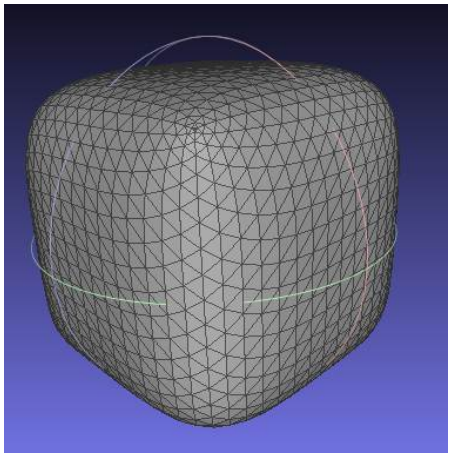
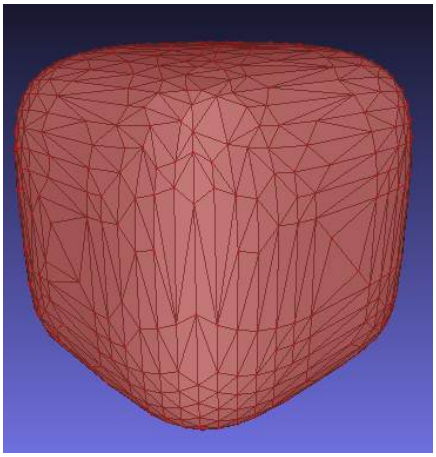
TD 4 - Mesh simplification

(Luca Castelli Aleardi)

The goal of this TD session is to perform an incremental decimation procedure for simplifying a mesh





Examples of mesh simplification based on quadric error metrics (Garland and Heckbert, 1997).

Votre opinion sur le cours:

Votre opinion sur le TD:

Submit

1. Getting started (a Processing frame for 2D and 3D rendering)

Files to download today

Here are some files to download (and extract to the main folder of your Eclipse project TD4):

- meshes.zip: examples of triangle meshes (OFF format).

Libraries to add (project /LIB):

- Jcg.jar: library for the manipulation of triangle meshes (here are the sources)
- TC.jar: input/output from files.

For the simplification and rendering of meshes (download the file TD4_src.zip):

- SurfaceMesh.java: class defining main methods for drawing the mesh
- MeshViewer.java: main drawing class, allowing to rotate the scene in 3D.
- ArcBall.java: simple class allowing to rotate the 3D scene with mouse events.
- MeshSimplification.java: abstract class defining main methods for mesh simplification (including edge contraction)
- HalfedgeContraction.java: skeleton of a simple decimation procedure (based on half-edge contractions)

Java documentation

Here is the javaDoc concerning the packages and classes used today:

- Jcg documentation.
- Processing documentation;

class MeshViewer

Class MeshViewer provides simple methods for drawing a triangle mesh in a 3D frame; it also provides a simple trackball (allowing to rotate the 3D scene with mouse events).
Method setup() allows to select the preferred simplification algorithm.

```
public void setup() {
  size(800,600,P3D);
  ArcBall arcball = new ArcBall(this);

  this.mesh=new SurfaceMesh(this, filename);
  MeshSimplification ms=new HalfedgeContraction(this.mesh.polyhedron3D);

  ms.simplify();
}
```

1. Incremental decimation based on edge contraction

1.1 Random half-edge contraction

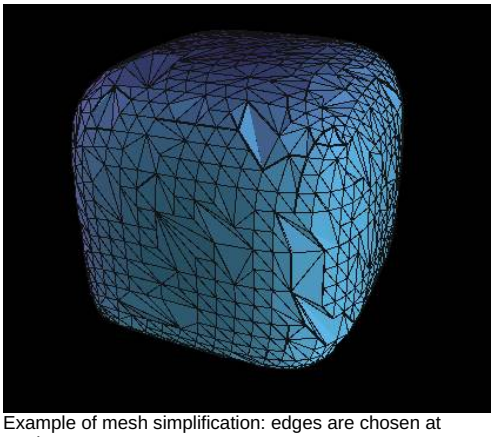
Given a triangle mesh (class Polyhedron_3<Point_3>) , we want to simplify it, by computing ***half-edge contractions***: we contract edge (u,v), by taking as new location the original position of u. Halfedges are randomly choosen.

- complete method simplify() of class HalfedgeContraction (which extends class MeshSimplification).

1.2 CHECK ILLEGAL EDGE CONTRACTIONS

Now we want to make method simplify() "safe", in order to preserve the validity of the mesh (which must remain manifold: no loops nor multiple edges).

- add a method boolean isLegal(Halfedge<Point_3> h) to class HalfedgeContraction which returns true if the contraction of edge h does preserve the combinatorial validity of the mesh.
- modify method simplify() of class HalfedgeContraction in order to perform only legal edge contractions.



Example of mesh simplification: edges are chosen at random.

1.3 EDGE CONTRACTION

Now we perform ***edge contractions***: we contract edge (u,v), and we place the new vertex, for example, at the barycenter of u and v. Halfedges are still randomly chosen.

- write class EdgeContraction (which extends class MeshSimplification).
- complete method simplify(): the new position is simply the barycenter of u and v.

- modify method simplify(): as new location, take the barycenter of vertices u, v and their neighbors.

1.4 CHOSE EDGES ACCORDING TO GEOMETRIC CRITERIA

In order to improve the quality of the simplified mesh, we choose to select edges according to a geometric criterion: we associate to each edge a cost, defined as its euclidean length.
When an edge contraction is performed, the neighboring edges involved in the contraction must be updated (their cost just changed after the edge contraction).

- modify method simplify() in order to select edges according to their length.

Remark: you can use a PriorityQueue in order to store the edges (sorted accorded to their cost)
After performing the contraction (u,v)--> z, we have to update all edges incident to the new vertex z.

2.QUADRIC ERROR METRICS (BONUS)

2.0 OPTIMAL PLACEMENT OF VERTICES BASED ON QUADRIC ERROR METRICS

As mentioned during the Lecture 4, there exists better ways for selecting edges to contract.
One solution (sketched today), consists in evaluating a local error based on quadric metrics.

- write class QuadricErrorMetrics (which extends class MeshSimplification).
- complete method simplify().