To Do

Assignment 1, Due Oct 7
- Start reading and working on it now.
- This lecture starts discussing early parts of assignment, followed by mesh simplification

Outline
- Basic assignment overview
- Detailed discussion of mesh simplification
- Progressive meshes
- Quadric error metrics

Assignment Overview
- Implement complete system for mesh simplification
- Plus maybe progressive meshes
  - Possibly challenging assignment: start very early and proceed in incremental fashion
  - Choice of data structure for meshes is the key (read the assignment)
  - This involves fairly recent work. No one answer
    - Think about the best way of proceeding, use creativity

Mesh Viewer (3.1)
Deliberately, no skeleton code for assignment
- Think about and implement full system from scratch
First step: Mesh viewer
- Read meshes (in simple OFF file format), view them
- Can reuse code from 184 (if stuck, try hw1 in 184)
- Shading: must average face normals per vertex (this may give you a start in implementing a mesh data structure)
- Debugging modes for shading (color each triangle separately with an individual color)
Software Design
- Define mesh class with display method etc.
- Use C++ STL data structures where appropriate (see assn)

Mesh Connectivity (3.2)
Build up mesh connectivity data structure
- Input is vertices and faces from input file
Goal is to do edge collapses in constant time
- No iteration over whole mesh
- Most of mesh unchanged
- Important questions for your data structure to answer: “What vertices neighbor my current vertex?” and “What faces neighbor my current vertex”
- Think about updating your data structure. Collapsing an edge may require more than just the edge itself. You must update every vertex or face that has changed
Mesh Decimation (edge collapse)

- Can you handle this correctly and efficiently? Debugging examples in testpatch and plane (do these first)

Mesh Data Structure Hints

- Simplest (I think): Faces store constituent vertices [indexed face set as in OFF], vertices store adjacent faces (how do you create vertex-face adjacency?)
- To simplify, first create new vertex v. Adjacent faces are those adjacent to v0 or v1
- For each of those faces, update to point to v instead of v0 or v1

Mesh Decimation (edge collapse 3.3)

- Create new vertex v (based on appropriate rule)
- Find all faces/edges neighbor vertex v1 (such as A)
- Change them to use v instead of v1. Do the same for v0
- Depend on data structure, you need to fix all faces, edges

Mesh Decimation (edge collapse)

- Find faces neighboring edge v0-v1 (such as X)
- Remove from mesh
  - This may involve updating face/vertex adjacency relationships etc.
  - E.g. what is adjacency for v (faces adjacent to vertex?)
  - Are other vertices affected in terms of adjacent faces?
- Worry about triangle fins (extra credit, not discussed)
Mesh Data Structure Hints

- With indexed face set plus vertex to face adjacency, removing a face should just work (remember to delete face from vertex adjacency lists)
- In general, winged edge, half-edge may be (slightly) more efficient, but also harder to implement
- Ultimately, your choice and work out the details
- Good luck!!

Mesh Decimation (edge collapse 3.3)

- Find faces neighboring edge v0-v1 (such as X)
- Union of adjacent faces to vertex v0 and vertex v1
- Update adjacency lists
  - For all vertices, remove that face from their adjacency list
  - Remove face from mesh

Implementation

- Tricky
- When you remove something, need to update appropriately
- Work out on paper first (e.g. indexed face set plus adjacent faces for each vertex)
- Depends on choice of data structure (pick easy to do)
- Start with simple debugging cases (make sure not just that it looks right, but all adjacencies remain correct)

Outline

- Basic assignment overview
- Detailed discussion of mesh simplification
- Progressive meshes
- Quadric error metrics

Successive Edge Collapses

- We have discussed one edge collapse, how to do that
- In practice, sequence of edge collapses applied
- Order etc. based on some metric (later in lecture)
- So, we gradually reduce complexity of model
- Progressive meshes is opposite: gradually increase complexity

Appearance Preserving

- Caltech & Stanford Graphics Labs and Jonathan Cohen
Progressive Meshes (3.5)

- Write edge collapses to file
- Read in file and invert order
- Key idea is **vertex-split** (opposite of edge-collapse)
- Include some control to make model coarser/finer

  - E.g. Hoppe geomorph demo

GeoMorph (may not play in ppt)

- Hoppe geomorph demo

Vertex splits

- Can you handle this correctly and efficiently? Debugging examples in testpatch and plane (do these first)

Implementation

- Tricky
- What info do you need to add something?
- Work out on paper first (e.g. indexed face set plus adjacent faces for each vertex)
- Start with simple debugging cases (make sure not just that it looks right, but all adjacencies remain correct)

View-Dependent Simplification

- Simplify dynamically according to viewpoint
  - Visibility
  - Silhouettes
  - Lighting

Outline

- Basic assignment overview
- Detailed discussion of mesh simplification
- Progressive meshes
- Quadric error metrics
Quadric Error Metrics
- Garland & Heckbert, SIGGRAPH 97
- Greedy decimation algorithm
- Pair collapse (allow edge + non-edge collapses)
- Quadric error metrics:
  - Evaluate potential collapses
  - Determine optimal new vertex locations

Background: Computing Planes
- Each triangle in mesh has associated plane
  \[ ax + by + cz + d = 0 \]
- For a triangle, find its (normalized) normal using cross products
  \[ \vec{n} = \frac{\vec{A} \times \vec{C}}{\| \vec{A} \times \vec{C} \|} \]
  \[ \vec{n} \cdot \vec{A} = \vec{n} \cdot \vec{C} = 0 \]
- Plane equation?
  \[ \vec{n} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}, \quad d = -\vec{A} \cdot \vec{n} \]

Quadric Error Metrics
- Based on point-to-plane distance
- Better quality than point-to-point

Quadric Error Metrics
- Sum of squared distances from vertex to planes:
  \[ \Delta = \sum_p \text{Dist}(\vec{v}, p)^2 \]
  \[ \vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad p = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \]
  \[ \text{Dist}(\vec{v}, p) = ax + by + cz + d = \vec{p} \cdot \vec{v} \]

Quadric Error Metrics
- Common mathematical trick: quadratic form = symmetric matrix \( Q \) multiplied twice by a vector
- Initially, distance to all planes 0, net is 0 for all verts

Using Quadrics
- Approximate error of edge collapses
  - Each vertex \( v \) has associated quadric \( Q_v \)
  - Error of collapsing \( v_1 \) and \( v_2 \) to \( v' \) is \( v'^\top Q_1 v' + v'^\top Q_2 v' \)
  - Quadric for new vertex \( v' \) is \( Q' = Q_1 + Q_2 \)
Using Quadrics

- Find optimal location $v^*$ after collapse:

$$Q = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ q_{14} & q_{24} & q_{34} & q_{44} \end{bmatrix}$$

$$\min_{v^*} v^T Q v^* : \frac{\partial}{\partial v} v^T Q v^* = 0$$

Results

- Ellipsoids: iso-error surfaces
- Smaller ellipsoid = greater error for a given motion
- Lower error for motion parallel to surface
- Lower error in flat regions than at corners
- Elongated in “cylindrical” regions

Quadric Visualization

Summary

- First, implement basic mesh simplification on one edge
- Helps to have right data structure
  - Tricky since needs to be efficient and properly update
- Then, implement quadric error metrics
  - Tricky; we will spend most of another lecture on this
  - Put edge collapses in priority queue
  - Problem is that when you do one, you have to update all the neighbors as well (just as for standard edge collapse)
  - And re-insert in queue (use appropriate data structure)