GPU-based
Tolerance Volumes
for Mesh Processing

Mario Botsch, David Bommes, Christoph Vogel, Leif Kobbelt
GPU-based Tolerance Volumes for Mesh Processing

Raw Data → Mesh Generation → Mesh Optimization → HQ mesh
GPU-based Tolerance Volumes for Mesh Processing

- Raw Data
- Mesh Generation
  - Smoothing
- HQ mesh

Computer Graphics Group
Mario Botsch
GPU-based Tolerance Volumes for Mesh Processing

Raw Data → Mesh Generation

- Smoothing
- Decimation

HQ mesh
GPU-based Tolerance Volumes for Mesh Processing

Raw Data → Mesh Generation

- Smoothing
- Decimation
- Remeshing

→ HQ mesh
GPU-based Tolerance Volumes for Mesh Processing

Raw Data → Mesh Generation → Mesh Optimization → HQ mesh

Error Control
GPU-based Tolerance Volumes for Mesh Processing

Raw Data → Mesh Generation → Error Ctrl’ed Mesh Opt. → HQ mesh

Computer Graphics Group
Mario Botsch
• Control global approximation error
  • Exact (or conservative)

• Each method may provide error control
  • Local errors may accumulate

• Need general global error control
  • Independent of mesh algorithm!
GPU-based \textbf{Tolerance Volumes for Mesh Processing}

- Tolerance volume around original
- Triangles have to stay within it
General distance query
- Implicit representation best suited
  \( \text{distance} = \text{function evaluation} \)
- Approximate signed distance field

Check modified triangle
- Find SDF maximum over triangle

How to approximate SDF?
• Piecewise constant, $C^{-1}$, regular grid
• Permission Grids [Zelinka & Garland]
• Simple triangle test
• High grid resolution
  \textit{(low approx. order, alias artifacts)}
GPU-based Tolerance Volumes for Mesh Processing

• Piecewise tri-linear, $C^0$, octree
  • Adaptively sampled DFs [Frisken et al.]

• Low memory consumption

• Complicated triangle test
  (piecewise cubic function)
GPU-based Tolerance Volumes for Mesh Processing

- Piecewise linear, $C^{-1}$, BSP tree
  - Linear approx. SDFs [Wu & Kobbelt]
- Low memory consumption
- Complicated triangle test
  (split triangles into BSP leaves)
GPU-based Tolerance Volumes for Mesh Processing

- Piecewise tri-linear, $C^0$, regular grid
  - Medium memory consumption
    - (regular grid, linear approximation)
  - Complicated triangle test?
GPU-based Tolerance Volumes for Mesh Processing

- Represent SDF as 3D texture
- Triangle test: Just render it!
  - Automatic voxelization
  - Automatic tri-linear interpolation
- GPUs are efficient
  - Real-time error control
  - Real-time error visualization
Overview

- Introduction
- Texture setup
- Triangle check
- Results
SDF Texture

- Build regular grid \( g_{ijk} \)
- Compute distances \( d_{ijk} = \text{sdf}(g_{ijk}) \)
- Convert to OpenGL 3D texture
Distance Field Generation

- Compute distance at each grid node
- Get \textit{signed} distance from normal

\[(q-p)^T n < 0 ?\]
Distance Field Generation

- Fast Marching Method
  - Initialization
  - March outward
  - March inward
Non-closed Models?

- Use *unsigned* distance field instead
- May over-estimate error by \(\frac{1}{2}h\)
SDF Approximation Error

- Piecewise tri-linear approximation
- May under-estimate error by $\frac{\sqrt{3}}{2} h$
- Adjust user-specified tolerance

$$\varepsilon_{\text{max}} \leftarrow \varepsilon_{\text{max}} - \frac{\sqrt{3}}{2} h$$
Texture Size

- Texture size should be power of 2
  - Fill up with empty rows/cols/slices
  - Waste of texture memory
- Improved by new extension
  - ARB_texture_non_power_of_two
Texture Value Type

- Use unsigned type ALPHA8
  - Map $[-\varepsilon_{\text{max}}, +\varepsilon_{\text{max}}] \rightarrow \{0, \ldots, 255\}$
  - 8 bit discretization of possible errors
  - Turned out sufficient

- Can also use
  - ALPHA16, ALPHA32
  - 32bit float (ATI_texture_float)
Transfer Function

- Map interpolated texture value to
  - color (*error visualization*)
  - abs. distance value (*error check*)

- Apply after tri-linear texture interpolation
  - Texture shader or pixel shader
  - Post-classification
Distance Volume
Overview

- Introduction
- Texture setup
- Triangle check
- Results
Texture Coordinates

- OpenGL assigns texture coordinates to texel *centers*, not *corners*!

\[ t_x = \frac{p_x - o_x + h/2}{(res_x + 1)h} \]
\[ t_y = \frac{p_y - o_y + h/2}{(res_y + 1)h} \]
Texture Coordinates

• OpenGL assigns texture coordinates to texel *centers, not corners*!

• Compute texture coords on GPU
  • Texture matrix
  • Vertex shader

• Real-time visualization
  • 15M dynamic triangles
Triangle Checking

• How to choose viewing position for error checking?
  • Each triangle has to be fully visible
  • Generate sufficiently many pixels

• 3D coordinates not important, only texture coordinates matter
2D Positions, 3D Tex. Coords

3D

\( p_1, t_1 \)

\( p_3, t_3 \)

\( p_2, t_2 \)

2D

\( (0,0), t_1 \)

\( (0,L), t_2 \)

\( (L,0), t_3 \)
Triangle Check

- Draw 2D triangle with 3D texcoords
- Use alpha transfer function
  \[ \alpha(x) = \begin{cases} 
  0, & x \leq \varepsilon_{\text{max}} \\ 
  1, & \text{otherwise} 
\end{cases} \]
- Only invalid pixels will be drawn
  - Pixel drawn \(\Rightarrow\) triangle invalid
  - \texttt{ARB_occlusion_query}
Overview

- Introduction
- Texture setup
- Triangle check
- Results
Error Control & Visualization

Decimation

Smoothing
### Decimation Timings

- GPU decimation is faster than
  - Hausdorff-based decimation (2×)
  - Permission grids (1.5× - 2×)

<table>
<thead>
<tr>
<th>Model</th>
<th>Input</th>
<th>Grid</th>
<th>FM</th>
<th>Deci</th>
<th>QEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bunny</td>
<td>70k</td>
<td>74×73×58</td>
<td>1.7s</td>
<td>2.3s</td>
<td>2.0s</td>
</tr>
<tr>
<td>Horse</td>
<td>96k</td>
<td>85×71×42</td>
<td>1.9s</td>
<td>3.2s</td>
<td>2.6s</td>
</tr>
<tr>
<td>Venus</td>
<td>269k</td>
<td>53×74×74</td>
<td>5.0s</td>
<td>10.3s</td>
<td>8.8s</td>
</tr>
<tr>
<td>Buddha</td>
<td>1M</td>
<td>45×101×45</td>
<td>18.0s</td>
<td>39.6s</td>
<td>34.5s</td>
</tr>
</tbody>
</table>
Freeform Deformation

- Exact deviations due to error control
- Real-time visual feedback
Freeform Deformation
GPU-based Tolerance Volumes

• General global error framework
  • Independent of application

• Simple implementation
  • Complicated tasks done by GPU

• Efficient GPU implementation
  • Error check: 3M tri/sec
  • Error visualization: 15M tri/sec
Acknowledgments

• David Bommes (Framework)
• Christoph Vogel (Fast Marching)
• Michael Lambertz (3D Texture)
• Andreas Wiratanaya (FFD)
• Benjamin Molitor (Vol. Vis.)