

# Out-of-Core Simplification of Large Polygonal Models

Peter Lindstrom  
Georgia Institute of Technology

Presented by Jinzhu Gao  
(Some graphs are from Lindstrom's Ph.D thesis and his SIGGRAPH presentation)

1

## Motivation

- Extremely complex polygonal models
  - Large number of triangles
    - Visible Human: over 10 billion voxels
    - Michelangelo's sculpture: 2 billion triangles
  - Great challenge to mesh processing tools as well as simplification method
    - Large memory consumption
    - Insufficient simplification speed
- Difficult to adapt existing in-core methods for out-of-core simplification
  - full connectivity information needed → large in-core data structures

2

## Potential Solution

- Oversampling
  - Solution: Adaptive sampling during the data acquisition
  - Problems:
    - Need specialized tools
    - Place additional burden on the data acquirer
    - Ignore application demands
    - Model may still be very large

3

## Surface Simplification

- In-core algorithm:
  - Pros:
    - Automated process
    - Can be tailored for the application
  - Cons:
    - Models may be too large to fit in the main memory
- Out-of-core simplification

4

## Previous Work

- Vertex Clustering: [Rossignac93]
  - Fast, but low quality
- Model segmentation [Bernardini99]
  - Split model, simplify separately in-core, stitch together, re-partition to simplify seams
  - Simple, but high time and space overhead

5

## OoCS: New Out-of-Core Simplification Algorithm

- A hybrid method:
  - Vertex clustering [Rossignac93]
    - Uniform rectilinear partitioning of vertices into 3D grid cells
  - Quadric error metric [Garland97]
    - Accurate positioning of vertex cluster representatives
  - Non-uniform quadric weighting [Lindstrom98]
    - Weight each quadric by the squared triangle area → Improved speed and quality for non-uniform triangulations

6

## The Goal of OoCS

- **Input:** A model of arbitrary complexity
- **Output:** A model that is small enough for in-core mesh processing tools to handle and store internally

7

## Model Representation

- “triangle soup”: an off-line data structure
  - Each triangle is represented directly as a triplet of vertex coordinates
  - **Pros:**
    - Allows non-fixed-length representation (text file)
    - Can be compressed externally and then uncompressed on-the-fly
    - The model can be split up into several files
- Use a bounding box for the model, which is divided into a user-specified number of rectilinear grid cells

8

## Vertex Clustering (1)

- Main idea:
  - Uniform rectilinear partitioning of vertices into 3D grid cells
  - Replace all vertices in a grid cell by a single representative vertex
- Vertex clustering is a special case of vertex pair contraction



9

## Vertex Clustering (2)

- Original algorithm

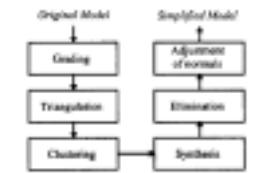


Figure 1: Steps in the vertex-clustering method.

(Low & Tan 97)

10

## Vertex Clustering (3)

- New algorithm:
  - Use error quadrics instead of grading each vertex with an important value
  - Compared to original vertex pair contraction algorithm, the sequence of contractions in OoCS is determined by the cluster grid rather than by the mesh geometry

11

## Error Quadrics for Vertex Placement

- Quadric error metric:
  - A weighted sum of squared distances from vertex to a set of planes
    - Vertex: Cluster representative
    - Planes: Triangles in cluster



12

## Quadric Error for Each Triangle

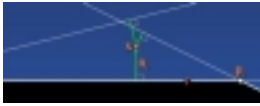
- Squared distance from vertex  $x$  to plane  $(\hat{n}, p_i)$  of triangle  $t = (X'_1, X'_2, X'_3)$

$$Q_t(x) = (\hat{n}^T(x - p_i))^2 = \bar{x}^T Q_t \bar{x}$$

$$Q' = Nt^T Nt$$

$$Nt = [-(X'_1 \times X'_2 + X'_2 \times X'_3 + X'_3 \times X'_1) \quad [X'_1, X'_2, X'_3]]$$

$$\bar{x} = [x_1 \quad x_2 \quad x_3 \quad 1]$$



13

## Representative Vertex Position

- Sum of squared distances:

$$Q(x) = \sum_{t \in V} \sum_{i \in [1,3]} Q_t(x) = \bar{x}^T \begin{bmatrix} A & -b \\ -b^T & c \end{bmatrix} \bar{x}$$

- Optimal position: Minimizes  $Q(x)$ 
  - Solve  $Ax = b$  using singular value decomposition:

$$x = A^+ b + (I - A^+ A) \hat{x}$$

- $x$  is solution to  $Ax=b$  and closest to  $\hat{x}$

14

## OoCS Algorithm

```

1 set  $V_{\text{out}}$  and  $T_{\text{out}}$  to 0
2 for each triangle  $t \in T_{\text{in}}$ 
3   fetch vertex coordinates  $x'_1, x'_2, x'_3$ 
4   compute quadric matrix  $Q^t$ 
5   for each vertex  $v \in t$ 
6     map  $v$  to cluster representative  $\hat{v}$ 
7     if  $\hat{v} \notin V_{\text{out}}$ 
8       then add  $v$  to  $V_{\text{out}}$  and set  $Q^v$  to 0
9     add  $Q^t$  to  $Q^{\hat{v}}$ 
10    if  $v'_1, v'_2, v'_3$  belong to distinct clusters
11      then add  $t$  to  $T_{\text{out}}$ 
12 for each vertex  $v \in V_{\text{out}}$ 
13   compute  $x^v = \text{argmin}_x x^T Q^v x$ 
14 output  $(V_{\text{out}}, T_{\text{out}})$ 
    
```

Table 5.1: Pseudo-code for the OoCS algorithm.

15

## Features of OoCS Algorithm

- Fast, easy, single pass  $O(n)$  algorithm
  - About 100,000 triangles/second reduction
- Output-sensitive memory requirements
  - Depend only on the size of the simplified model
- No extra disk space beyond the input mesh
- “Triangle soup” makes model segmentation and compression possible

16

## Weakness of OoCS Algorithm

- Allow modification of the surface topology
  - May introduce non-manifold simplices
- No connection information
  - Surface boundaries are generally not preserved well.

17

## Results

- Test models:
  - Dragon: 871,306 triangles
  - Buddha: 1,087,716 triangles
  - Subdivided turbine blade: 28,246,208 triangles
  - St. Matthew: 386,488,573 triangles
- Test machine:
  - 195 MHz R10000 SGI Origin, 4GB RAM, SCSI disk drive
- Comparison with two in-core methods:
  - Qslim [Garland97]
  - Memoryless Simplification [Lindstrom98]

18

*Results (1)*  
 – Buddha Model



19

*Results (2)*  
 – Exterior Turbine Blade Model

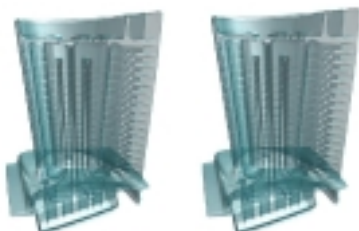


28,246,208

507,104

20

*Results (3)*  
 – Interior Turbine Blade Model

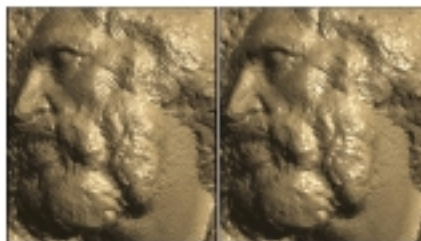


28,246,208

507,104

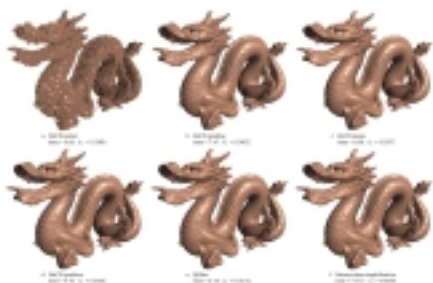
21

*Results (4) – St. Matthew Model*



22

*Results (5) – Dragon Model*



23

*Results (6)*  
 -- Mean Geometric Errors

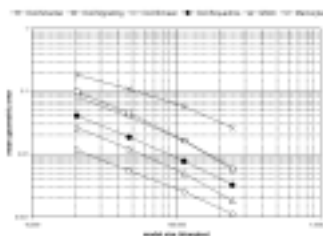


Figure 5.5: Mean geometric errors for the dragon model.

24

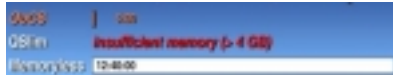
## Results (7)

### – Simplification Time

- Dragon model: 871,306 to 47,236 triangles



- Blade model: 28,246,208 to 507,104 triangles



25

## Results (8) – Theoretical Memory Usage

- OoCS: 72 bytes × # triangles in *simplified* model
- Qslim: 100 bytes × # triangles in *original* model
- Memoryless: 80 bytes × # triangles in *original* model
  
- St. Matthew Model (386 million to 3 million triangles)
  - OoCS: 210 MB
  - Qslim: 37,000 MB
  - Memoryless: 29,000 MB

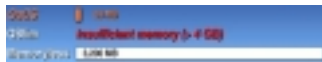
26

## Results (9) – Measured Memory Usage

- Dragon model: 871,306 to 47,236 triangles



- Blade model: 28,246,208 to 507,104 triangles



27

## Future Work

- Adaptive/hierarchical simplification
  - Cells are recursively merged in less detailed regions
- Use quadric information to improve the connectivity of the mesh
- Integrate the algorithm with Marching Cubes algorithm for isosurface extraction

28

Thank you !!!

29