Digital Geometry Processing
Exercise 6 - Remeshing

Handout date: 03.27.2014
Submission deadline: 04.09.2014, 23:59 h

Note

Copying of code (either from other students or from external sources) is strictly prohibited! Any violation of this rule will be reported to USC Office of Student Judicial Affairs and Community Standards.

What to hand in

A .zip compressed file renamed to "Exercise6-YourName.zip". It should contain:

- All the source code necessary to compile your application. Please, do not hand in libraries, build folders, binaries. You only need to submit `cc` and `hh` files.
- A "readme.txt" file containing a description on how you solved each exercise (use the same numbers and titles) and the encountered problems.
- Upload the archive at `http://www.dropitto.me/usc-cs599dgp` before the deadline. Password is `ididit`.

Remeshing

In this exercise you will implement a surface-based remeshing algorithm. The algorithm is described in detail in the SIGGRAPH 2007 course notes "Botsch et al.: Geometric Modeling Based on Polygonal Meshes", pages 117–121.
You can download a PDF version here `http://goo.gl/9O5Iag`. You should read the description of the algorithm.

This exercise is a simplified version of the above algorithm and is composed of four main steps:

- Split long edges
- Collapse short edges
- Flip edges to improve vertex valences
- Tangential smoothing to improve triangle quality

Framework

The project generated by CMake will produce two binaries: `remesh` and `viewer`. For remeshing use the following command line:

```
remesh 2.00 max.off output-mesh.off
```

Here `output-mesh.off` is the final remeshed mesh file name and the first number is the target edge length (in this case 2.00). You can view the results using

```
viewer output-mesh.off
```

6.1 Splitting long edges

Implement the `split_long_edges()` function so that it splits in half edges longer than the 4/3 of the target length. In order to compute the target length of an edge compute the mean of the property `vtargetlength_` of the edge's two vertices. The loop tries to split edges until there are no long edges 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 unreasonably long time.

6.2 Collapsing short edges

Complete the `collapse_short_edges()` function. First, you shouldn't consider edges connecting a boundary and a non-boundary vertex for collapse to avoid shrinking around the boundaries. Then, you should check if the edge is shorter than the 4/5 of the edges target length, computed as described in 6.1. If so, check if both of the halfedges corresponding to the edge are collapsible. If they are, you should collapse the lower valence vertex into the higher one. Otherwise collapse the halfedge which is collapsible, or don't collapse at all if both of the tests returned false. Notice that this step is not the same step which is described in the SIGGRAPH course.

Hint: The `is_collapse_ok()` function checks if a halfedge can be collapsed.

6.3 Flipping edges to improve 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 of the triangles incident 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). Only do the edge flip if it improves the valences.

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

6.4 Tangential smoothing

Implement the `tangential_relaxation()` function to improve the triangle shapes by smoothing vertices in the tangent plane of the mesh.

Similarly to a previous exercise (Smoothing), 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. For more details about tangential smoothing consult the lecture notes and the notes for the previous exercise.

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

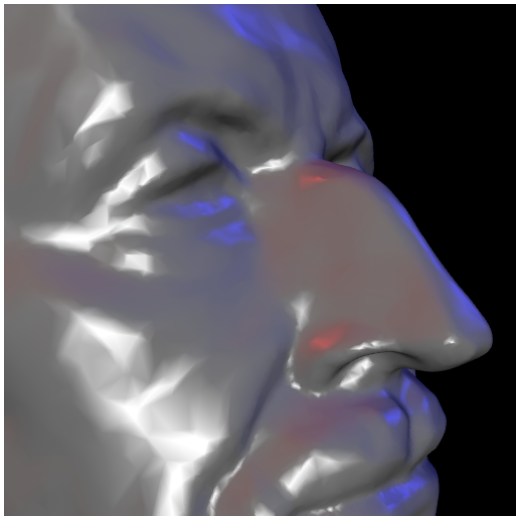
6.5 Adaptive remeshing

Reimplement the `calc_target_length()`, which calculates the vertex property `vtargetlength_` 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:

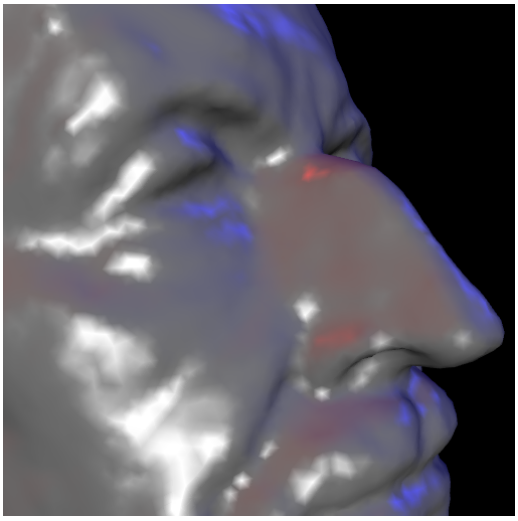
$$k_{max} = H + \sqrt{H^2 - K}$$
$$k_{min} = H - \sqrt{H^2 - K}$$

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 equals the user specified target length.

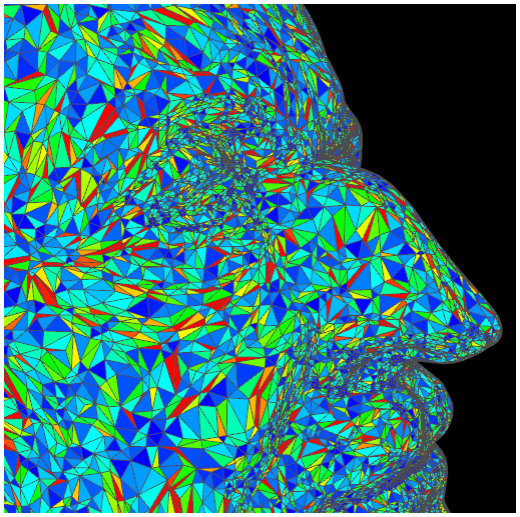
This adaptive remeshing should lead to results shown on the Figure 2.



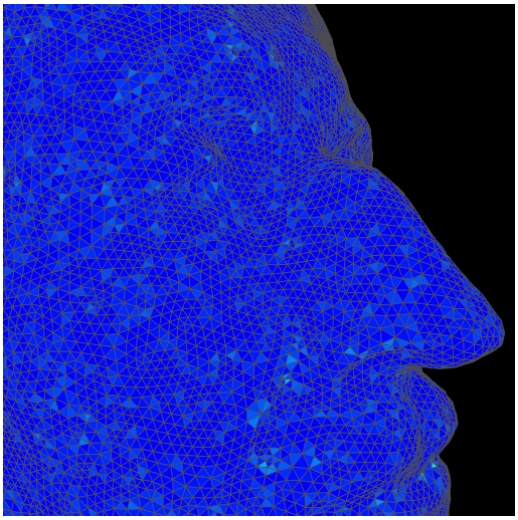
(a) Original with smooth shading



(b) Remeshed with smooth shading

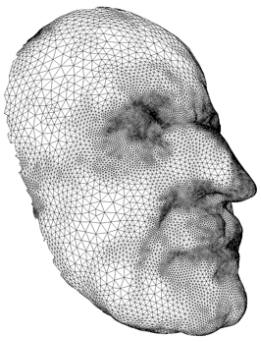


(c) Triangle qualities of original

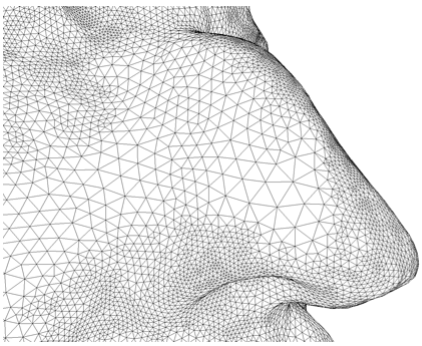


(d) Triangle qualities of remeshed

Figure 1: Meshes before and after remeshing (using the `remesh 2.00 max.off max_regular.off` command)



(a) Original mesh and its valences



(b) After adaptive remeshing

Figure 2: The adaptively remeshed max.off and a zoom-in on it's nose (using the `remesh 2.00 max.off max_regular.off` command).

6.6 For the passionate (optional)

There is a way to approximate the curvature along an edge directly. You can derive a simple formula of a curvature approximation along an edge. It uses as input the endpoint coordinates of the edge and the corresponding normals. The curvature is the inverse of the radius of the osculating circle. You can approximate the osculating circle by fitting a circle passing through the two endpoints of the edge so that the normals at the vertices are perpendicular to the circle tangent lines at that points. The inverse of the radius of this circle approximates the curvature along the edge.