Mesh Simplification for Visualization

Miguel Pasenau
Contents

› Motivation
› Algorithm
› Results
› In GiD
› Future
Contents

› Motivation

› Algorithm

› Results

› In GiD

› Future
Motivation

Some of GiD objectives:
• Run both in modest and high performance platforms
• At interactive frame rates
• Remote visualization

Today and future simulations:
• Large meshes
• Many, many time-steps

→ Simplification:
• Achieve interactive frame rates
• Visualize simplified model, featured model behind

6th GiD Convention, 11th May 2012, 4.42
Contents

› Motivation

› Algorithm

› Results

› In GiD

› Future
Vertex clustering algorithm:

- Uniform vertex grouping using a regular grid
- DeCoro and Tatarchuk 2007 approach:
  - Step 1: creation of QEF map from original mesh
  - Step 2: find optimal representative position for each cell
  - Step 3: mesh decimation, collapse original vertices to cell representatives
- Using the Quadric Error Function of Garland and Heckbert 1997:

\[
 f(v) = \sum_{p \in \text{planes}(v)} (p^T v)^2
\]

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

\[
 = v^T Q v v
\]
**Vertex cluster algorithm**

**Step 1:** per triangle’s vertex, accumulate position and triangle QEF in vertex’s cell
**Vertex cluster algorithm**

**Step 2**: find optimal representative by:
- inverting QEF matrix, if it’s outside cell, use centroid

<table>
<thead>
<tr>
<th>QEF acc</th>
<th>QEF acc</th>
<th>QEF acc</th>
<th>QEF acc</th>
<th>QEF acc</th>
<th>QEF acc</th>
<th>QEF acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
</tr>
<tr>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
</tr>
<tr>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
</tr>
<tr>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
</tr>
<tr>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
</tr>
<tr>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
</tr>
<tr>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
<td>QEF acc</td>
</tr>
</tbody>
</table>

6th GiD Convention, 11th May 2012, 8.42
**Step 3:** for each triangle:
if all three nodes are in different cells:
use cell representatives as **triangle** vertices;
save **triangle**.
Vertex cluster algorithm

**Step 3**: for each triangle:
if **two of three** nodes are in different cells:
use cell representatives as **line** vertices;
save **line**.

![Image of vertex cluster algorithm with a 9 x 9 x 9 grid and two teapot models with line vertices highlighted.](image-url)
Enhancements over original

Collapsed triangles $\rightarrow$ creation of lines

19 x 19 x 19 grid
Enhancements over original

Avoid repeated triangles and lines in simplified mesh

Normal correction:

Original mesh

Simplified mesh
First level: group cells in 512 sized buckets

\[ h(k) = k \mod \# \text{buckets} \]
Second level: at each bucket, cuckoo hash with three sub-tables

Max 512 items

409 items on average

\[ g_1(k) \quad g_2(k) \quad g_3(k) \]

Table T1

Table T2

Table T3
List of keys = cells id’s of points

Constant access: 2..4

Occupation level = 409 / 573 = 71.38 %

Cell stores:
  cell id, QEF and accumulated coordinates
Final algorithm

- **Step 1:**
  - create list of occupied cell id’s
  - Create hybrid hash
- **Step 2:** fill hybrid hash with QEF and positions from original mesh
- **Step 3:** find optimal representative position for each cell
- **Step 3:** for each triangle, collapse vertices to cell representatives and get simplified element
Grid $2^3$
8 puntos
14 triángulos
0 líneas
< 0,0001 %
62 s. con HashUC
62 s. con FullUC
City model

Original: 6,1 M puntos y 15,6 M triángulos
Scalability

Cpu: Intel i7-920
Simplification of Lucy with a $256^3$ grid

<table>
<thead>
<tr>
<th>proceses</th>
<th>1 core</th>
<th>2 cores</th>
<th>4 cores</th>
<th>4 c. + HT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash creation</td>
<td>7 %</td>
<td>7 %</td>
<td>7 %</td>
<td>5 %</td>
</tr>
<tr>
<td>Qef map creation</td>
<td>60 %</td>
<td>58 %</td>
<td>59 %</td>
<td>59 %</td>
</tr>
<tr>
<td>Optimal repres.</td>
<td>1 %</td>
<td>2 %</td>
<td>3 %</td>
<td>3 %</td>
</tr>
<tr>
<td>Mesh decimation</td>
<td>30 %</td>
<td>32 %</td>
<td>28 %</td>
<td>29 %</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>9 s.</td>
<td>5 s.</td>
<td>3 s.</td>
<td>2.5 s.</td>
</tr>
</tbody>
</table>
Scalability

Time distribution

<table>
<thead>
<tr>
<th>processes</th>
<th>1 core</th>
<th>2 cores</th>
<th>4 cores</th>
<th>4 c. + HT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash creation</td>
<td>23 %</td>
<td>27 %</td>
<td>32 %</td>
<td>36 %</td>
</tr>
<tr>
<td>QEF map creation</td>
<td>22 %</td>
<td>20 %</td>
<td>17 %</td>
<td>15 %</td>
</tr>
<tr>
<td>Optimal repres.</td>
<td>3 %</td>
<td>3 %</td>
<td>3 %</td>
<td>3 %</td>
</tr>
<tr>
<td>Mesh decimation</td>
<td>51 %</td>
<td>49 %</td>
<td>46 %</td>
<td>44 %</td>
</tr>
<tr>
<td>Total</td>
<td>34 s.</td>
<td>20 s.</td>
<td>13 s.</td>
<td>10 s.</td>
</tr>
</tbody>
</table>

Cpu: Intel i7-920
Simplification of Lucy with a $2.048^3$ grid
Simplification library times

Lucy model
14,027,872 nodes
28,055,742 triangles

Neptune model
2,003,932 nodes
4,007,872 triangles

Car model
3,005,848 nodes
6,011,696 triangles

Jaw model
457,164 nodes
855,652 triangles
## Simplification library times

QuadCore Q9550  4 cores  @ 2.83 GHz

<table>
<thead>
<tr>
<th>model</th>
<th># nodes</th>
<th># triangles</th>
<th>level</th>
<th># nodes</th>
<th>triangles # lines</th>
<th>T simpl</th>
<th>T attr</th>
<th>T total</th>
</tr>
</thead>
<tbody>
<tr>
<td>lucy</td>
<td>14.027.872</td>
<td>28.055.742</td>
<td>128</td>
<td>22.785</td>
<td>46.569</td>
<td>4,24</td>
<td>1,25</td>
<td>5,49</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>90.780</td>
<td>182.902</td>
<td>3,77</td>
<td>1,12</td>
<td>4,90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>351.327</td>
<td>705.207</td>
<td>4,56</td>
<td>1,18</td>
<td>5,74</td>
</tr>
<tr>
<td>neptune</td>
<td>2.003.932</td>
<td>4.007.872</td>
<td>128</td>
<td>17.017</td>
<td>34.401</td>
<td>0,78</td>
<td>0,21</td>
<td>0,98</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>66.242</td>
<td>133.215</td>
<td>0,80</td>
<td>0,18</td>
<td>0,99</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>244.733</td>
<td>491.619</td>
<td>1,38</td>
<td>0,25</td>
<td>1,62</td>
</tr>
<tr>
<td>car</td>
<td>3.005.848</td>
<td>6.011.696</td>
<td>128</td>
<td>20.632</td>
<td>41.603</td>
<td>1,03</td>
<td>0,27</td>
<td>1,30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>83.739</td>
<td>168.453</td>
<td>1,13</td>
<td>0,29</td>
<td>1,42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>331.411</td>
<td>665.791</td>
<td>1,95</td>
<td>0,37</td>
<td>2,32</td>
</tr>
<tr>
<td>jaw</td>
<td>457.164</td>
<td>855.652</td>
<td>128</td>
<td>53.279</td>
<td>109.438</td>
<td>0,23</td>
<td>0,05</td>
<td>0,28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>182.695</td>
<td>349.512</td>
<td>0,49</td>
<td>0,08</td>
<td>0,58</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>365.064</td>
<td>679.911</td>
<td>0,87</td>
<td>0,14</td>
<td>1,02</td>
</tr>
</tbody>
</table>
## Simplification library times

**i7-920 4 cores + HT @ 2.67 GHz**

<table>
<thead>
<tr>
<th>model</th>
<th># nodes</th>
<th># triangles</th>
<th>level</th>
<th># nodes</th>
<th>triangles # lines</th>
<th>T simp</th>
<th>T attr</th>
<th>T total</th>
</tr>
</thead>
<tbody>
<tr>
<td>lucy</td>
<td>14.027.872</td>
<td>28.055.742</td>
<td>128</td>
<td>22.785</td>
<td>46.569</td>
<td>2,70</td>
<td>0,36</td>
<td>3,06</td>
</tr>
<tr>
<td></td>
<td>90.780</td>
<td>182.902</td>
<td>11</td>
<td>1,92</td>
<td>0,23</td>
<td>2,15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>351.327</td>
<td>705.207</td>
<td>17</td>
<td>1,97</td>
<td>0,21</td>
<td>2,18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>neptune</td>
<td>2.003.932</td>
<td>4.007.872</td>
<td>128</td>
<td>17.017</td>
<td>34.401</td>
<td>0,59</td>
<td>0,09</td>
<td>0,68</td>
</tr>
<tr>
<td></td>
<td>66.242</td>
<td>133.215</td>
<td>6</td>
<td>0,39</td>
<td>0,04</td>
<td>0,43</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>244.733</td>
<td>491.619</td>
<td>6</td>
<td>0,53</td>
<td>0,05</td>
<td>0,58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>3.005.848</td>
<td>6.011.696</td>
<td>128</td>
<td>20.632</td>
<td>41.603</td>
<td>0,66</td>
<td>0,09</td>
<td>0,75</td>
</tr>
<tr>
<td></td>
<td>83.739</td>
<td>168.453</td>
<td>0</td>
<td>0,53</td>
<td>0,06</td>
<td>0,59</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>331.411</td>
<td>665.791</td>
<td>0</td>
<td>0,75</td>
<td>0,08</td>
<td>0,84</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jaw</td>
<td>457.164</td>
<td>855.652</td>
<td>128</td>
<td>53.279</td>
<td>109.438</td>
<td>0,24</td>
<td>0,06</td>
<td>0,30</td>
</tr>
<tr>
<td></td>
<td>182.695</td>
<td>349.512</td>
<td>376</td>
<td>0,32</td>
<td>0,04</td>
<td>0,36</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>365.064</td>
<td>679.911</td>
<td>212</td>
<td>0,33</td>
<td>0,04</td>
<td>0,37</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Simplification library: gidml_gmeshsim
• Attribute interpolation (averaging cell values)
• Fast rotation mode:
  simplified mesh
  simplified contour fill result
parameters: grid size, mesh size to be simplified
• Available in next developer version
Library in GiD

Original car surface: 5,215,728 triangles
0.278 fps

laptop dual pentium
1.3GHz w. intel graphics
Library in GiD

Original car surface:
5,215,728 triangles
0.278 fps

laptop dual pentium
1,3GHz w. intel graphics

Simplified mesh, 1024³:
592,340 triangles
2.33 fps
Library in GiD

Original car surface: 5,215,728 triangles 0.278 fps

laptop dual pentium 1,3GHz w. intel graphics

Simplified mesh, 512³: 233,470 triangles 6.42 fps
Library in GiD

Original car surface: 5.215.728 triangles, 0.278 fps

laptop dual pentium 1,3GHz w. intel graphics

Simplified mesh, $256^3$: 58.472 triangles, 20.8 fps
Library in GiD

Original car surface:
5,215,728 triangles
0.278 fps

laptop dual pentium
1.3GHz w. intel graphics

Simplified mesh, 128³:
14,384 triangles
42.9 fps
Library in GiD

Original car surface:
5,215,728 triangles
0.278 fps

laptop dual pentium
1,3GHz w. intel graphics

Simplified mesh, 64³:
3,534 triangles
58.1 fps
Library in GiD

Node close-up: original mesh
Library in GiD

Node close-up: $256^3$ simplification
Library in GiD

Live demo
Future lines: GiD

Fast rotation:
  - interpolation of results for isosurfaces, vectors, etc.
  - also in preprocess

Use simplified mesh always for visualization, user enabled, but internally using full featured mesh

Stream lines, isosurfaces:
  - fast calculation for preview purposes
Another spatial hash scheme, higher occupation factor
Preserve details: boundaries, normals in cells

Use result as error control, in addition to geometry
Multi grid (octree) for adaptive simplification:
already in test
Multi grid / octree

Puget Sound original mesh: 16Mquads
Grid (512^3 735kt 2,6%) vs Octree (<1e-6 723kt 1'9%)
Future lines: library

Port to OpenCL or CUDA

Simplification of volume meshes, and their results

In simulation program, two outputs
simplified version of the model for quick visual analysis
full featured model and results


“Simplificación de mallas de triángulos”, by Miguel Pasenau, Final Project directed by profesor Carlos Andujar, 2011 June, http://upcommons.upc.edu/pfc/handle/2099.1/12487


"Perfect Spatial Hashing" by Sylvain Lefebvre and Hugues Hoppe, ACM Transactions on Graphics 25, 3 (July), 579–588, 2006.


Heckbert, IEEE Visualization 98


Thanks for your attention

Questions and comments?