Simplification
&
Level Of Detail

CS 184 Spring 2004

Outline

- Basic Level of Detail (LOD) issues
- Simplification Algorithms
- View-Dependent LOD

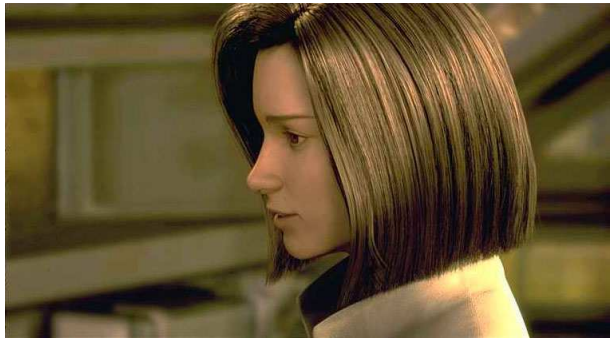
Motivation

Fact:

- Realistic scenes require complex models with huge number of polygons.

Problem:

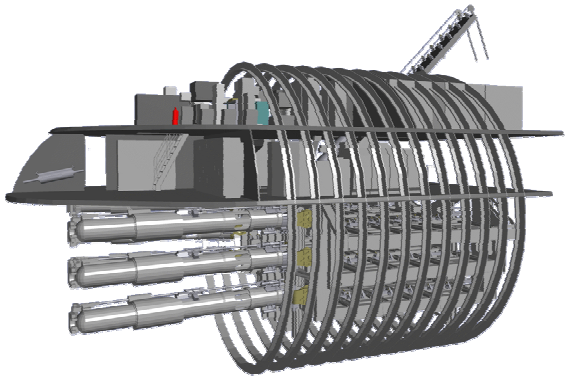
- How can we visualize them efficiently (at interactive rates)?



Aki's Hair

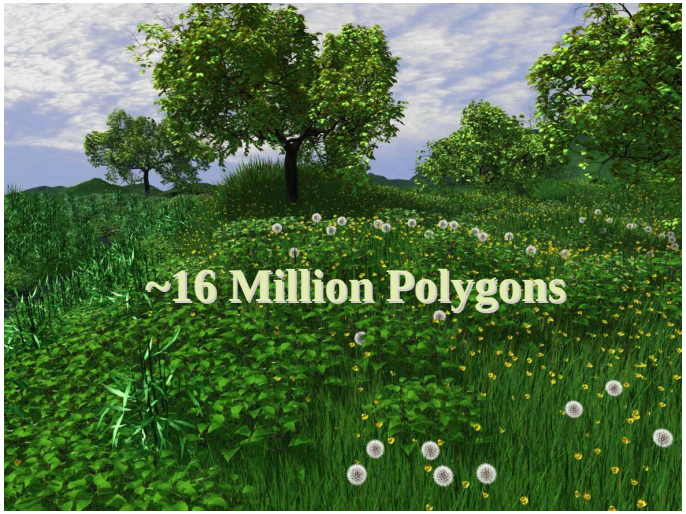
~60,000 Strands!

Submarine Torpedo Room



General Dynamics, Electric Boat Div.

~700,000 Polygons



Deussen et al: *Realistic Modeling of Plant Ecosystems*

David

~56 Million Polygons

Courtesy Digital Michelangelo Project

St. Matthew

~372 Million Polygons

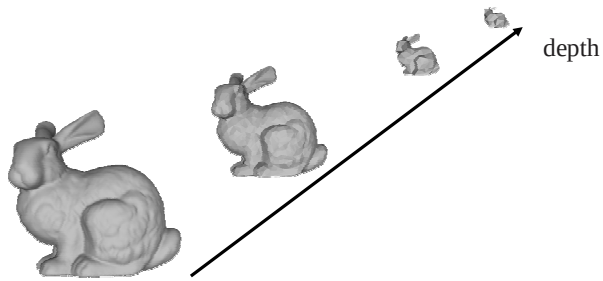
Basic Solution: Discrete LOD

Step 1: Simplify the model at different levels of detail

69,451 polys | 2,502 polys | 251 polys | 76 polys

Discrete LOD (contin'd)

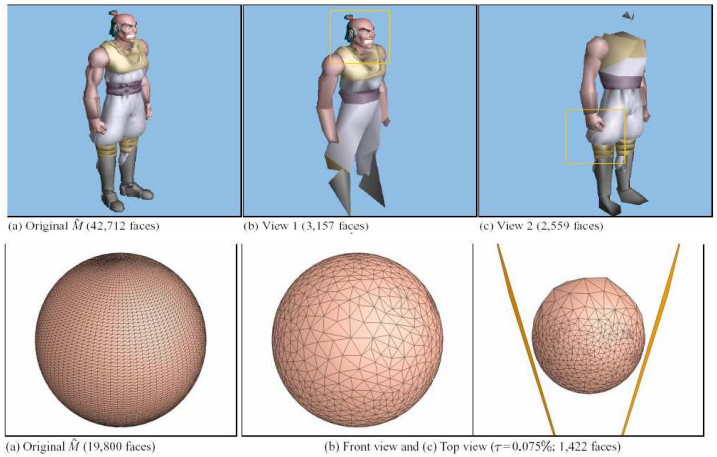
Step 2: Use low detail for objects that are small or far away



Problems

- What if:
  - The object is big, i.e some parts of it are close to the viewer and others are far away (e.g. terrains)
  - Only a portion of it is in the view frustum
  - Some parts that are in the view frustum are back facing
- Need to be able to change LOD on the fly depending on the view (View-Dependent LOD)

View-Dependent LOD Examples



(a) Original $\hat{M}$ (42,712 faces) (b) View 1 (3,157 faces) (c) View 2 (2,559 faces)

(a) Original $\hat{M}$ (19,800 faces) (b) Front view and (c) Top view (τ=0.075%; 1,422 faces)

Questions we need to answer:

1. How do we simplify a complex object?
2. How can we generate view dependent LOD efficiently?

Simplification

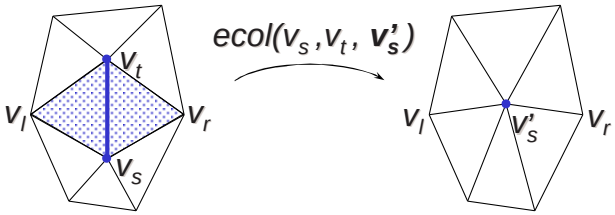
- Basic Goal:

  Decrease number of vertices (and/or faces) while staying *close* (by some *quality* metric) to the original shape

- Common Techniques:
  - Vertex Decimation
    - Select a vertex for removal, remove all adjacent faces and re-triangulate the hole using less number of triangles
  - Vertex Clustering
    - Divide the bounding box of the object into a grid, cluster every vertex in a cell into a single vertex and update the faces
  - Iterative Edge and/or Vertex Pair Collapse
    - Choose two vertices that share an edge or that are "close enough", collapse them into a single vertex and update the faces.

Edge Collapse

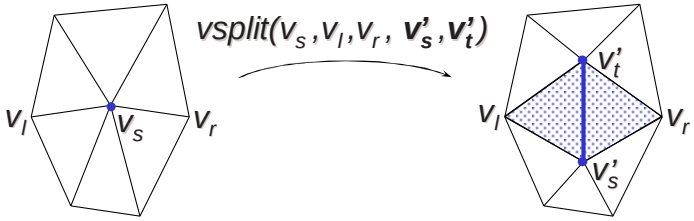
- Collapse edge v_t-v_s by contracting v_t and v_s into v'_s



- Beware of face inversions! Not every ecol is valid.

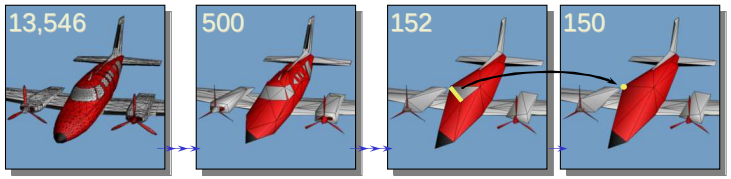
Vertex Split

- Inverse of edge collapse



Progressive meshes

- Start with the original (fine) mesh, denoted as $\hat{M}$
- Keep collapsing edges until you have a coarse enough mesh, denoted as M^0



$$\hat{M}=M^n \xrightarrow{ecol_{n-1}} \cdots M^{175} \cdots \xrightarrow{ecol_i} M^1 \xrightarrow{ecol_0} M^0$$

Progressive meshes

- Basic representation is a list of ecols and/or vsplits:

$$\hat{M}=M^n \underset{vsplit_{n-1}}{\overset{ecol_{n-1}}{\rightleftarrows}} \cdots \underset{vsplit_1}{\overset{ecol_1}{\rightleftarrows}} M^1 \underset{vsplit_0}{\overset{ecol_0}{\rightleftarrows}} M^0$$

- Important Questions:
  1. Which edge to choose at each step?
  2. What is the location of v'_s after an edge collapse?

Choosing Edges

- Random
- Shortest Edge
- One that causes the minimum error, i.e minimum deviation from original mesh by some metric
  - Hausdorff distance

$$h(A,B) = \max_{a\in A} \min_{b\in B} \|a-b\| \qquad \text{One-sided}$$

$$H(A,B) = \max(h(A,B), h(B,A)) \qquad \text{Two-sided}$$

  - Vertex-Vertex distance
  - Vertex-Plane distance (e.g. Quadric Error Metric)

Quadric Error Metric

- Every vertex *v* in the original mesh has a set of faces incident on it.
- Each one of those faces defines a plane

$$p: Ax + By + Cz + D = 0$$

- Define the error *e* at a vertex *v* to be the sum of the squared distances to the planes associated with *v*:

$$e(v) = \sum_{p\in planes(v)} dist(v,p)^2$$

Quadric Error Metric

- Distance of a vertex $v = [v_x\ v_y\ v_z\ 1]^T$ to a plane $p = [A\ B\ C\ D]^T$ is simply:

$$dist(v,p) = p^T v$$

- Squared distance?

$$dist(v,p)^2 = (p^T v)^T (p^T v) = v^T (p\, p^T)\, v$$

- Error e(v) can be reformulated as:

$$e(v) = \sum_{p\in planes(v)} v^T (pp^T) v = v^T \left[\sum_{p\in planes(v)} (pp^T) \right] v = v^T \mathbf{Q}\, v$$

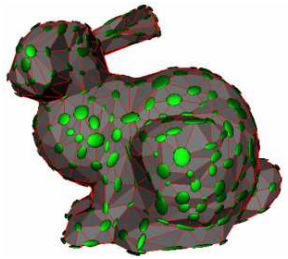
What is $v^T Q v = \varepsilon$?

- For an arbitrary v=[x y z 1]:

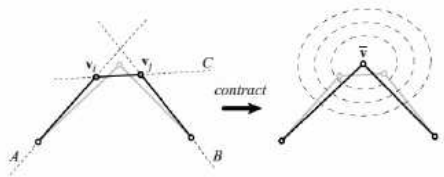
$v^T Q v = q_{11} x^2 + 2 q_{12} xy + 2 q_{13} xz + 2 q_{14} x + q_{21} y^2 + \ldots + q_{44} = \varepsilon$

So, it defines a quadric surface (usually an ellipsoid)

You can move v in this surface freely without deviating from the original surface by more than ε



Cost of an Edge Collapse



- What is the cost of collapsing v_i and v_j into $\bar{v}$?

$$\text{cost}(ecol(v_i, v_j, \bar{v})) = e(\bar{v}) = \bar{v}^T \bar{\mathbf{Q}} \bar{v}$$

where $\bar{\mathbf{Q}} = (\mathbf{Q}_i + \mathbf{Q}_j)$

Simplification Algorithm

1. Compute the **Q** matrices for all initial vertices
2. Select all valid edges (pairs)
3. Compute optimal contraction target $\bar{v}$ for each valid pair (v_1, v_2). Compute the cost of collapsing v_1 and v_2 into $\bar{v}$.
4. Place all pairs into a min heap using their costs
5. Iteratively collapse min cost pair (v_1, v_2) and update the costs.

View Dependent LOD

- Need to refine parts of the object if:
  - That part is in view frustrum
  - It is front facing
  - Not refining that part causes big error on screen
- So we need to start from coarsest level M^0 and <u>selectively</u> refine parts of the object as efficient as possible.
- Is regular Progressive Mesh good enough for this?

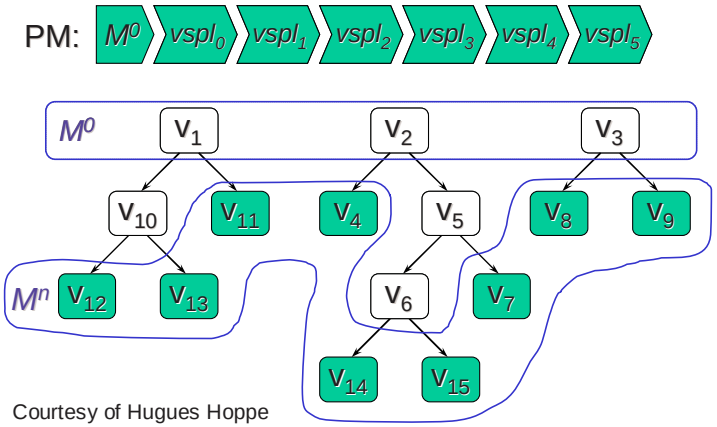
$$\hat{M}=M^n \underset{vsplit_{n-1}}{\overset{ecol_{n-1}}{\rightleftarrows}} \ldots \underset{vsplit_1}{\overset{ecol_1}{\rightleftarrows}} M^1 \underset{vsplit_0}{\overset{ecol_0}{\rightleftarrows}} M^0$$

- No! The order in the list is not necessarily the order we want. We need to keep moving back and forth on the list.

View-Dependent Progressive Meshes

- Start from vertices in the coarse mesh M^0 and build a vsplit tree under each vertex (we'll end up with a forest with coarse level vertices as roots).
- At run time start from the roots nodes and split them if they meet the criteria discussed before

VDPM Hierarchy



Courtesy of Hugues Hoppe

Algorithm

Add the vertices of M^0 to the *active vertex list*
Foreach vertex v in active vertex list
    if qrefine(v)
        if vsplit(v,..) is legal (see details in VDPM paper)
            split v (add its children to the active vertex list)
        else
            force split of every vertex that v's neighborhood depends on
    else
        if ecol(v,..) is legal (see details in VDPM paper)
            collapse v (add its parent to active vertex list)

```
function qrefine(v)
        if outside_view_frustum(v) return false
        if oriented_away(v) return false
        if screen_space_error(v) <= t return false
        return true
```

Further Issues

- Geomorphs:
  - Blend between levels M^i and M^j to avoid popping
- Regulation
  - Try achieving constant frame rate by varying screen space tolerance t
- Amortization
  - Distribute traversing the tree over frames (works well for slowly changing view)

References

1. H. Hoppe. Progressive Meshes. ACM Siggraph 1996.
2. M. Garland, P. S. Heckbert. Surface simplification using quadric error metrics. ACM Siggraph 97
3. H. Hoppe. View dependent refinement of progressive meshes. ACM Siggraph 1997.
4. Various LOD course notes at http://lodbook.com/course/
5. P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, Real-time continuous level-of-detail rendering of height fields. ACM Siggraph 1996.
6. M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, M. B. Mineev-Weinstein. ROAMing terrain: real-time optimally adapting meshes. IEEE Visualization 97.
7. More terrain LOD links at: http://www.vterrain.org/LOD/