Digital 3D Geometry Processing
Exercise 8 - Remeshing

Handout date: 05.11.2018

Submission deadline: 15.11.2018, 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. Indicate what fraction of the total workload each project member contributed.
- 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 a surface-based remeshing algorithm. The algorithm is composed of the four main steps:

- Split long edges;
- Collapse short edges;
- Flip edges to improve vertex valences;
- Implement tangential smoothing to improve triangle quality;
- Compute the target edge lengths for adaptive remeshing.

Splitting long edges

Implement the `split_long_edges()` function so that it splits edges longer than the 4/3 of the edges target length L in two halves. In order to compute the target length of an edge compute the mean of the property `target_length` of the edge's two vertices.

Splitting the edge (use the function `mesh.split()`) requires some additional operations. A new vertex needs to be added to the mesh. For that vertex you need to compute the normal (use the functions `set_normal()` and `get_normal()`) and interpolate `target_length` property.

The loop tries to split edges until no longer edge is in the mesh, or a maximum threshold of 100 iterations have been executed. Similarly all subsequent tasks will use this limit to make sure the algorithm finishes and does not run for unreasonable long time.

Collapsing short edges

Complete the `collapse_short_edges()` function. You shouldn't consider for collapse half-edges going from a boundary to a non-boundary vertex to avoid shrinking around the boundaries. Note that their opposite edges are fine for collapse. Use `mesh.is_boundary()` function to learn whether a given vertex is on the boundary.

Check if the edge is shorter than the 4/5 of the edges target length L. If so, check if both of the halfedges corresponding to the edge are collapsible. The `mesh.is_collapse_ok()` function checks if a halfedge can be collapsed. If they are, you should collapse (use `mesh.collapse()`) the lower valence vertex into the higher one (use `mesh.valence()`). Otherwise collapse the halfedge which is collapsible, or don't collapse at all if both of the tests returned false.

Equalizing valences

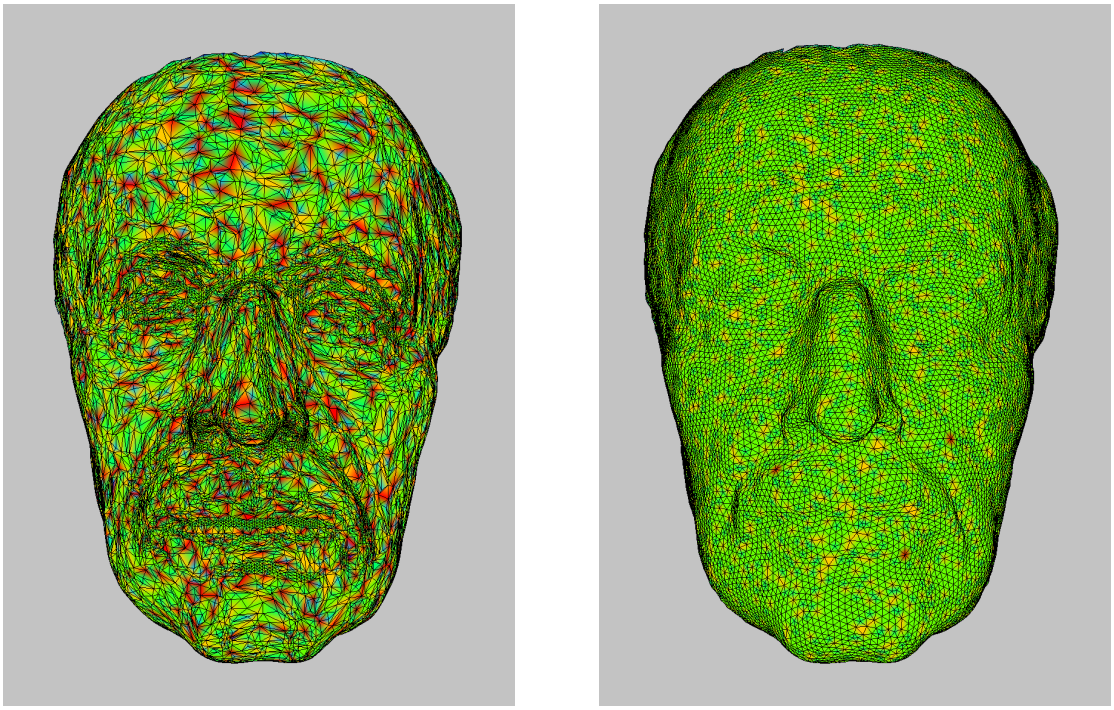
Complete the `equalize_valences()` function, so that it flips vertices if it improves vertex valences in the local neighborhood.

We know that an "ideal" mesh vertex has valence 4 if it lies on the boundary and 6 otherwise. For an edge e now consider the two end vertices and the two other vertices on the neighbor triangles to e. For every vertex compute its valence deviation, i.e. the difference between the current valence and the optimal valence for this vertex. By comparing the sum of squared valence deviation before and after an eventual edge flip you can decide if the edge flip will improve the valences locally (smaller valence deviations are better). If the edge improves on the local valences, flip it.

Don't forget to use the `mesh.is_flip_ok()` function in order to make sure an edge can be flipped before you try to flip it.

Tangential smoothing

Implement the `tangential_relaxation()` function to improve the triangle shapes by smoothing vertices in the tangent plane of the mesh. Approximate the mean curvature with the uniform Laplacian. Now, decompose this vector into two components: one



(a) Valences before

(b) Valences after

Figure 1: Meshes before and after remeshing.

parallel to the vertex normal and one parallel to the tangent plane in the vertex (perpendicular to the normal). Use the tangential component to move the vertex and thus improve the triangle quality.

These four remeshing steps should lead to results shown in Figure 1.

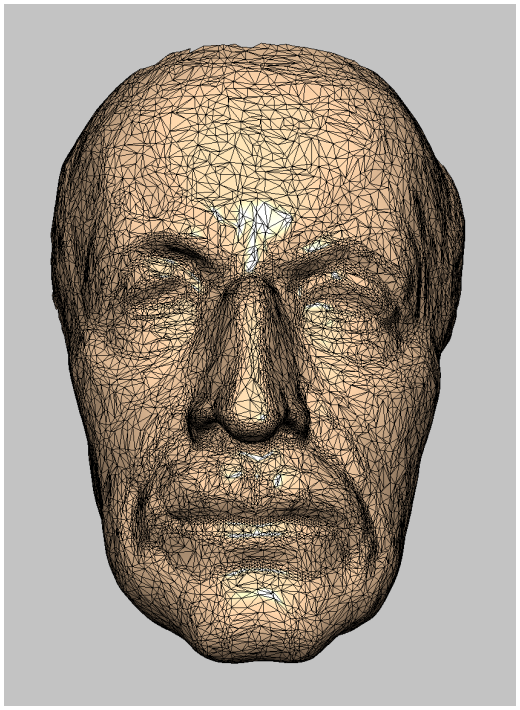
Does this algorithm lead to a stationary solution after finite number of steps, i.e. does the mesh not change anymore when more remeshing steps are applied? Experiment with your code to gain insights into this question. Explain your insights in a few sentences.

Adaptive remeshing

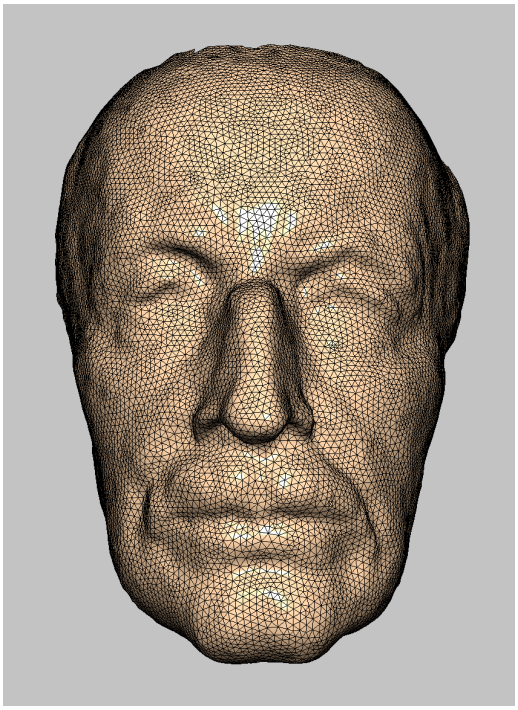
Reimplement the `calc_target_length()`, which calculates the vertex property `target_length` for adaptive remeshing. The general idea is that we encourage splits on edges where there are high maximal curvatures, but do not split edges with lower curvature. Given the mean curvature H and the Gaussian curvature K the following equality states the principal curvatures:

$$k_{max} = H + \sqrt{H^2 - K}$$
$$k_{min} = H - \sqrt{H^2 - K}$$

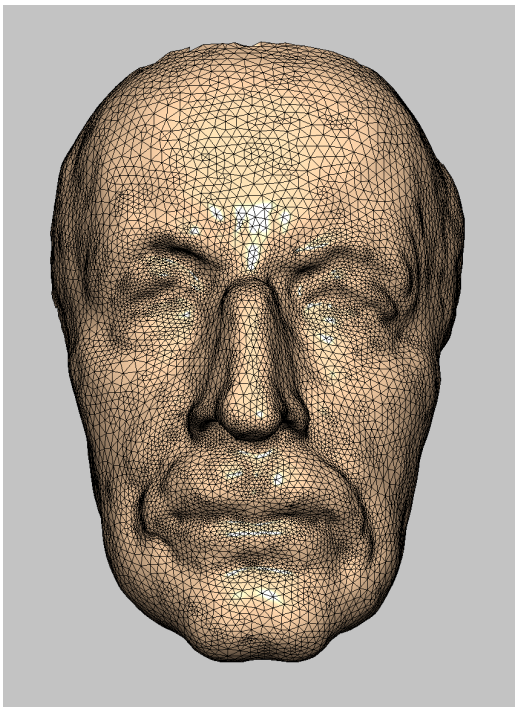
In order to adapt the target length at each vertex, scale it by the inverse of k_{max}. Since the curvature estimates are usually noisy, but we aim for a regular meshing, apply a few iteration of uniform smoothing to the resulting target length property. Finally, scale the target length property such that it's mean over all the vertices equals the user specified target length. This adaptive remeshing should lead to results shown in Figure 2.



(a) Original mesh



(b) Fixed target length remeshing



(c) Adaptive remeshing

Figure 2: Initial mesh and results from different target edge length strategies.