Single-pass Wireframe Rendering

with Hidden Surface Removal

Andreas Bærentzen, Steen Lund Nielsen, Mikkel Gjøl, Bent D. Larsen and Niels Jørgen Christensen
Overview

• What is wrong with the traditional **offset based methods** for wireframe drawing?

• A **single-pass** method for wireframe drawing with hidden surface removal.

• Drawing **prefiltered** lines.

• Results

• Discussion and future work.
Offset Based Methods

- Offset based methods have two passes:
  - Pass 1: Draw mesh as filled polygons.
    *Z write and comparison enabled.*
  - Pass 2: Draw mesh as wireframe with Z offset.
    *Z write disabled. Z comparison enabled.*

- Offset ensures filled polygons do not occlude wireframe.
Offset Based Methods

- Two triangles with their edges drawn using no offset, slope based offset, constant offset.
Offset Based Methods

- use line drawing which is slow.
- are inherently two pass (even slower)
- require *antialiasing* to look good (slower still)
- have artifacts because it is impossible to pick a perfect offset in most cases.

• Admittedly, we can live with these methods, but we don't have to!
Approaches

• Edge is drawn only if the pixel already contains a fragment from the corresponding filled polygon. E.g. [Herrel et al. 1995].

• **Our method**: Draw wireframe directly as you are drawing filled polygon. All in one pass.
Single-Pass Method

• For each fragment, compute color from distance to closest edge:

\[ d = \min(d_{01}^P, d_{12}^P, d_{20}^P) \]

\[ \text{if } (d < d_{\text{thresh}}) \]

\[ \text{col} = \text{wire}\_\text{col}; \]

\[ \text{else} \]

\[ \text{col} = \text{fill}\_\text{col}; \]
Computing Distance

- We need 2D window space distance.
- In 2D, distance to line is an affine function.
- Linear interpolation reproduces affine functions.
Computing Distances

- **Per vertex:**
  - Input: All vertices of triangle.
  - Vertex shader projects all vertices of triangle.
  - Distances to all edges are computed.
  - **Output:** vector of distances to all three edges.
Computing Distances

• Per fragment:
  – Interpolate distances.
  
  \[
  \begin{bmatrix}
  d_{01}^p \\
  d_{12}^p \\
  d_{20}^p
  \end{bmatrix}
  = \alpha
  \begin{bmatrix}
  0 \\
  d_{12}^0 \\
  0
  \end{bmatrix}
  + \beta
  \begin{bmatrix}
  0 \\
  0 \\
  d_{20}^1
  \end{bmatrix}
  + \gamma
  \begin{bmatrix}
  d_{01}^2 \\
  0 \\
  0
  \end{bmatrix}
  \]
  
  – Compute color using previous fragment program.

• Note: This method extends to quads.
Perspective Correction

- Oops! Graphics cards perform perspective correct interpolation

\[ f = \frac{\alpha f_a / w_a + b f_b / w_b + c f_c / w_c}{\alpha/w_a + b/w_b + c/w_c} \]

- However, it is easy to fix:
  - Negate perspective correction.
  - Wait for D3D 10 or OpenGL 3.0 hardware.
Teapot using our method ... so far.
Preintegrated lines

- Color is now a function of distance:

\[
d = \min(d_{01}^p, d_{12}^p, d_{20}^p);
\]

\[
\text{if}\ (d < d_{\text{thresh}})
\]
\[
\text{col} = I(d) \ast \text{wire\_col} + (1 - I(d)) \ast \text{fill\_col};
\]

\[
\text{else}
\]
\[
\text{col} = \text{fill\_col};
\]
Teapot using our method
Single-pass Method

- Detail of teapot
- Notice smooth lines
- Perhaps lines are a little thick?
Single-pass Method

- Thinner lines are easy.
- However, we trade thinness for a little aliasing.
Offset-based method

- Const offset = 2000
- Notice stippling
- Offset could be changed but there is no perfect value
Offset-based method

- Slope based offset
- Notice disocclusion artefacts
Use depth to fade out line color.
• Use depth to decide line thickness.
• Use alpha testing to remove interior of polygon.
Indexed Primitives

- Using the geometry shaders of D3D 10 allows us to draw wireframe triangle strips.
• Line Styles
## Results

- **Performance**

<table>
<thead>
<tr>
<th></th>
<th>GeForce FX5900XT</th>
<th>Quadro FX3000</th>
<th>GeForce 6600GT</th>
<th>GeForce 7800 GTX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Teapot, 6320 triangles</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>offset</td>
<td>209.31</td>
<td>734.76</td>
<td>363.04</td>
<td>482.68</td>
</tr>
<tr>
<td>single</td>
<td>254.22</td>
<td>195.81</td>
<td>589.59</td>
<td>1512.09</td>
</tr>
<tr>
<td><strong>Bunny, 69451 triangles</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>offset</td>
<td>44.26</td>
<td>138.73</td>
<td>65.12</td>
<td>66.44</td>
</tr>
<tr>
<td>single</td>
<td>97.68</td>
<td>103.65</td>
<td>146.7</td>
<td>348.22</td>
</tr>
<tr>
<td><strong>Buddha, 1087716 triangles</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>offset</td>
<td>3.3</td>
<td>9.96</td>
<td>4.23</td>
<td>4.55</td>
</tr>
<tr>
<td>single</td>
<td>6.56</td>
<td>7.07</td>
<td>5.31</td>
<td>24.19</td>
</tr>
</tbody>
</table>
Issues

- The method does not support indexed geometry
- Primitive/geometry shaders alleviate this!
- Silhouette edges drawn from one side only
- Only noticeable on small meshes
- This is essentially a method for triangles and quadrilaterals
- yes – but that is a very common special case
Merits

• Method is fast
  – Single pass
  – Avoids line primitive
  – Little per-fragment computation

• High quality lines
  – Prefiltered
  – No Z issues

• Tweakable
  – Many line styles are possible
This is a future work teaser.

Acknowledgements:
- Mark Harris, NVIDIA, and Peter-Pike Sloan, Microsoft, were very helpful.
- Microsoft made it possible to test the method on D3D 10 via a Windows Vista Beta.
Line types

4x4 super sampling

Our method

Prefiltering with Gaussian