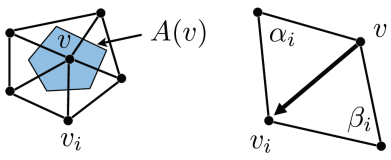
smoothing result. **Note that you only need to update non-boundary vertices, since the Laplace operator is not well-defined for boundary vertices.** Afterwards, you can press the `Smooth -> Uniform Laplacian` button in the viewer to run 10 iterations of uniform Laplacian smoothing. Example is shown in Figure 4. Note that uniform Laplacian smoothing moves a vertex towards the centroid of its one-ring neighbors.

2.2 Laplace-Beltrami Smoothing

In general, the uniform Laplacian smoothing moves vertices not only along the surface normals, but also tangentially along the surface. In order to move vertices only along the surface normals, one can use Laplace-Beltrami smoothing. This operator is closely related to the Laplace-Beltrami operator that you have implemented in the previous exercise. Remember that the Laplace-Beltrami operator for a vertex v is

$$L_B(v) = \frac{1}{2A(v)} \sum_i w_i (v_i - v), \tag{2}$$

where $A(v)$ is the area of the cell associated with v, $\{v_i\}$ are the one-ring neighbors of v, and $w_i = \cot \alpha_i + \cot \beta_i$ is the cotan weight corresponding to the edge $\overline{v\, v_i}$ (see the figure on the right). The Laplace-Beltrami smoothing is done using a 'normalized' Laplace-Beltrami operator, such that the new position v' of vertex v is computed as



$$v' = v + \frac{1}{2} \overline{L}_B(v),$$

where

$$\overline{L}_B(v) = \frac{1}{\sum_i w_i} \sum_i w_i (v_i - v). \tag{3}$$

Note the difference between Eq. (2) and Eq. (3): the normalization factor $2A(v)$ in (2) is replaced by $\sum_i w_i$ in (3).

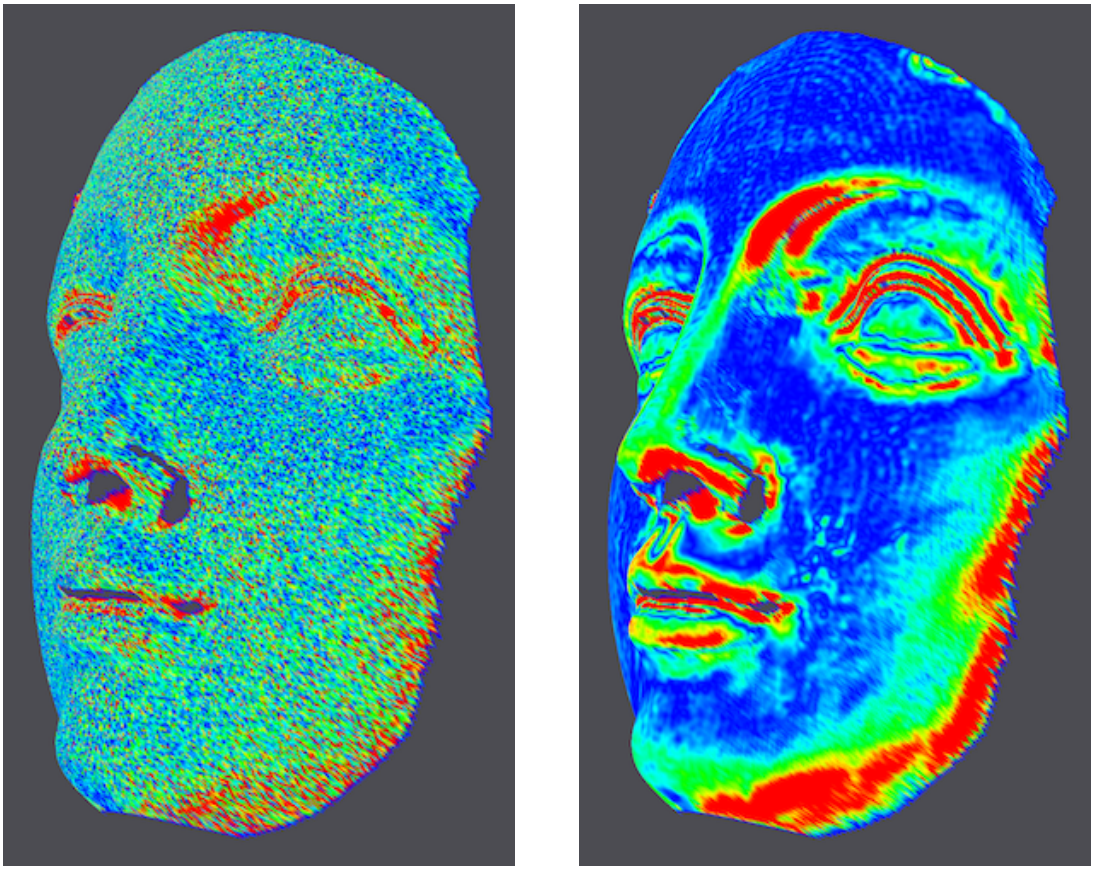


Figure 5: Mean curvature approximation on the scanned face model, before and after 10 iterations of Laplace-Beltrami smoothing.

You need to complete the member function `Viewer::smooth(...)` for Laplace-Beltrami smoothing. Similar to uniform Laplacian smoothing, **you only need to update non-boundary vertices.** You can use the member function `calc_edge_weights()` to pre-compute the edge weights $\{w_i\}$, which will be stored in the edge property `e_weight`. Afterwards, pressing the `Smooth -> Laplace-Beltrami` button in the viewer will perform 10 iterations of Laplace-Beltrami smoothing. An example is shown in Figure 5.

3 Feature Enhancement

Mesh smoothing using Laplace operators can be seen as low-pass filtering, such that the resulting mesh consists of low-frequency geometric signals from the input mesh. And the high-frequency features of the input mesh are represented by the difference between the vertex positions before and after smoothing. This allows us to enhance the features of the input mesh as follows. Let $\{v_j^{\text{in}}\}$ and $\{v_j^{\text{out}}\}$ be the vertex positions before and after smoothing, respectively. Then we can compute the vertex positions $\{v_j^*\}$ of an enhanced mesh by

$$v_j^* = v_j^{\text{out}} + \alpha \cdot (v_j^{\text{in}} - v_j^{\text{out}}). \tag{4}$$

Here $v_j^{\text{in}} - v_j^{\text{out}}$ corresponds to the high-frequency features we want to enhance, and $\alpha \geq 1$ is a scalar coefficient that determines the magnitude of enhancement. When $\alpha = 1$ we recover the input mesh, while for $\alpha > 1$ the features are amplified.
Within the framework, there are two member functions for feature enhancement:

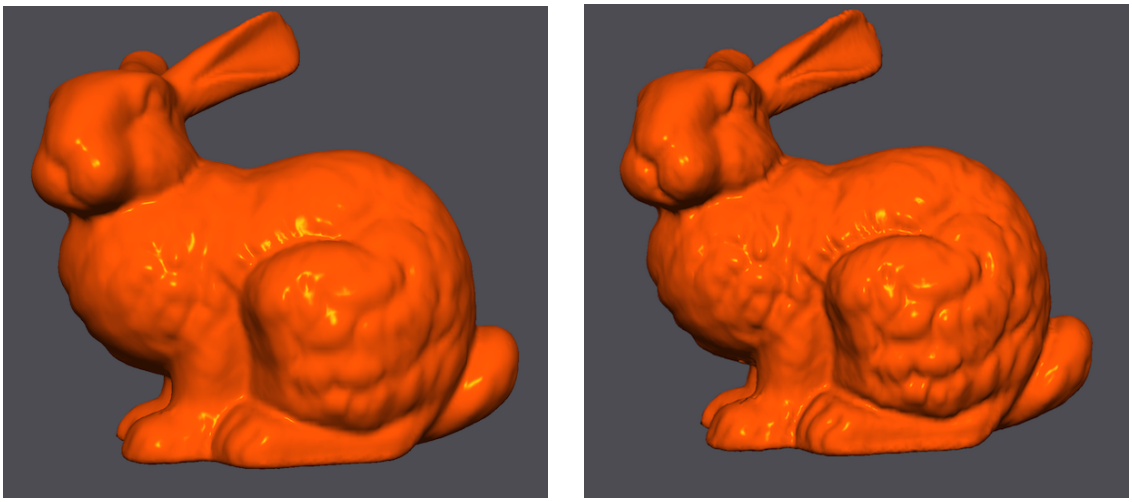


Figure 6: Feature enhancement of the bunny model, using 10 iterations of Laplace-Beltrami smoothing, and with $\alpha = 2.0$.

- `Viewer::uniform_laplacian_enhance_feature(...)`
- `Viewer::laplace_beltrami_enhance_feature(...)`

They perform feature enhancement using the uniform Laplace operator and the Laplace-Beltrami operator, respectively. You need to complete these two functions by implementing the following:

1. Perform a certain number of iterations of mesh smoothing, using the functions `smooth(...)` or `uniform_smooth(...)`. The required number of iterations can be modified using the GUI.
2. Compute the enhanced mesh vertex positions according to Eq. (4), and update the data member `mesh` accordingly. The value of α in Eq. (4) can be changed using the GUI.

Afterwards, you can perform feature enhancement in the viewer using the `Enhancement` button. Figure 6 shows an example with these parameters. You can modify the number of iterations and α to experiment with different parameter values using the GUI. Note, however, that in general you cannot set α to a very large value, because this can lead to self-intersection of the enhanced mesh in concave regions.
Typically we can only perform feature enhancement for a few times on a given mesh, before it contains flipped triangles. You can recover from such bad shapes using a few iterations of uniform Laplacian smoothing. With this approach, you can alternate between multiple iterations of feature enhancement, and uniform Laplacian smoothing to recover from flipped triangles. Figure 7 shows a result that is achieved in this way.

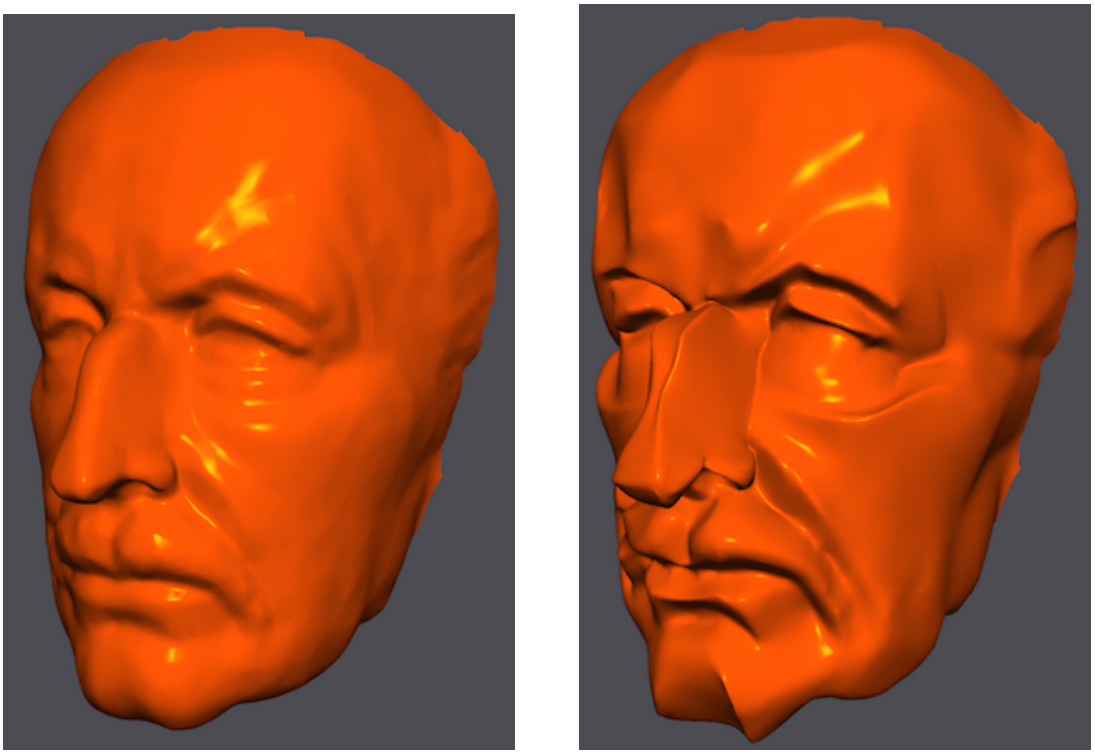


Figure 7: Processing result on the Max Plank model, by alternating between of feature enhancement and uniform Laplacian smoothing.

In your submission, please also provide the following:

- **At least three** images of interesting feature enhancement results.
- In the `readme.txt` file, explain how you achieve these results (including parameter values, the order of applying enhancement/smoothing, the iterations of enhancement/smoothing, etc.).
- From a signal processing point of view, what are the effects of the operations you apply, and why do they produce the results you show? Please provide your answer in the `readme.txt` file.