Digital 3D Geometry Processing
Exercise 6 - Remeshing

Handout date: 08.11.2016

Submission deadline: 16.11.2016, 23:00 h

Goal

In this exercise you will implement a surface-based remeshing algorithm. The algorithm is composed of the four main steps:

- Computing the target edge lengths;
- Split long edges;
- Collapse short edges;
- Flip edges to improve vertex valences;
- Tangential smoothing to improve triangle quality.

Compute the target edge lengths

In this exercise, we will use 3 strategies to compute the target edge lengths. They are selected by checking the corresponding box in the menu and then pressing the `Remesh` button.

1. *Scaled average edge length.* Compute the average edge length of the current mesh and scale it up or down.
2. *Height-based.* Compute the average edge length and scale it based on the height of each vertex.
3. *Curvature-based adaptive remeshing.* Compute the target edge lengths as a function of the mesh curvature (see the last section of the exercise sheet for more details).

Figure 2 shows the results for the different target edge length strategies.

You are encouraged to experiment with other strategies and send in screenshots with brief explanations.

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_lengths` 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 and interpolate `target_lengths` 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 an 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 edge 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.

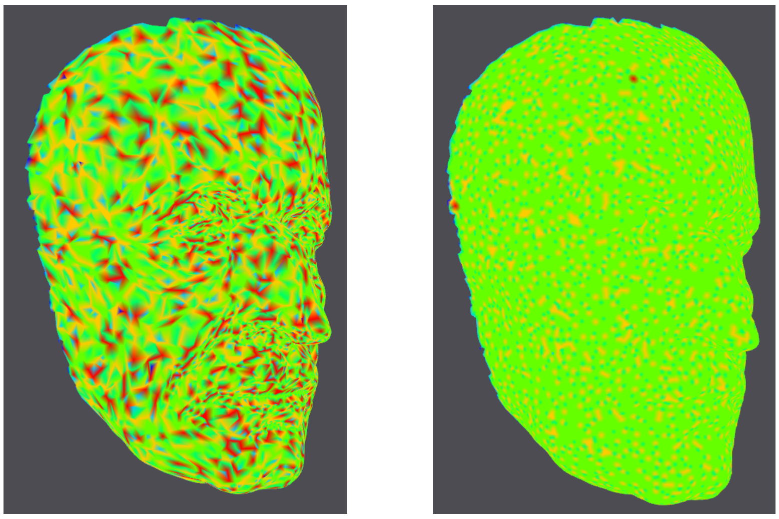

(a) Valences before (b) Valences after

Figure 1: Meshes before and after remeshing

Similarly to the exercise “Surface Quality”, approximate the mean curvature with the uniform Laplacian. Now, decompose this vector into two components: one 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.

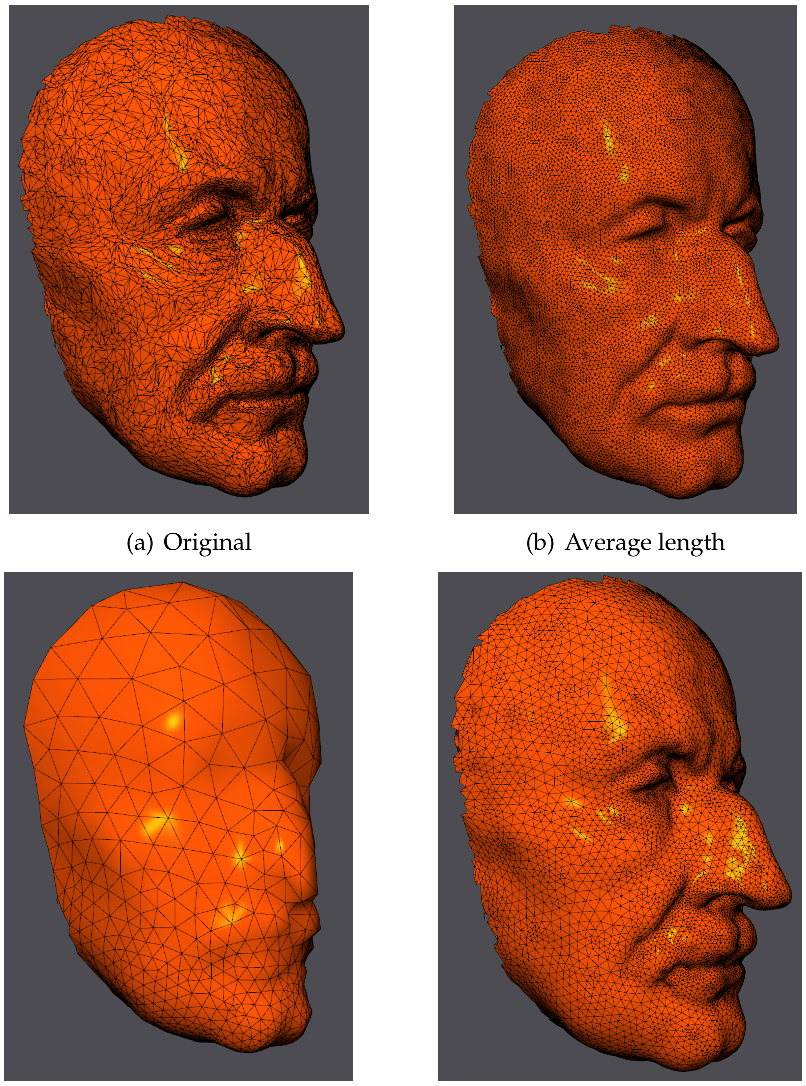
These four remeshing steps should lead to results shown on the Figure 1.

Adaptive remeshing

Reimplement the `calc_target_length()`, which calculates the vertex property `target_lengths` for adaptive remeshing. The general idea is that we encourage splits on edges where there are high curvatures, but do not split edges with lower curvature. Briefly describe which curvature measure you are using and why.

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 its mean over all the vertices equals the user specified target length.



(a) Original (b) Average length

(c) Height-based (d) Adaptive remeshing

Figure 2: Results for the different target edge length strategies.