Digital 3D Geometry Processing
Exercise 7 – Smoothing and Feature Enhancement

Handout date: 30.10.2018

Submission deadline: 08.11.2018, 23:00 h

What to hand in

A .zip compressed file renamed to `Exercise`n`-Group`i`.zip` where n is the number of the current exercise sheet and i is the number of your group. 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 Moodle before the submission deadline. Late submissions will receive 0 points!

Goal

In this exercise you will implement the following tasks in the file `mesh_processing.cpp`:

- Surface smoothing using the uniform Laplace operator and the normalized cotan Laplacian operator;
- Implicit surface smoothing by solving a linear system;
- Enhance surface features, using the difference between a mesh surface and its Laplace smoothing.

1 Explicit Smoothing

Explicit integration of the diffusion equation with the cotangent discretization of the Laplace-Beltrami operator quickly becomes unstable with increasing time steps. In practice, using a normalized Laplacian has proven useful:

$$L(\mathbf{p}_i) = \frac{1}{\sum_j w_j} \sum_j w_j (\mathbf{p}_j - \mathbf{p}_i),$$

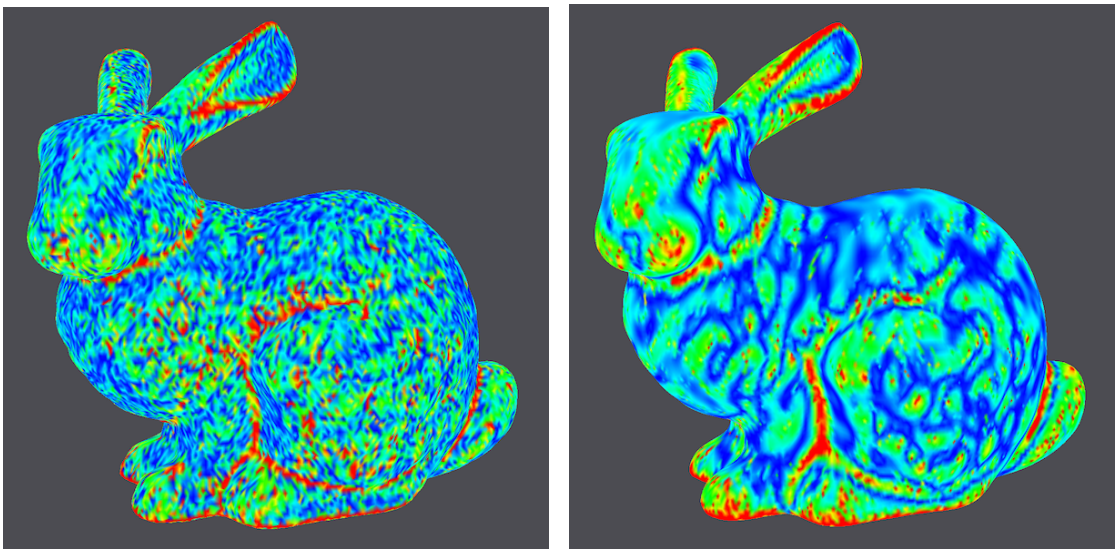
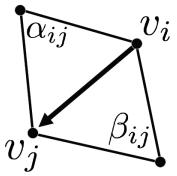


Figure 1: Mean curvature approximation of the bunny model, before and after 10 iterations of Uniform Laplacian smoothing.



where $\mathbf{p}_i$ is a position of vertex v_i, and $\{v_j\}$ are the one-ring neighbors of v_i. We consider two variants. One is using constant weights $w_j = 1$, and we will call it *Uniform Laplacian*. The other one is using cotan weights $w_j = \cot\alpha_{ij} + \cot\beta_{ij}$ corresponding to the edge $\overline{v_i\, v_j}$ (see the figure on the right). This one is called *Normalized Cotan Laplacian*, but for the simplicity of notation in this exercise in the future text we will refer to it as *Cotan Laplacian*.
Implement both versions of explicit smoothing and comment on the differences you observe.
Within the framework, you need to implement uniform Laplacian smoothing and cotan Laplacian smoothing by completing the member function `MeshProcessing::uniform_smooth(...)` and `MeshProcessing::smooth(...)` respectively. The mesh is stored in the data member `mesh_`, and you need to update its vertex positions according to the smoothing result. The position $\mathbf{p}_i$ of each vertex v_i is updated to a new position $\mathbf{p}_i'$ as

$$\mathbf{p}_i' = \mathbf{p}_i + \delta t \lambda L(\mathbf{p}_i),$$

where $L(\mathbf{p}_i)$ is a uniform Laplacian or cotan Laplacian, δt time step and λ a diffusion constant. In your implementation $\delta t\lambda$ should be a constant number smaller or equal to $\frac{1}{2}$. Note that you should only update non-boundary vertices and keep boundary vertices fixed. You can use the member function `calc_edge_weights()` to precompute the edge weights $\{w_j\}$, which will be stored in the edge property `e_weight`. Afterwards, you can press the `Smooth -> Uniform Laplacian` button or `Smooth -> Cotan Laplacian` in the viewer to run 10 iterations of smoothing. Examples are shown in Figure 1 and Figure 2.

2 Implicit smoothing

Implicit integration avoids the difficulties of instability and allows using larger timesteps. Implement the implicit smoothing algorithm, using the method presented in class, which is based on solving the following equation:

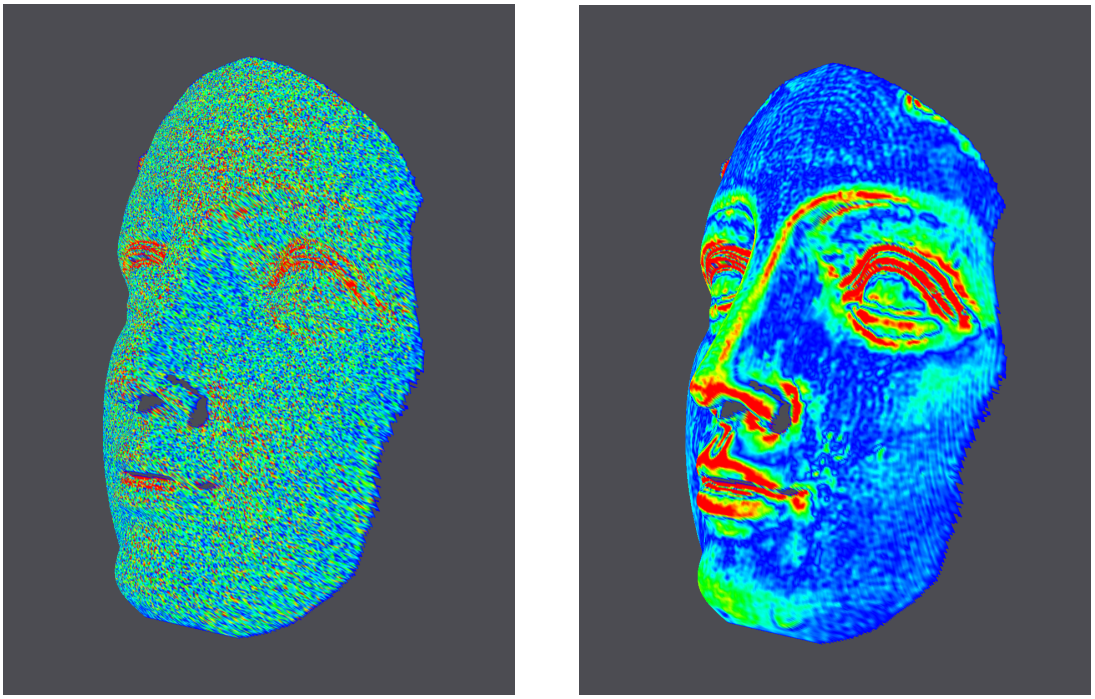


Figure 2: Mean curvature approximation on the scanned face model, before and after 10 iterations of Cotan Laplacian smoothing.

$$(\mathbf{D}^{-1} - \delta t \lambda \mathbf{M})\mathbf{P}^{(t+1)} = \mathbf{D}^{-1}\mathbf{P}^{(t)}.$$

This method is called via the `Smooth -> Implicit Smoothing` button and the code is located in the `implicit_smoothing(...)` method inside `mesh_processing.h/cpp`. The effect on the bunny mesh is shown in Figure 3.

For curves we saw that any curve under curvature flow goes to a convex shape and then converges to a point (Gage-Hamilton-Grayson theorem, see this webpage `https://en.wikipedia.org/wiki/Curve-shortening_flow#Gage.E2.80.93Hamilton.E2.80.93Grayson_theorem`).
Do you experience an analogous behavior for surfaces? Experiment with various meshes, time steps, and number of iterations. Briefly comment on your observations (no formal proof expected).

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 $\{\mathbf{p}_j^{\text{in}}\}$ and $\{\mathbf{p}_j^{\text{out}}\}$ be the vertex positions before and after smoothing, respectively. Then we can compute the vertex positions $\{\mathbf{p}_j^*\}$ of an enhanced mesh by

$$\mathbf{p}_j^* = \mathbf{p}_j^{\text{out}} + \alpha \cdot (\mathbf{p}_j^{\text{in}} - \mathbf{p}_j^{\text{out}}). \tag{1}$$

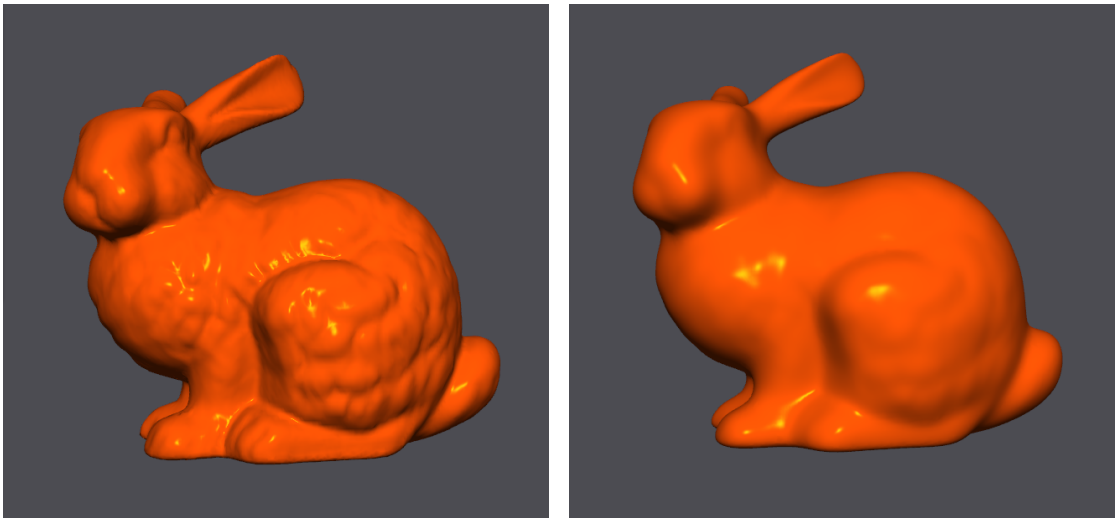


Figure 3: The Stanford bunny and its smoothed version using implicit smoothing.

Here $\mathbf{p}_j^{\text{in}} - \mathbf{p}_j^{\text{out}}$ 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:

- `MeshProcessing::uniform_laplacian_enhance_feature(...)`
- `MeshProcessing::cotan_laplacian_enhance_feature(...)`

They perform feature enhancement using the uniform Laplace operator and the cotan Laplacian 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. (1), and update the data member `mesh` accordingly. The value of α in Eq. (1) can be changed using the GUI.

Afterwards, you can perform feature enhancement in the viewer using the `Enhancement` button. Figure 4 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 5 shows a result that is achieved in this way.
You are given a bad quality mesh `Bad Max` and a good quality mesh `Nice Max` of a same model. Observe the difference when applying the algorithms on these meshes. Why are the methods unstable when applied on a bad quality mesh?
In your submission, please also provide the following:

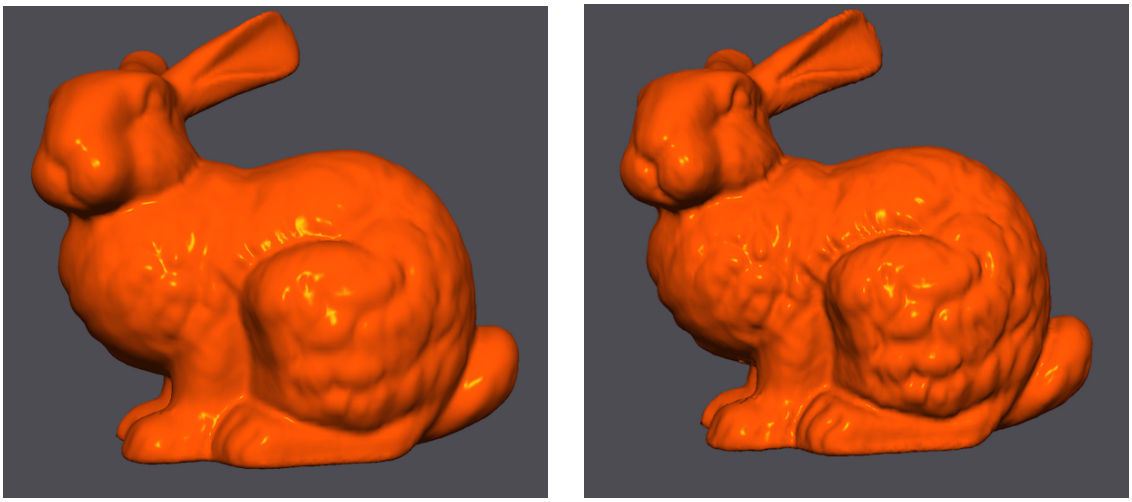


Figure 4: Feature enhancement of the bunny model, using 10 iterations of Cotan Laplacian smoothing, and with $\alpha = 2.0$.

- **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.

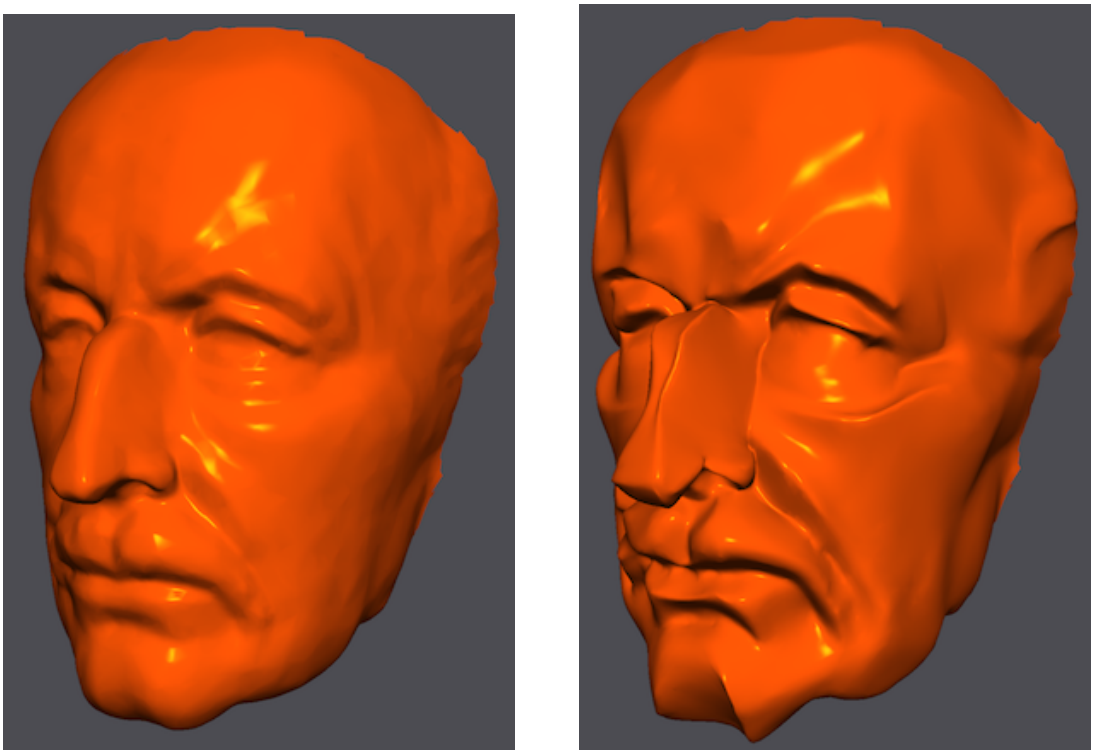


Figure 5: Processing result on the Max Plank model, by alternating between of feature enhancement and uniform Laplacian smoothing.