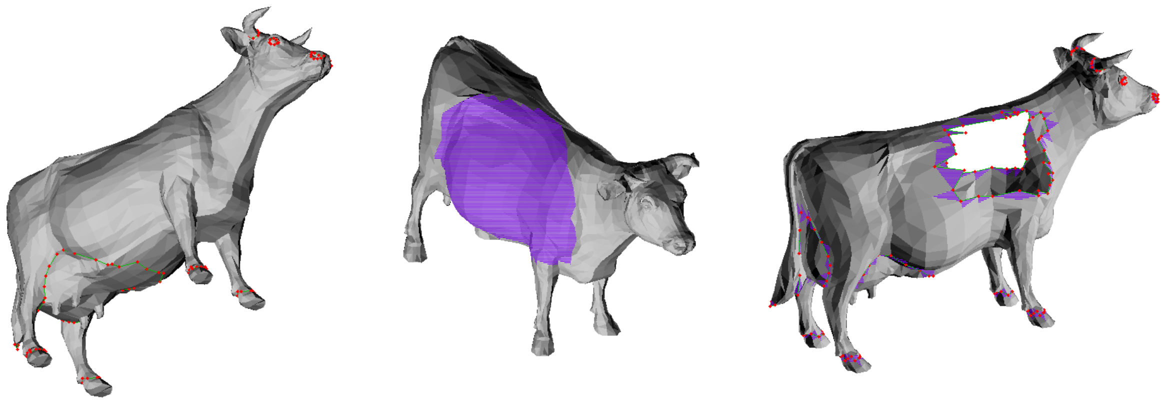
EN 292s34 / CS 220: 3D Photography and Geometry Processing

Assignment 1: Introduction to Geometry Processing

Instructor: Gabriel Taubin[*]
TA: Douglas Lanman[†]

6 February 2007



Introduction

This assignment will introduce basic geometry processing for polygon meshes. After completing this homework you should have a good understanding of the type of algorithms, data structures, and coding practices expected for this course – especially with regard to the Java framework we'll provide. In particular, you'll be implementing a variety of fundamental operations including selection, deletion, and classification of vertices, edges, and faces. The methods developed in this problem set will form the foundation for further assignments focusing on more advanced topics such as compression and filtering.

1 Getting Started: Selection Operations

Before you begin this assignment, you should first read the j3DPGP application guide [5]. Once you can successfully compile and run the application, you should begin familiarizing yourself with the following five files.

```
IfsDesktopPanels.java
IfsPanel.java
IfsPanelColors.java
IfsPanelOptions.java
IfsPanelSelection.java
```

The first four files can be used to modify the basic user interface in j3DPGP. In general, you shouldn't need to modify any of these classes. Instead, the majority of modifications will be made directly to the **IfsPanelSelection** class.

[*]taubin@mesh.brown.edu
[†]dlanman@brown.edu

To get you started, we've already implemented the *clear* and *invert* functions for vertices, edges, and faces. For example, to select all the edges in a mesh, clear the current selection and then invert (i.e., the inverse of an empty edge selection list contains all the edges in the mesh). Take a look at the **selectEdges** callback in the **IfsPanelSelection** class. Notice how the **clearEdgeIndex** and **invertEdgeIndex** methods of the **Selection** class were used to implement these operations. As you can see, the basic clear and invert operations can be implemented using methods already existing with the j3DPGP support classes.

1.1 Dilate and Erode

Now that you're familiar with the format of the **IfsPanelSelection** class, let's implement a new selection mode. Many times you want to slightly increase or decrease the current selection. We can formalize this behavior as the *dilate* and *erode* operations. For example, we can define *vertex dilation* as the operation which appends all the vertices that share an edge with a currently-selected vertex to the selection list. Similarly, we can define *face erosion* as the operation that removes all the faces that share an edge with an unselected face from the selection list. The remaining dilation and erosion operations can be defined in a similar manner.

For this problem we expect you to first define the expected behavior of each operation you implement. At a minimum, we require you to implement vertex dilation, face dilation, and face erosion. If you feel more ambitious, modify the user interface and implement the additional operations of edge dilation, edge erosion, and vertex erosion. Note that we leave the specific definition of these operations up to you. Try to be consistent and have the results provide useful modifications to the selection list. For example, if all the current faces are selected, should a face erosion do nothing or should it unselect faces containing border edges?

1.2 Connected Components

After the previous warm-up problem you should have a fairly robust set of selection behaviors. One operation we'd like to add to this set is the ability to find additional components that are connected to the current selection. For example, if you select a single vertex on `cow_npf.wrl`, the *connected vertices* button should select all the vertices connected to this point (note that the cow contains several components, which will be revealed by this operation).

As discussed in class, in order to determine connected components we'll need to partition the mesh. In the case of connected vertices, this partition will be performed on the *primal graph.* Alternatively, determining connected faces will require partitioning the *dual graph.* Note that we have provided an efficient partition routine with the **Partition** class (which uses the Union-Find data structure discussed in class [1, 2]).

For this problem we expect you to determine connected faces by implementing the **selectConnectedFaces** callback. To get you started with the problem, we've provided a complete implementation of **selectConnectedVertices**. As before, you should describe in your write-up how you define connected faces and provide a pseudocode description of your implementation. In addition, please include several screen captures for `cow_npf.wrl` and indicate the connected components.

2 Classification: Is it Boundary, Regular, or Singular?

This problem will require classifying vertices, edges, and faces as either boundary, regular, or singular. Begin by reviewing the relevant course notes [3]. Afterwards, we recommend that you first implement the classification operations for edges. Recall that a boundary, regular, or singular edge is defined has having one, two, or three or more incident faces, respectively. Notice that your classification routines should be invoked by the `SELECT_BOUNDARY`, `SELECT_REGULAR`, or `SELECT_SINGULAR` cases within the **selectVertices**, **selectEdges**, and **selectFaces** callbacks. Once you have implemented edge classification, face and vertex classification should follow directly – except for the special case of isolated singular vertices. (Think about what we discussed in class for handling this case.)

For this problem we expect you to first define the behavior of each classification operation. In general classification should be performed on the entire mesh, although you can modify this behavior as desired. As before, strive for consistency and handle as many special cases as possible. Please hand in a pseudocode description along with your solution, as well as several screenshots for `cow_npf.wrl` (similar to those provided at the beginning of this assignment). For certain cases `cow_npf.wrl` may not provide sufficient examples to demonstrate or debug your code. We'll provide additional meshes on the course website as needed.

3 Efficient Deletion for Polygonal Meshes

Up to this point all of the operations you have implemented have only modified the selection lists. For this problem you will be implementing efficient deletion for polygon meshes. To get you started we've already provided an implementation of face deletion using the **deleteSelectedFaces** callback. Unlike the previous problems, deleting vertices, faces, or edges will require updating all the associated properties of the mesh. That is, removing a vertex will require updating the coordinate indices for each face in order to rebuild the connectivity. After examining our face deletion implementation, you'll notice that you have to take care to update all the properties supported by our limited VRML'97 format – including normal and color bindings, texture coordinates, and selection lists.

For this problem we expect you to implement vertex and edge deletion using the **deleteSelectedVertices** and **deleteSelectedEdges** callbacks, respectively. Once again, begin by defining the behavior of each operation. Provide sufficient documentation of your results using pseudocode descriptions and screenshots. Finally, take care to achieve a natural interface. For example, you may find it cumbersome to maintain selection lists or other properties upon deletion. Document any assumptions or quirks specific to your implementation.

4 Topological Operations: Cutting Edges and Faces

For this problem we will add another critical operation: cutting selected edges and faces. Similar to deletion, cutting an edge or face will require updating the property bindings. We have not provided you with an implementation of either edge of face cutting, although we expect their implementation will follow the approaches you developed in the previous problem. As always, you should begin by defining the behavior of edge and face cuts. For example, cutting the set of currently selected faces should duplicate vertices and edges such that the selected region becomes one or more connected components disconnected from the unselected faces. As with classification, this will require traversing both the primal and dual graphs.

For this problem we expect you to provide a pseudocode description of your implementation, as well as supporting images documenting the results. In addition, please save several modified meshes so we can confirm your results.

Submission Instructions

You should submit clear evidence that you have successfully implemented as many features as possible. In particular, you should submit: (1) an archive named `j3DPGP-HW1-Lastname.zip` containing the modified source code, (2) a typeset document similar to this assignment handout documenting your implementations, and (3) provide modified meshes as WRL files demonstrating the algorithm output. In particular, please include modified meshes showing cutting and deletion, and describe in your write-up what modifications were performed. Final solutions should be emailed to the TA at `dlanman@brown.edu`. (Please note that we reserve the right to request a 15 minute demonstration of your implementation if the submitted documentation is insufficient to verify correctness.)

Finally, note that the best submissions will be asked to present a short demo in class and answer student questions regarding their solution. Remember that this assignment will lay the foundation for future problems sets and possible final projects – devote the necessary time to familiarize yourself with the j3DPGP framework.

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* The MIT Press and McGraw-Hill, 2001.

[2] Robert Endre Tarjan. *Data Structures and Network Algorithms.* CBMS-NSF Regional Conference Series in Applied Mathematics, 1983.

[3] Gabriel Taubin. Course notes on surfaces representations. `http://mesh.brown.edu/3dpgp/pdfs/01-SurfReps.pdf`.

[4] Gabriel Taubin. Course notes on the half-edge data structure. `http://mesh.brown.edu/3dpgp/pdfs/02-HalfEdge.pdf`.

[5] Gabriel Taubin. The j3DPGP application. `http://mesh.brown.edu/3dpgp`.