A Hybrid Method for Computing Apparent Ridges

Eric Jardim
Luiz Henrique de Figueiredo
Evolution of realism in computer graphics

the 60s
Evolution of realism in computer graphics

the 60s

nowadays
Line drawings

- Line drawings are the most basic type of artistic expression
- Compact, clear and effective
- Expressive line drawing of 3D models $\Rightarrow$ NPR problem

Examples of line drawings
Line drawings

- Line drawings are the most basic type of artistic expression
- Compact, clear and effective
- Expressive line drawing of 3D models \( \Rightarrow \) NPR problem
- Apparent ridges is a type of line drawing from 3D models

Examples of line drawings

[Images of line drawings]
Previous work

- There are several line definitions and methods for extraction
- We will review some related curves
- They will be extracted them from the following shaded models
Definition

Normal perpendicular to view direction: \( \langle n, v \rangle = 0 \)

- First-order, view-dependent
- Essential, but don't capture enough information
Ridges & valleys

Definition

Extrema of curvature in principal direction: $D_{e_1} k_1 = 0, |k_1| \geq |k_2|$

- Second-order, not view-dependent
- Angles are too sharp and are visually rigid
Suggestive contours

**Definition**

Zeros of radial curvature in view direction projected onto the tangent plane: $D_w n = 0, \ w = (v)_{T_p} S$

- Second-order, view-dependent
- Naturally extend contours, but don’t appear on convex regions
Apparent ridges

- Second-order, view-dependent
- Appear also on convex regions
- Does not need to be combined with contours
Apparent ridges are based on a new geometric property: View-dependent curvature

- Key idea: measure how the surface bends with respect to the viewpoint, taking into account the perspective transformation
- Plays analogue role for apparent ridges as ordinary curvature does for ridges and valleys
View-dependent curvature

- Let $p \in M$, $n$ normal, $\Pi$ parallel projection and $q = \Pi(p)$
- If $p$ is not a contour point $\Rightarrow \Pi$ is locally invertible
- $J_\Pi$ maps vectors from $T_p M$ to the screen
- Let $\tilde{n}(q) = n \circ \Pi^{-1}(q)$
View-dependent curvature

- The shape operator on $T_p M$ is defined as $S(r) = D_r n$
- Define $Q(s) = D_s \tilde{n}$
- Since $D_s \tilde{n} = D_{r(s)} n = S(r(s))$ and $r = J_{\Pi}^{-1}(s)$ then

$$Q = S \circ J_{\Pi}^{-1}$$
View-dependent curvature

**Definition**

Maximum view-dependent curvature and principal direction

\[ q_1 = \max_{||s||=1} ||Q(s)|| \quad t_1 = \text{direction } ||Q(s)|| \text{ is maximum} \]
Apparent ridges

**Definition**

Apparent ridges := local maxima of $q_1$ in the $t_1$ direction

\[ D_{t_1} q_1 = 0 \quad \text{and} \quad D_{t_1} \left(D_{t_1} q_1 \right) < 0 \]
Contours

- When $p \rightarrow$ contour $\Rightarrow q_1 \rightarrow \infty$
- Extending the definition, $q_1$ achieves a maximum at $\infty$
- Contours are apparent ridges and can be extracted together
Motivation

- Apparent ridges were defined in Judd’s work \(^1\)
- Definition + CPU implementation on triangle meshes
- Excelent visual results compared to other lines, **but...**

---

Motivation

- Apparent ridges were defined in Judd’s work \(^1\)
- Definition + CPU implementation on triangle meshes
- Excellent visual results compared to other lines, but...
- Lower performance compared to other lines

Motivation

- Apparent ridges were defined in Judd’s work\(^1\)
- Definition + CPU implementation on triangle meshes
- Excelent visual results compared to other lines, but...
- Lower performance compared to other lines

Main motivation

Exploit the GPU processing power to speed up extraction

Overview

- Judd’s is an object-space method: compute per-vertex $D_{t_1} q_1$
- Direct approach: port Judd’s code to a GPU vertex shader
- Problem: vertex adjacency ⇒ not easy to port

**Object Space**

1. Compute vd-curvature and its derivative (CPU)
2. Extract the Apparent Ridges lines on the mesh (CPU)
3. Project lines onto the screen
Overview

- Judd’s is an object-space method: compute per-vertex $D_t q_1$
- Direct approach: port Judd’s code to a GPU vertex shader
- Problem: vertex adjacency $\Rightarrow$ not easy to port
- Our solution: split the method in vertex and fragment stages
Object-space stage

- Compute $q_1$ and $t_1$ at each vertex of the mesh
- Ported Judd’s code to vertex shader
- The required data are passed as color and texture
- Since $q_1 \in [0, \infty] \Rightarrow$ scaled and truncated to fit $[0, 1]$

$$q = 2^\tau q_1 \quad \tau \text{ is user-controlled}$$

- Each $t_1$ coordinate $\in [-1, 1] \Rightarrow$ affine transform to $[0, 1]$

$q_1, t_1 \mapsto q, t_x, t_y$
Object-space stage

- Each \((q, t_x, t_y)\) is rasterized as vertex color to an off-screen buffer.
- Usual bilinear interpolation by the graphics pipeline
- The off-screen buffer is the input to the next stage
Image-space stage

- Edge detection to find the maxima of $q_1$ in $t_1$ direction
- Laplacian-like filter $\Rightarrow t_1$ quantized: $\theta \in \{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}\}$
- The $\lambda$ parameter weights the filter in the $\theta$ direction

$$t_1 \in [0 - \frac{\pi}{8}, 0 + \frac{\pi}{8}] \Rightarrow \theta = 0$$  
$$t_1 \in [\frac{\pi}{4} - \frac{\pi}{8}, \frac{\pi}{4} + \frac{\pi}{8}] \Rightarrow \theta = \frac{\pi}{4}$$  
$$t_1 \in [\frac{\pi}{2} - \frac{\pi}{8}, \frac{\pi}{2} + \frac{\pi}{8}] \Rightarrow \theta = \frac{\pi}{8}$$  
$$t_1 \in [\frac{3\pi}{4} - \frac{\pi}{8}, \frac{3\pi}{4} + \frac{\pi}{8}] \Rightarrow \theta = \frac{3\pi}{4}$$
Our results
Our results
Parameter variation $\tau$
Parameter variation $\lambda$
Comparison with Judd’s method

Judd’s

Our
Comparison with Judd’s method

Judd’s  

Our
Comparison with Judd’s method

Judd’s

Our
Comparison with Judd’s method

Judd’s

Our
## Performance comparison on laptop

<table>
<thead>
<tr>
<th>Model</th>
<th># Vertices</th>
<th>Judd’s (FPS)</th>
<th>Ours (FPS)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>roundedcube</td>
<td>1538</td>
<td>202</td>
<td>90</td>
<td>0.5</td>
</tr>
<tr>
<td>torus</td>
<td>4800</td>
<td>148</td>
<td>84</td>
<td>0.6</td>
</tr>
<tr>
<td>tablecloth</td>
<td>22653</td>
<td>32</td>
<td>64</td>
<td>2.0</td>
</tr>
<tr>
<td>hippo</td>
<td>23105</td>
<td>41</td>
<td>66</td>
<td>1.6</td>
</tr>
<tr>
<td>cow</td>
<td>46433</td>
<td>24</td>
<td>58</td>
<td>2.4</td>
</tr>
<tr>
<td>horse</td>
<td>48484</td>
<td>22</td>
<td>52</td>
<td>2.4</td>
</tr>
<tr>
<td>maxplanck</td>
<td>49132</td>
<td>20</td>
<td>46</td>
<td>2.3</td>
</tr>
<tr>
<td>bunny</td>
<td>72027</td>
<td>15</td>
<td>42</td>
<td>2.8</td>
</tr>
<tr>
<td>elephant</td>
<td>78792</td>
<td>14</td>
<td>47</td>
<td>3.4</td>
</tr>
<tr>
<td>golfball</td>
<td>122882</td>
<td>9</td>
<td>30</td>
<td>3.3</td>
</tr>
<tr>
<td>igea</td>
<td>134345</td>
<td>8</td>
<td>29</td>
<td>3.6</td>
</tr>
<tr>
<td>armadillo</td>
<td>172974</td>
<td>5</td>
<td>18</td>
<td>3.6</td>
</tr>
<tr>
<td>column</td>
<td>262653</td>
<td>3</td>
<td>9</td>
<td>3.0</td>
</tr>
<tr>
<td>lucy</td>
<td>262909</td>
<td>4</td>
<td>13</td>
<td>3.3</td>
</tr>
<tr>
<td>heptoroid</td>
<td>286678</td>
<td>4</td>
<td>21</td>
<td>5.3</td>
</tr>
<tr>
<td>brain</td>
<td>294012</td>
<td>4</td>
<td>20</td>
<td>5.0</td>
</tr>
</tbody>
</table>
### Performance comparison on workstation

<table>
<thead>
<tr>
<th>Model</th>
<th># Vertices</th>
<th>Judd’s (FPS)</th>
<th>Ours (FPS)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>roundedcube</td>
<td>1538</td>
<td>600</td>
<td>1100</td>
<td>1.8</td>
</tr>
<tr>
<td>torus</td>
<td>4800</td>
<td>270</td>
<td>870</td>
<td>3.2</td>
</tr>
<tr>
<td>tablecloth</td>
<td>22653</td>
<td>32</td>
<td>160</td>
<td>5.0</td>
</tr>
<tr>
<td>hippo</td>
<td>23105</td>
<td>42</td>
<td>160</td>
<td>3.8</td>
</tr>
<tr>
<td>cow</td>
<td>46433</td>
<td>23</td>
<td>198</td>
<td>8.6</td>
</tr>
<tr>
<td>horse</td>
<td>48484</td>
<td>26</td>
<td>147</td>
<td>5.6</td>
</tr>
<tr>
<td>maxplanck</td>
<td>49132</td>
<td>22</td>
<td>90</td>
<td>4.1</td>
</tr>
<tr>
<td>bunny</td>
<td>72027</td>
<td>16</td>
<td>102</td>
<td>6.4</td>
</tr>
<tr>
<td>elephant</td>
<td>78792</td>
<td>15</td>
<td>113</td>
<td>7.5</td>
</tr>
<tr>
<td>golfball</td>
<td>122882</td>
<td>9</td>
<td>52</td>
<td>5.8</td>
</tr>
<tr>
<td>igea</td>
<td>134345</td>
<td>8</td>
<td>58</td>
<td>7.3</td>
</tr>
<tr>
<td>armadillo</td>
<td>172974</td>
<td>5</td>
<td>30</td>
<td>6.0</td>
</tr>
<tr>
<td>column</td>
<td>262653</td>
<td>3</td>
<td>15</td>
<td>5.0</td>
</tr>
<tr>
<td>lucy</td>
<td>262909</td>
<td>4</td>
<td>26</td>
<td>6.5</td>
</tr>
<tr>
<td>heptoroid</td>
<td>286678</td>
<td>5</td>
<td>19</td>
<td>3.8</td>
</tr>
<tr>
<td>brain</td>
<td>294012</td>
<td>4</td>
<td>34</td>
<td>8.5</td>
</tr>
</tbody>
</table>
### Performance comparison (laptop and workstation)

<table>
<thead>
<tr>
<th>Model</th>
<th># Vert</th>
<th>Judd’s L</th>
<th>Judd’s W</th>
<th>Ours L</th>
<th>Ours W</th>
</tr>
</thead>
<tbody>
<tr>
<td>roundedcube</td>
<td>1538</td>
<td>202</td>
<td>600</td>
<td>90</td>
<td>1100</td>
</tr>
<tr>
<td>torus</td>
<td>4800</td>
<td>148</td>
<td>270</td>
<td>84</td>
<td>870</td>
</tr>
<tr>
<td>tablecloth</td>
<td>22653</td>
<td>32</td>
<td>32</td>
<td>64</td>
<td>160</td>
</tr>
<tr>
<td>hippo</td>
<td>23105</td>
<td>41</td>
<td>42</td>
<td>66</td>
<td>160</td>
</tr>
<tr>
<td>cow</td>
<td>46433</td>
<td>24</td>
<td>23</td>
<td>58</td>
<td>198</td>
</tr>
<tr>
<td>horse</td>
<td>48484</td>
<td>22</td>
<td>26</td>
<td>52</td>
<td>147</td>
</tr>
<tr>
<td>maxplanck</td>
<td>49132</td>
<td>20</td>
<td>22</td>
<td>46</td>
<td>90</td>
</tr>
<tr>
<td>bunny</td>
<td>72027</td>
<td>15</td>
<td>16</td>
<td>42</td>
<td>102</td>
</tr>
<tr>
<td>elephant</td>
<td>78792</td>
<td>14</td>
<td>15</td>
<td>47</td>
<td>113</td>
</tr>
<tr>
<td>golfball</td>
<td>122882</td>
<td>9</td>
<td>9</td>
<td>30</td>
<td>52</td>
</tr>
<tr>
<td>igea</td>
<td>134345</td>
<td>8</td>
<td>8</td>
<td>29</td>
<td>58</td>
</tr>
<tr>
<td>armadillo</td>
<td>172974</td>
<td>5</td>
<td>5</td>
<td>18</td>
<td>30</td>
</tr>
<tr>
<td>column</td>
<td>262653</td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>lucy</td>
<td>262909</td>
<td>4</td>
<td>4</td>
<td>13</td>
<td>26</td>
</tr>
<tr>
<td>heptoroid</td>
<td>286678</td>
<td>4</td>
<td>5</td>
<td>21</td>
<td>19</td>
</tr>
<tr>
<td>brain</td>
<td>294012</td>
<td>4</td>
<td>4</td>
<td>20</td>
<td>34</td>
</tr>
</tbody>
</table>
Conclusion

- Apparent ridges are perceptually pleasant and visually competitive

- We presented a new method that:
  - Replaces estimation of $D_{t1}q_1$ with simple edge detection
  - Performs computations on GPU
  - Produces faster results
  - Provides similar image quality

- Apparent ridges are now even more competitive
Future work

- Enhance image quality:
  - Try other edge detection filters
  - Phong-like shading of $q_1$ and $t_1$

- Modular image-space stage: extract apparent ridges from volume data and implicit models

- Adaptation of the method to extract other lines

- View-dependent curvature: shading and modeling
A Hybrid Method for Computing Apparent Ridges

Eric Jardim
Luiz Henrique de Figueiredo