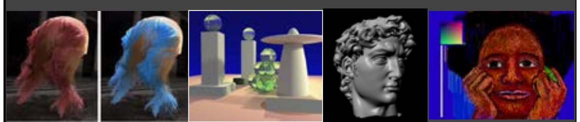
## Advanced Computer Graphics

CSE 163 [Spring 2018], Lecture 7

Ravi Ramamoorthi

http://www.cs.ucsd.edu/~ravir



## To Do

- Assignment 1, Due Apr 27
  - Any last minute issues or difficulties?
- Starting Geometry Processing
  - Assignment 2 due May 18
  - This lecture starts discussing relevant content
  - Please START EARLY. Can do most after this week
  - Contact us for difficulties, help finding partners etc.

## Motivation

- A polygon mesh is a collection of triangles
- We want to do operations on these triangles
  - E.g. walk across the mesh for simplification
  - Display for rendering
  - Computational geometry
- Best representations (mesh data structures)?
  - Compactness
  - Generality
  - Simplicity for computations
  - Efficiency

## Mesh Data Structures

Desirable Characteristics 1

- Generality – from most general to least
  - Polygon soup
  - Only triangles
  - 2-manifold: ≤ 2 triangles per edge
  - Orientable: consistent CW / CCW winding
  - Closed: no boundary
- Compact storage

## Mesh Data Structures

Desirable characteristics 2

- Efficient support for operations:
  - Given face, find its vertices
  - Given vertex, find faces touching it
  - Given face, find neighboring faces
  - Given vertex, find neighboring vertices
  - Given edge, find vertices and faces it touches
- These are adjacency operations important in mesh simplification (homework), many other applications

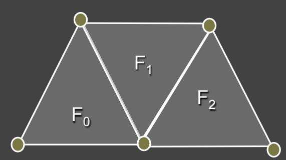
## Outline

- *Independent faces*
- *Indexed face set*
- Adjacency lists
- Winged-edge
- Half-edge

Overview of mesh decimation and simplification

## Independent Faces

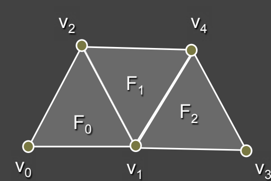
Faces list vertex coordinates
- Redundant vertices
- No topology information




Face Table
$F_0$: $(x_0,y_0,z_0)$, $(x_1,y_1,z_1)$, $(x_2,y_2,z_2)$
$F_1$: $(x_3,y_3,z_3)$, $(x_4,y_4,z_4)$, $(x_5,y_5,z_5)$
$F_2$: $(x_6,y_6,z_6)$, $(x_7,y_7,z_7)$, $(x_8,y_8,z_8)$

## Indexed Face Set

- Faces list vertex references – "shared vertices"
- Commonly used (e.g. OFF file format itself)
- Augmented versions simple for mesh processing



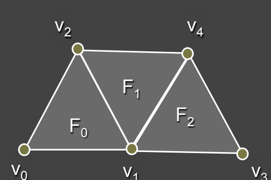

Vertex Table
$v_0$: $(x_0,y_0,z_0)$
$v_1$: $(x_1,y_1,z_1)$
$v_2$: $(x_2,y_2,z_2)$
$v_3$: $(x_3,y_3,z_3)$
$v_4$: $(x_4,y_4,z_4)$

Face Table
$F_0$: 0, 1, 2
$F_1$: 1, 4, 2
$F_2$: 1, 3, 4

Note CCW ordering

## Indexed Face Set

- Storage efficiency?
- Which operations supported in O(1) time?




Vertex Table
$v_0$: $(x_0,y_0,z_0)$
$v_1$: $(x_1,y_1,z_1)$
$v_2$: $(x_2,y_2,z_2)$
$v_3$: $(x_3,y_3,z_3)$
$v_4$: $(x_4,y_4,z_4)$

Face Table
$F_0$: 0, 1, 2
$F_1$: 1, 4, 2
$F_2$: 1, 3, 4

Note CCW ordering

## Efficient Algorithm Design

- Can *sometimes* design algorithms to compensate for operations not supported by data structures
- Example: per-vertex normals
  - Average normal of faces touching each vertex
  - With indexed face set, vertex → face is $O(n)$
  - Naive algorithm for all vertices: $O(n^2)$
  - Can you think of an $O(n)$ algorithm?

## Efficient Algorithm Design

- Can *sometimes* design algorithms to compensate for operations not supported by data structures
- Example: per-vertex normals
  - Average normal of faces touching each vertex
  - With indexed face set, vertex → face is $O(n)$
  - Naive algorithm for all vertices: $O(n^2)$
  - Can you think of an $O(n)$ algorithm?
- Useful to augment with vertex → face adjacency
  - For all vertices, find adjacent faces as well
  - Can be implemented while simply looping over faces

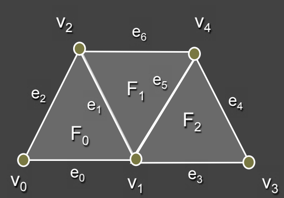
## Outline

- Independent faces
- Indexed face set
- *Adjacency lists*
- Winged-edge
- Half-edge

Overview of mesh decimation and simplification

## Full Adjacency Lists

- Store all vertex, face, and edge adjacencies




Edge Adjacency Table
$e_0$: $v_0$, $v_1$; $F_0$,ø; ø,$e_2$,$e_1$,ø
$e_1$: $v_1$,$v_2$; $F_0$,$F_1$; $e_5$,$e_0$,$e_2$,$e_6$
⋮

Face Adjacency Table
$F_0$: $v_0$,$v_1$,$v_2$; $F_1$,ø,ø; $e_0$,$e_2$,$e_1$
$F_1$: $v_1$,$v_4$,$v_2$; ø,$F_0$,$F_2$; $e_6$,$e_1$,$e_5$
$F_2$: $v_1$,$v_3$,$v_4$; ø,$F_1$,ø; $e_4$,$e_5$,$e_3$

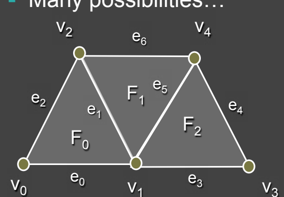
Vertex Adjacency Table
$v_0$: $v_1$,$v_2$; $F_0$; $e_0$,$e_2$
$v_1$: $v_3$,$v_4$,$v_2$,$v_0$; $F_2$,$F_1$,$F_0$; $e_3$,$e_5$,$e_1$,$e_0$
⋮

## Full adjacency: Issues

- Garland and Heckbert claim they do this
- Easy to find stuff
- Issue is storage
- And updating everything once you do something like an edge collapse for mesh simplification
- I recommend you implement something simpler (like indexed face set plus vertex to face adjacency)

## Partial Adjacency Lists

- Store some adjacencies, use to derive others
- Many possibilities…



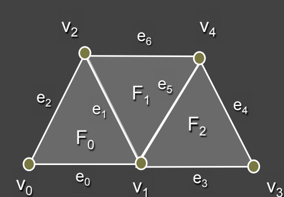

Edge Adjacency Table
$e_0$: $v_0$, $v_1$; $F_0$,ø; ø,$e_2$,$e_1$,ø
$e_1$: $v_1$,$v_2$; $F_0$,$F_1$; $e_5$,$e_0$,$e_2$,$e_6$
⋮

Face Adjacency Table
$F_0$: $v_0$,$v_1$,$v_2$; $F_1$,ø,ø; $e_0$,$e_2$,$e_1$
$F_1$: $v_1$,$v_4$,$v_2$; ø,$F_0$,$F_2$; $e_6$,$e_1$,$e_5$
$F_2$: $v_1$,$v_3$,$v_4$; ø,$F_1$,ø; $e_4$,$e_5$,$e_3$

Vertex Adjacency Table
$v_0$: $v_1$,$v_2$; $F_0$; $e_0$,$e_2$
$v_1$: $v_3$,$v_4$,$v_2$,$v_0$; $F_2$,$F_1$,$F_0$; $e_3$,$e_5$,$e_1$,$e_0$
⋮

## Partial Adjacency Lists

- Some combinations only make sense for closed manifolds




Edge Adjacency Table
$e_0$: $v_0$, $v_1$; $F_0$,ø; ø,$e_2$,$e_1$,ø
$e_1$: $v_1$,$v_2$; $F_0$,$F_1$; $e_5$,$e_0$,$e_2$,$e_6$
⋮

Face Adjacency Table
$F_0$: $v_0$,$v_1$,$v_2$; $F_1$,ø,ø; $e_0$,$e_2$,$e_1$
$F_1$: $v_1$,$v_4$,$v_2$; ø,$F_0$,$F_2$; $e_6$,$e_1$,$e_5$
$F_2$: $v_1$,$v_3$,$v_4$; ø,$F_1$,ø; $e_4$,$e_5$,$e_3$

Vertex Adjacency Table
$v_0$: $v_1$,$v_2$; $F_0$; $e_0$,$e_2$
$v_1$: $v_3$,$v_4$,$v_2$,$v_0$; $F_2$,$F_1$,$F_0$; $e_3$,$e_5$,$e_1$,$e_0$
⋮

## Outline

- Independent faces
- Indexed face set
- Adjacency lists
- *Winged-edge*
- *Half-edge*

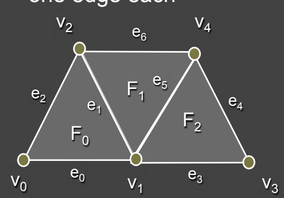
Overview of mesh decimation and simplification

## Winged, Half Edge Representations

- Idea is to associate information with edges
- Compact Storage
- Many operations efficient
- Allow one to walk around mesh
- Usually general for arbitrary polygons (not triangles)
- But implementations can be complex with special cases relative to simple indexed face set++ or partial adjacency table

## Winged Edge

- Most data stored at edges
- Vertices, faces point to one edge each




Edge Adjacency Table

$e_0$: $v_0$, $v_1$; $F_0$, ø; ø,$e_2$,$e_1$, ø
$e_1$: $v_1$,$v_2$; $F_0$,$F_1$; $e_5$,$e_0$,$e_2$,$e_6$
⋮

Face Adjacency Table

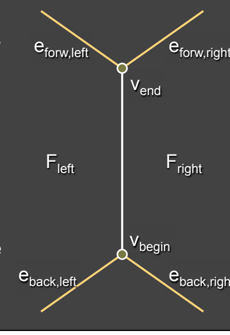
$F_0$: $v_0$,$v_1$,$v_2$; $F_1$, ø, ø; $e_0$,$e_2$,$e_1$
$F_1$: $v_1$,$v_4$,$v_2$; ø,$F_0$,$F_2$; $e_6$,$e_1$,$e_5$
$F_2$: $v_1$,$v_3$,$v_4$; ø,$F_1$, ø; $e_4$,$e_5$,$e_3$

Vertex Adjacency Table

$v_0$: $v_1$,$v_2$; $F_0$; $e_0$,$e_2$
$v_1$: $v_3$,$v_4$,$v_2$,$v_0$; $F_2$,$F_1$,$F_0$; $e_3$,$e_5$,$e_1$,$e_0$
⋮

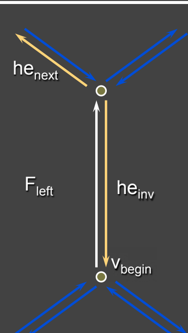
## Winged Edge

- Each edge stores 2 vertices, 2 faces, 4 edges – fixed size
- Enough information to completely "walk around" faces or vertices
- Think how to implement
  - Walking around vertex
  - Finding neighborhood of face
  - Other ops for simplification




## Half Edge

- Instead of single edge, 2 directed "half edges"
- Makes some operations more efficient
- Walk around face very easily (each face need only store one pointer)




## HalfEdge Data Structure (example)

```
class HalfEdge {  // Only one example, some critical functions
  public:
     HalfEdgeIter next;    // points to the next halfedge around the current face
     HalfEdgeIter flip;    // points to the other halfedge associated with this edge
     VertexIter vertex;    // points to the vertex at the "tail" of this halfedge
     EdgeIter edge;        // points to the edge associated with this halfedge
     FaceIter face;        // points to the face containing this halfedge
     bool onBoundary;      // true if this halfedge is contained in a boundary
                           // loop; false otherwise
     } ;
```

From Keenan Crane Geometry Processing code

https://github.com/dgpdec/course but write your own version

## HalfEdge Walk Around Faces

```
int Vertex :: valence( void ) const {   // returns the number of incident faces
    int n = 0;
    HalfEdgeCIter h = he;    // Start loop with half-edge for that vertex
    do {
      n++;                   // Increment Valence.  Other operations similarly
                             // For area,  A += h -> face -> area() ;
      h = h->flip->next;     // Next Face.  Why does this work?
    }
    while( h != he );        // Stop when loop is complete.  How does this work?
    return n;
  }
```

From Keenan Crane Geometry Processing code

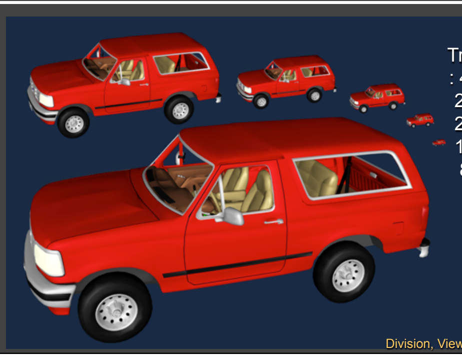
https://github.com/dgpdec/course but write your own version

## Outline

- Independent faces
- Indexed face set
- Adjacency lists
- Winged-edge
- Half-edge

*Overview of mesh decimation and simplification*

## Mesh Decimation



Triangles
: 41,855
27,970
20,922
12,939
8,385
4,766

Division, Viewpoint, Cohen

## Mesh Decimation

- Reduce number of polygons
  - Less storage
  - Faster rendering
  - Simpler manipulation
- Desirable properties
  - Generality
  - Efficiency
  - Produces “good” approximation

Michelangelo's St. Matthew
Original model: ~400M polygons

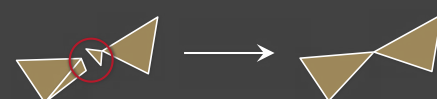
## Primitive Operations

Simplify model a bit at a time by removing a few faces
- Repeated to simplify whole mesh

Types of operations
- Vertex cluster
- Vertex remove
- Edge collapse (main operation used in assignment)

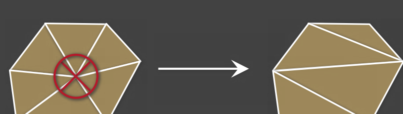
## Vertex Cluster

- Method
  - Merge vertices based on proximity
  - Triangles with repeated vertices can collapse to edges or points
- Properties
  - General and robust
  - Can be unattractive if results in topology change



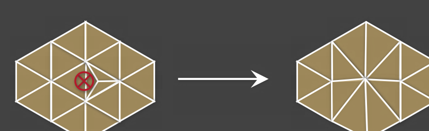
## Vertex Remove

- Method
  - Remove vertex and adjacent faces
  - Fill hole with new triangles (reduction of 2)
- Properties
  - Requires manifold surface, preserves topology
  - Typically more attractive
  - Filling hole well not always easy



## Edge Collapse

- Method
  - Merge two edge vertices to one
  - Delete degenerate triangles
- Properties
  - Special case of vertex cluster
  - Allows smooth transition
  - Can change topology



## Mesh Decimation/Simplification

Typical: greedy algorithm

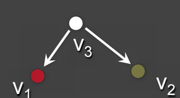
- Measure error of possible “simple” operations (primarily edge collapses)
- Place operations in queue according to error
- Perform operations in queue successively (depending on how much you want to simplify model)
- After each operation, re-evaluate error metrics

## Geometric Error Metrics

- Motivation
  - Promote accurate 3D shape preservation
  - Preserve screen-space silhouettes and pixel coverage
- Types
  - Vertex-Vertex Distance
  - Vertex-Plane Distance
  - Point-Surface Distance
  - Surface-Surface Distance

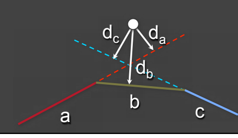
## Vertex-Vertex Distance

- E = max(|v3−v1|, |v3−v2|)
- Appropriate during topology changes
  - Rossignac and Borrel 93
  - Luebke and Erikson 97
- Loose for topology-preserving collapses



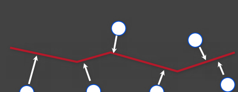

## Vertex-Plane Distance

- Store set of planes with each vertex
  - Error based on distance from vertex to planes
  - When vertices are merged, merge sets
- Ronfard and Rossignac 96
  - Store plane sets, compute max distance
- Error Quadrics – Garland and Heckbert 97
  - Store quadric form, compute sum of squared distances




## Point-Surface Distance

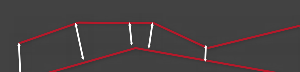
- For each original vertex, find closest point on simplified surface
- Compute sum of squared distances



## Surface-Surface Distance

Compute or approximate maximum distance between input and simplified surfaces

- Tolerance Volumes - Guéziec 96
- Simplification Envelopes - Cohen/Varshney 96
- Hausdorff Distance - Klein 96
- Mapping Distance - Bajaj/Schikore 96, Cohen et al. 97



## Geometric Error Observations

- Vertex-vertex and vertex-plane distance
  - Fast
  - Low error in practice, but not guaranteed by metric
- Surface-surface distance
  - Required for guaranteed error bounds


vertex-vertex ≠ surface-surface

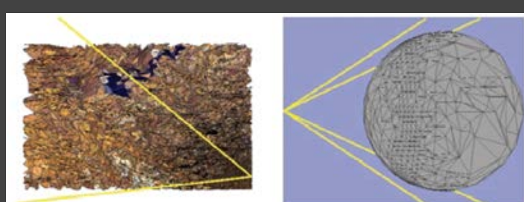
## Mesh Simplification

Advanced Considerations

- Type of input mesh?
- Modifies topology?
- Continuous LOD?
- Speed vs. quality?

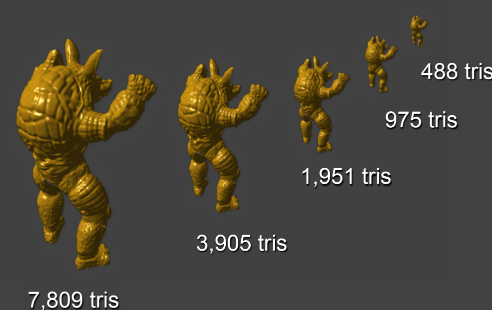
## View-Dependent Simplification

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



Hoppe

## Appearance Preserving




Caltech & Stanford Graphics Labs and Jonathan Cohen

## Summary

- Many mesh data structures
  - Compact storage vs ease, efficiency of use
  - How fast and easy are key operations
- Mesh simplification
  - Reduce size of mesh in efficient quality-preserving way
  - Based on edge collapses mainly
- Choose appropriate mesh data structure
  - Efficient to update, edge-collapses are local
- Fairly modern ideas (last ~20 years)
  - Think about some of it yourself, see papers given out
  - We will cover simplification, quadric metrics next