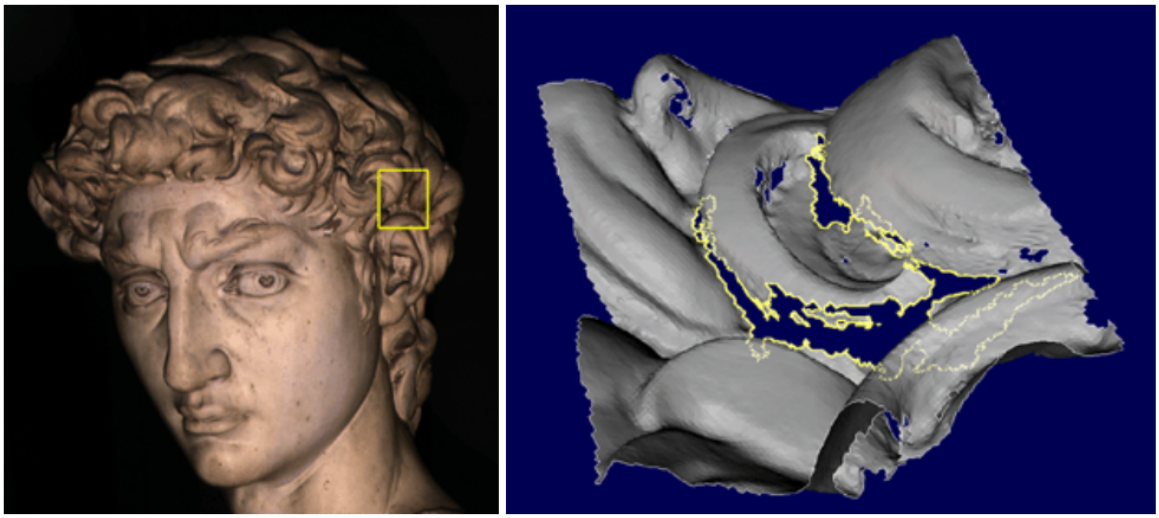
Filling Holes in Meshes

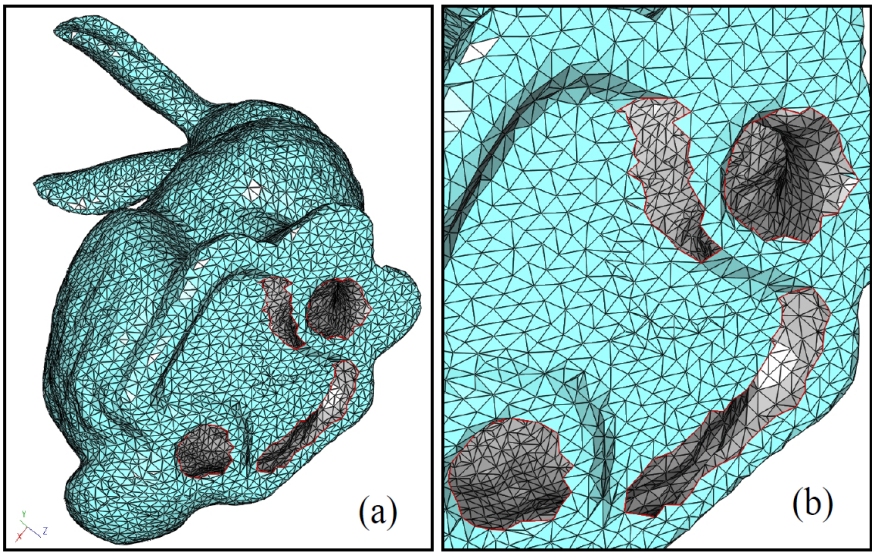
A rotation project in computer graphics

Motivation

Triangular meshes are the most common discrete representation of 3D surfaces in computer graphics. A mesh is a network of triangles that are connected via shared edges and vertices. Meshes can be created from scratch via an interactive modeling tool (e.g., Maya), but more often are reconstructed automatically from other data such as point clouds and images. Due to the imperfection of input data (e.g., occlusion in the view point where the point clouds or images are obtained), the reconstructed meshes often contain missing triangles that form “holes”. The boundary of each hole is identified by a ring of edges in the meshes that are incident to only one triangle. See two examples below.



Left: a photograph of the head of Michelangelo's David. Right: a rendered 3D model of a section of his hair (corresponding to the square in the photograph). This surface was scanned dozens of times (from many different angles), but occlusions prevented access to the deepest crevices, creating many holes (the boundary of one of them is highlighted). Credit: Marc Levoy @ Stanford.



Left: the bottom of a Bunny reconstructed from laser scans in multiple views, which contains several holes due to occlusion. Right: a close-up of the holes. Each hole's boundary is a ring of edges, each incident to only one triangle. Credit: Peter Liepa

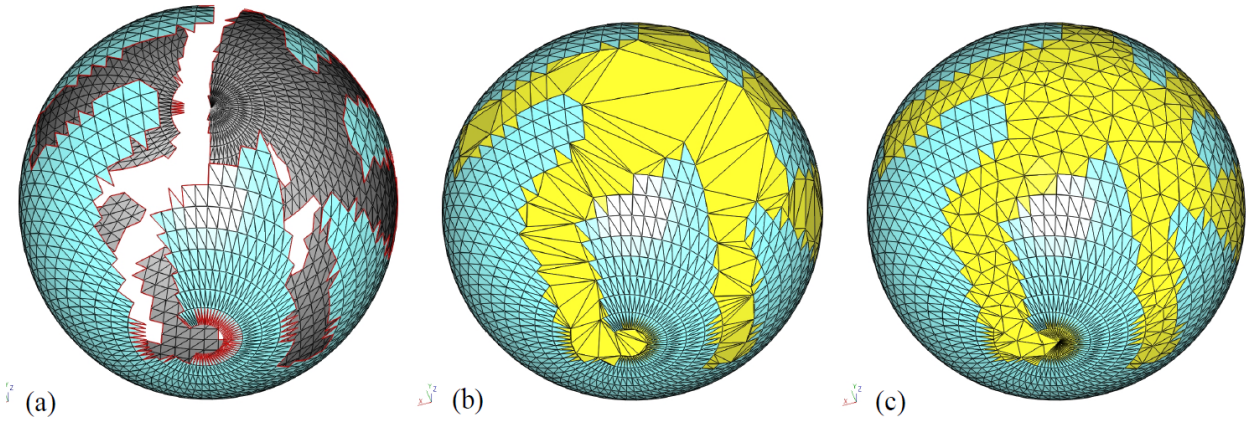
Holes in a mesh can be detrimental for many downstream applications of meshes, such as geometric calculations, physical simulations, and 3D printing. The goal of this project is to fill the holes on a given mesh.

Method

There are many hole-filling methods available. You will implement the following method by Liepa, which remains as one of the most commonly used because of its (relative) simplicity:

Peter Liepa. 2003. Filling holes in meshes. In Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing (SGP '03), 200–205. (PDF)

This method proceeds in three steps (see picture below). After identifying each hole boundary (a ring of edges; red in (a)), it first fills in an initial patch of triangles (yellow in (b)). These triangles connect only the existing vertices on the hold boundary without adding new vertices. However, the shape and size of these triangles can be very different from their surrounding triangles. Next, the algorithm refines the patch triangles by adding more vertices and changing the edge structure. Finally, it deforms the patch triangles to better match the curvature of the surrounding mesh (yellow in (c)).



(a): Input mesh with hole boundaries highlighted (red). (b): Initial hole patch after triangulation (yellow). (c): Result after refinement and fairing. Credit: Peter Liepa

Tasks

Your task is to implement Liepa's method in C++. Your program will take in an input mesh (in .off format) with holes, and output another mesh (in the same format) with the holes filled. Use MeshLab to view the input and output of your program.

I recommend you to incrementally build up your code following these steps:

1. Write a program that reads in the .off format, makes changes to the mesh structure (e.g., deletes a few triangles), and writes the output in an .off file. Check your output in MeshLab.

2. Extend your program to identify each hold boundary and perform a naive triangulation. For example, you may connect triangles from one vertex on the boundary to all other vertices on the same boundary. Or, you can add a vertex at the centroid of the hole boundary and connect all boundary vertices with that vertex.

3. Implement the triangulation algorithm in Liepa's method (Section 4).

4. Implement the refinement algorithm (Section 5).

5. Implement the fairing algorithm (Section 6).

Data

Sample input meshes (in .off format) for testing can be found here. Each mesh has one or multiple holes. To test your program, start from the simpler examples (e.g., sphere_100_*, then sphere_500_*, etc.) before moving on to larger meshes (e.g., duck and bunny).