Real-Time Rendering (Echtzeitgraphik)

Dr. Michael Wimmer
wimmer@cg.tuwien.ac.at
Levels of Detail
Basic Idea

- Problem: even after visibility, model may contain too many polygons
- Idea: Simplify the amount of detail used to render small or distant objects
- Known as
  - Multiresolution modeling, polygonal simplification, geometric simplification, mesh reduction, decimation, multiresolution modeling, …
Polygonal simplification methods simplify the polygonal geometry of small or distant objects. Does not change rasterization.

Note:

- Levels of detail, but:
- Level-of-detail rendering
- NOT: level of details!
Traditional Approach

Create *levels of detail* (LODs) for each object in a preprocess (or by hand):

- 10,108 polys
- 1,383 polys
- 474 polys
- 46 polys
Traditional Approach

- At runtime, distant objects use coarser LODs:
LOD Issues

- LOD generation
  - Simplification methods
    - How to reduce polygons
  - Error measures
    - Which polygons to reduce

- Runtime system
  - LOD framework
    - Which LODs are eligible
  - LOD selection
    - Criteria for which LODs are selected
  - LOD switching
    - How to avoid artifacts
Runtime system

- LOD framework
  - Discrete
  - Continuous (a.k.a. progressive)
  - View-dependent

- LOD selection
  - Static (distance/projected area-based)
  - Reactive (react to last frames rendering time)
  - Predictive (cost/benefit model)

- LOD switching
  - Hard switching (popping artifacts!)
  - Blending (ill-defined because of z-buffer!)
  - Geomorph
Creating LODs

- Main topic of this lecture!
- Simplification methods (“operators”)
  - Geometry
    - Edge collapse
    - ...
  - Topology
- What criteria to guide simplification?
  - Visual/perceptual criteria are hard
  - Geometric criteria are more common
Simplification Operators

- Local geometry simplification
  - Iteratively reduce number of geometric primitives (vertices, edges, triangles)
- Topology simplification
  - Reducing number of holes, tunnels, cavities
- Global geometry simplification
Local Geometry Simplification

- Edge collapse
- Vertex-pair collapse
- Triangle collapse
- Cell collapse
- Vertex removal
- General geometric replacement
Edge Collapse

Hoppe, SIGGRAPH 96; Xia et al., Visualization 96; Hoppe, SIGGRAPH 97; Bajaj et al., Visualization 99; Gueziec et al., CG&A 99; …
Half-Edge Collapse

Half-edge collapse

Vertex split
Watch for Mesh Foldovers

Calculate the adjacent face normals, then test if they would flip after simplification.

If so, that simplification can be weighted heavier or disallowed.
Implementation: Watch for Identical / Non-Manifold Tris

Edge collapse

\( V_a \)

\( V_b \)

\( V_{\text{new}} \)
Vertex-Pair Collapse

Vertex pair collapse

Vertex split

Triangle Collapse

Triangle collapse

Hamann, *CAGD 94*; Gieng et al., *IEEE TVCG 98*
Cell Collapse

Octree-based: Luebke & Erikson, *SIGGRAPH* 98
Vertex Removal

Vertex removal

Triangulation

Schroeder et al., SIGGRAPH 92;
Klein & Kramer, Spring Conf. On Comp. Graphics 97
General Geometric Replacement

- Replace a subset of adjacent triangles by a simplified set with
- “Multi-triangulation”
- Fairly general: can encode edge collapses, vertex removals, and edge flips
Discussion / Comparison

- **Edge collapse and triangle collapse:**
  - Simplest to implement
  - Support geometric morphing across levels of detail
  - Support non-manifold geometry

- **Full-edge vs. half-edge collapses:**
  - Full edge represents better simplifications
  - Half-edge is more efficient in incremental encoding

- **Cell collapse:**
  - Simple, robust
  - Varies with rotation/translation of grid

- **Vertex removal vs edge collapse**
  - Hole retriangulation is not as simple as edge collapse
  - Smaller number of triangles affected in vertex removal
Simplifying Geometry vs Topology

- Pure geometric simplification not enough
Local Topology Simplification

- Collapsing vertex pairs ("pair contraction") / virtual edges
  - Schroeder, *Visualization 97*
  - Popovic and Hoppe, *SIGGRAPH 97*
  - Garland and Heckbert, *SIGGRAPH 97*

- Collapsing primitives in a cell
  - Rossignac and Borrel, *Modeling in Comp. Graphics 93*
  - Luebke and Erikson, *SIGGRAPH 97*
Virtual Edge Collapse

- Allow virtual edge collapses
- Limit no. of virtual edges (potentially $O(n^2)$)
- Typical constraints:
  - Delaunay edges
  - Edges that span neighboring cells in a spatial subdivision: octree, grids, etc.
  - Maximum edge length
Sample and reconstruct
Adaptive subdivision
Sample and Reconstruct

- Scatter surface with sample points
  - Randomly
  - Let them repel each other
- Reduce sample points
- Reconstruct surface
Adaptive Subdivision

- Create a very simple *base model* that represents the model
- Selectively subdivide faces of base model until fidelity criterion met (draw)
- Big potential application: *multiresolution modeling*
Example 1: Vertex Clustering

- Rossignac and Borrel, 1992
- Operator: cell collapse

- Apply a uniform 3D grid to the object
- Collapse all vertices in each grid cell to single *most important* vertex, defined by:
  - Curvature \((1 / \text{maximum edge angle})\)
  - Size of polygons (edge length)
- Filter out degenerate polygons
Example 1: Vertex Clustering

- Apply a uniform 3D grid to the object
- Collapse all vertices in each grid cell to single *most important* vertex, defined by:
  - Curvature \((1 / \text{maximum edge angle})\)
  - Size of polygons (edge length)
- Filter out degenerate polygons
Vertex Clustering

- Resolution of grid determines degree of simplification

- Representing degenerate triangles
  - Edges: OpenGL line primitive
  - Points: OpenGL point primitive
Vertex Clustering

Pros
- Very fast
- Robust (topology-insensitive)

Cons
- Difficult to specify simplification degree
- Low fidelity (topology-insensitive)
- Underlying grid creates sensitivity to model orientation
Creating LODs: Error Measures

- What criteria to guide simplification?
  - Visual/perceptual criteria are hard
  - Geometric criteria are more common

- Examples:
  - Vertex-vertex distance
  - Vertex-plane distance
  - Point-surface distance
  - Surface-surface distance
  - Image-driven

- Issues:
  - Error propagation?
  - Need to include attributes (tex coords, …)
Quadric Error Metric

- Vertex-plane distance
- Minimize distance to all planes at a vertex
- Plane equation for each face:

\[ p : \ Ax + By + Cz + D = 0 \]

- Distance to vertex \( \mathbf{v} \): 

\[ p^T \cdot \mathbf{v} = \begin{bmatrix} A & B & C & D \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \]
\[ \Delta(v) = \sum_{p \in \text{planes}(v)} (p^T v)^2 \]

\[ = \sum_{p \in \text{planes}(v)} (v^T p)(p^T v) \]

\[ = \sum_{p \in \text{planes}(v)} v^T (pp^T) v \]

\[ = v^T \left( \sum_{p \in \text{planes}(v)} pp^T \right) v \]
Quadric Derivation (cont’d)

- $pp^T$ is simply the plane equation squared:
  $$
  pp^T = \begin{bmatrix}
  A^2 & AB & AC & AD \\
  AB & B^2 & BC & BD \\
  AC & BC & C^2 & CD \\
  AD & BD & CD & D^2
  \end{bmatrix}
  $$

- The $pp^T$ sum at a vertex $v$ is a matrix, $Q$:
  $$
  \Delta(v) = v^T (Q) v
  $$
Using Quadrics

- Construct a quadric $Q$ for every vertex $v$

The edge quadric:

- Sort edges based on edge cost
  - Suppose we contract to $v_{new}$:
    - Edge cost $= V_{new}^T \bar{Q} V_{new}$
    - $V_{new}$’s new quadric is simply $Q$

$$\bar{Q} = Q_1 + Q_2$$
Optimal Vertex Placement

- Each vertex has a quadric error metric \( Q \) associated with it
  - Error is zero for original vertices
  - Error nonzero for vertices created by merge operation(s)
- Minimize \( Q \) to calculate optimal coordinates for placing new vertex
  - Details in paper
  - Authors claim 40-50% less error
Boundary Preservation

- To preserve important boundaries, label edges as normal or discontinuity.
- For each face with a discontinuity, a plane perpendicular intersecting the discontinuous edge is formed.
- These planes are then converted into quadrics, and can be weighted more heavily with respect to error value.
Pros:
- Fast! (bunny to 100 polygons: 15 sec)
- Good fidelity even for drastic reduction
- Robust -- handles non-manifold surfaces
- Aggregation -- can merge objects
Quadric Error Metric

Cons:

- Introduces non-manifold surfaces

- Tweak factor $t$ is ugly
  - Too large: $O(n^2)$ running time
  - Correct value varies with model density

- Needs further extension to handle color (7x7 matrices)
Image-Driven Simplification

12 cameras used to capture quality of bunny simplification (Lindstrom/Turk 2000)

- Measure error by rendering
  - Compare resulting images
  - Lindstrom/Turk 2000
- Captures attribute and shading error, as well as texture content
Appearance-Preserving Simplification

- Reduce drastically
- Simulate lost geometry using bump maps
- NVIDIA/ATI tools available

original: 13,000 tris
simplification: 1,700 tris
normal-mapped: 1,700 tris
Three basic LOD frameworks:

- *Discrete LOD*: the traditional approach
- *Continuous LOD*: encoding a continuous spectrum of detail from coarse to fine
- *View-dependent LOD*: adjusting detail across the model in response to viewpoint
Discrete LOD: Advantages

- Simplest programming model; decouples simplification and rendering
  - LOD creation need not address real-time rendering constraints
  - Run-time rendering engine need only pick LODs
- Fits modern graphics hardware well
  - Easy to compile each LOD into triangle strips, cache-aware vertex arrays, etc.
  - These render *much* faster than immediate-mode triangles on today’s hardware
So why use anything but discrete LOD?

- **Reason 1:** sometimes discrete LOD not suited for **drastic simplification**
- **Reason 2:** in theory, can get better **fidelity/polygon** with other approaches
Continuous Level of Detail

- A departure from the traditional discrete approach:
  - **Discrete LOD**: create individual levels of detail in a preprocess
  - **Continuous LOD**: create data structure from which a desired level of detail can be extracted *at run time.*
Continuous LOD: Advantages

- Better granularity → better fidelity
  - LOD is specified exactly, not chosen from a few pre-created options
  - Thus objects use no more polygons than necessary, which frees up polygons for other objects
  - Net result: better resource utilization, leading to better overall fidelity/polygon
Continuous LOD: Advantages

- Better granularity → smoother transitions
  - Switching between traditional LODs can introduce visual “popping” effect
  - Continuous LOD can adjust detail gradually and incrementally, reducing visual pops
    - Can even *geomorph* the fine-grained simplification operations over several frames to eliminate pops (e.g., w/ a vertex shader)
Continuous LOD: Advantages

- Supports progressive transmission (*streaming*)
  - Progressive Meshes [Hoppe 97]
  - Progressive Forest Split Compression [Taubin 98]

- Leads to
  - Use current view parameters to select best representation *for the current view*
  - Single objects may thus span several levels of detail
Continuous LOD Algorithm

- “Progressive meshes”
- Iteratively apply local simplification operator
  - Until base mesh
- Entity = edge or vertex or triangle …

Sort all entities (by some metric)
repeat
    Apply local simplification operator:
    remove entity
    Fix-up topology
until (no entities left)
View-Dependent LOD: Examples

- Show nearby portions of object at higher resolution than distant portions

View from eyepoint

Birds-eye view
View-Dependent LOD: Examples

- Show silhouette regions of object at higher resolution than interior regions
Advantages of View-Dependent LOD

- Even better granularity
- Enables drastic simplification of very large objects
  - Example: stadium model
  - Example: terrain flyover
Drastic Simplification:
The Problem With Large Objects
Terrain LOD

- Has been around for long (flight simulators, GIS, games …)
- Geometry is more constrained
  - Specialized solutions

- Properties
  - Simultaneously very near and very far
    - Requires progressive/view-dependent LOD!
  - Very large terrains → out-of-core

- Problems:
  - Dynamic modification of terrain data
  - Fast rotation
Regular Grids

- Uniform array of height values
- Simple to store and manipulate
- Easy to interpolate to find elevations
- Less disk/memory (only store z value)
- Easy view culling and collision detection
- Used by most implementers
Triangulated Irregular Networks

- Fewer polygons needed to attain required accuracy
- Higher sampling in bumpy regions and coarser in flat ones
- Can model maxima, minima, ridges, valleys, overhangs, caves
LOD Hierarchy Structures

QuadTree Hierarchy

BinTree Hierarchy
Quadtrees

- Each quad is actually two triangles
- Produces cracks and T-junctions
- Easy to implement
- Good for out-of-core operation
Bintrees

- Terminology
  - Binary triangle tree (bintree, binritree, BTT)
  - Right triangular irregular networks (RTIN)
  - Longest edge bisection
- Easier to avoid cracks and T-junctions
- Neighbor is never more than 1 level away
- Very popular “ROAM” algorithm
Cracks and T-Junctions

- Avoid cracks:
  - Force cracks into T-junctions / remove floating vertex
  - Fill cracks with extra triangles

- Avoid T-junctions:
  - Continue to simplify ...
Avoiding T-junctions

- In bintrees:
View-Dependent Terrain LOD

- Hoppe et al.

actual view

overhead view