Fast and Precise: GPU Techniques for 2D

Sean James
Carnegie Mellon University
AutoCAD Product Line, Autodesk Inc.

Ravi Krishnaswamy
Sr. Software Architect
AutoCAD Product Line, Autodesk Inc.
Agenda

- Goals
- Analytical Curves
- Line styles (linetypes)
- Draw order
- Performance
- Conclusion
Background: Project Goal

Improve visual quality and performance of 2D curve rendering, particularly for AutoCAD

Move from performance OR quality to performance AND quality through better use of the GPU
2D: Engineering Documentation
Some Stats
Background: Product Usage

- Many 2D users
- Many DX9 and DX10 feature level users
- Drawing data size typically 0.5-1MB
- Number of 2D entities in the 1000s to 100000s
Considerations

- Productivity: fast open, zoom, pan, etc.
- Visual quality: smooth curves
Analytical Curves
Typically curves are tessellated line segments

Our solution:

- Render “envelope” geometry around curve
- Solve curve equation in pixel shader
Analytical Curves: Benefits

- DX9, OpenGL 3.2 compatibility: no geometry shader tessellation
- Smooth curves and antialiasing at pixel level
- Quality independent of amount of geometry: lower memory/PCIe bandwidth requirement
- High-quality option for users investing in better GPUs
Analytical Curves: Instancing

- Drawing thousands of curves
- GPU instancing:
  - One copy of base geometry
  - GPU duplicates geometry automatically
  - Vertex shader transforms geometry for each instance
- Memory/bandwidth savings and performance improvement over AutoCAD implementation
- Complicates envelope generation and vertex shading
Analytical Curves: Ellipses

- Problem: No closed form for distance from pixel to curve
- Correct solution: find $\theta$ such that error is minimized. Would require iterative solver in pixel shader.

\[ \frac{d}{d\theta} \sqrt{(x - a \cos \theta)^2 + (y - b \sin \theta)^2} = 0 \]

- Approximation: to find distance $d(\vec{p})$ from point $p$ to curve given implicit ellipse function $f$. Can be implemented with ddx/ddy shader instructions. See Blinn/Loop 05 for details.

\[ f(\vec{p} = (x, y)) = \frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 \quad \Rightarrow \quad d(\vec{p}) \approx \frac{f(\vec{p})}{\|\nabla f(\vec{p})\|} \]
Problem: No closed form for arc length. Arc length needed for linetype parameterization (later).

Correct solution: would need to find integral in pixel shader:

\[ s(\theta) = \int_{\theta_0}^{\theta} \sqrt{a^2 \sin^2 \theta + b^2 \cos^2 \theta} \, d\theta \]

Approximation: note that the term being integrated is very similar to the following, which has a simple integral.

\[ s(\theta) \approx \frac{b - a}{2} \cos 2\theta + a + \frac{b - a}{2} \]
Analytical Curves: Ellipses (cont.)

- Problem: Envelope generation for ellipses is non-trivial (tangent, radii depend on $\theta$)
- Approximation: use envelope for a circle and scale to ellipse radii in vertex shader.
- Excess pixel coverage but simple, fast, works with instancing.
Analytical Curves: Performance

- Slow, expensive pixel shading
- Culling: avoid drawing geometry off screen
- Level of detail: balance between envelope geometry complexity and screen size
- Could use an acceleration structure (ex: quad-tree) or geometry shader. Currently CPU only.
Culling/LOD

- Compute bounding box for each instance, and either cull or choose a level of detail and copy into LOD instance buffer
- Copy instance buffers to GPU
- LOD curve based on fraction of screen covered. Avoids shading too many pixels for small objects; grows more slowly for large objects.
Culling/LOD

- Compute bounding box for each instance, and either cull or choose a level of detail
- Copy to GPU
Culling/LOD: Multiple Threads

- Decouples render frame rate and slow LOD/culling/copy
- Worker threads assigned small batches of geometry for culling/LOD to balance work distribution
- Double-buffered level of detail buffers to avoid GPU stalling
- Example (quad-core CPU):

  Worker 0:  
  Worker 1:  
  Worker 2:  
  Render:  

  Swap LOD Instance Buffers
Render Pipeline

- Possible pipeline permutations:

  - **Geometry**
    - Circles, Arcs, Lines
    - Bézier Curves
    - ...

  - **LOD/Cull**
    - CPU (DX9+)
    - Geometry Shader (DX10+)
    - Compute (DX11+)

  - **Tessellate**
    - CPU (DX9+)
    - Geometry Shader (DX10+)
    - Tesellation Shader (DX11+)
    - Compute (DX11+)

  - **Transform**
    - Vertex Shader (DX9+)
    - Geometry Shader (DX10+)

  - **Shade**
    - Analytical Shader (High Qual.)
    - Line Shader (Low Qual.)

  - **Blend**
    - Pre-sort (DX9+)
    - Order Independent (DX9+)
    - Compute (DX11+)
Linetypes
Linetypes: Cases

- 1D and complex linetypes
- Standard and custom linetypes
- Polylines and composite curves
- Linetype scale and zoom invariance
Linetypes: Basic and Complex

- Lines are weighted (2H and HB pencils)
- Dash lengths are paper and scale specific
- Complex linetypes contain symbols and text
Linetypes: pattern adjustments

- End conditions shift the pattern or clamp
- Complex linetypes adjust symbol alignment
Linetype: Approach

- Texture: 2D as array of 1D textures (R,G)
- Use (cumulative) length as parameterization
- Handle end conditions via Phase Shift/Clamping
Linetypes: Dots

- Texel Center u: ‘G’
- Dot or Dash: ‘R’
- Test pixel u distance from Cu in the shader

Light up fixed number of pixels for dots. Use \( u \) and distance from center of texel, \( \text{abs}(pu - Cu) \leq k \cdot du \) where \( k \) = number of pixels to light up, \( pu = u \) param for pixel.
Draw Order and Transparency

- Curves are transparent: need to draw back to front for correctness
- Instancing: cannot interleave drawing different geometry types
- CPU Based: view as a graph problem: Rank Approach
  - Consider the DAG based on overlap and order
  - Topological sort
- GPU Based: No broad support.
  - Consideration of OIT Dx11 SM 5.5
- Static scenarios should consider preprocessing v/s every frame
Rank Algorithm:

- Topological Sort
- Node numbering based on depth
- Batch by type and node numbering
- Bounded by N*OPT where N = number of distinct batch types, OPT = optimal batch count
## Samples and Performance: Pan full scene

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Application</th>
<th>#Lines</th>
<th>#Elliptical Arcs</th>
<th>#Circles</th>
<th>#CircleArcs</th>
<th>GPU%</th>
<th>Average ms/Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stadium</td>
<td>Traditional</td>
<td>301378</td>
<td>529</td>
<td>1265</td>
<td>8350</td>
<td>11.2</td>
<td>65</td>
</tr>
<tr>
<td>Stadium</td>
<td>Analytic</td>
<td>301378</td>
<td>529</td>
<td>1265</td>
<td>8350</td>
<td>92.5</td>
<td>32</td>
</tr>
<tr>
<td>Site</td>
<td>Traditional</td>
<td>140755</td>
<td>533</td>
<td>1956</td>
<td>7699</td>
<td>18.9</td>
<td>45</td>
</tr>
<tr>
<td>Site</td>
<td>Analytic</td>
<td>140755</td>
<td>533</td>
<td>1956</td>
<td>7699</td>
<td>81</td>
<td>19</td>
</tr>
<tr>
<td>Plan</td>
<td>Traditional</td>
<td>104786</td>
<td>540</td>
<td>144</td>
<td>1866</td>
<td>23.6</td>
<td>25</td>
</tr>
<tr>
<td>Plan</td>
<td>Analytic</td>
<td>104786</td>
<td>540</td>
<td>144</td>
<td></td>
<td>76.2</td>
<td>11</td>
</tr>
<tr>
<td>Synthetic-lines-dashed</td>
<td>Traditional</td>
<td>125000</td>
<td></td>
<td></td>
<td></td>
<td>13.4</td>
<td>68</td>
</tr>
<tr>
<td>Synthetic-lines-dashed</td>
<td>Analytic</td>
<td>125000</td>
<td></td>
<td></td>
<td></td>
<td>81.1</td>
<td>4</td>
</tr>
<tr>
<td>Synthetic-circles</td>
<td>Traditional</td>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td>11.4</td>
<td>35</td>
</tr>
<tr>
<td>Synthetic-circles</td>
<td>Analytic</td>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td>92.7</td>
<td>9</td>
</tr>
</tbody>
</table>
GPU Utilization

![Graph showing GPU Utilization with bars for Stadium, Site, Plan, Synthetic-lines-dashed, and Synthetic-circles.](image)
Milliseconds per frame

- Stadium: Analytic 30ms, Traditional 60ms
- Site: Analytic 20ms, Traditional 40ms
- Plan: Analytic 10ms, Traditional 25ms
- Synthetic-lines-dashed: Analytic 5ms, Traditional 15ms
- Synthetic-circles: Analytic 3ms, Traditional 8ms
Extension of linetypes to hatch pattern
Hatch Pattern Definition

*AR-B88, 8x8 Block elevation stretcher bond
0, 0, 0, 0.8
90, 0, 0, 0.4, 0, 0

*AR-8ELM, Standard brick elevation english bond with mortar joints
0, 0, 0, 0.334, 7.625, -3.75
0, 0.25, 0.5, 0.334, 7.625, -3.75
0, 2.667, 0.5, 0.334, 3.625, -3.75
0, 2.667, 0.5, 0.334, 3.625, -3.75
90, 0, 0, 0.8, 2.25, -3.004
90, -0.375, 0, 0.8, 2.25, -3.004
90, 2.667, 0.4, 2.25, -3.004
90, 1.625, 2.5, 0.4, 2.25, -3.004

*AR-BSTD, Standard brick elevation stretcher bond
0, 0, 0, 2.667
90, 0, 2.667, 2.667, 2.667

*AR-CONC, Random dot and stone pattern
Generated as lines with linetypes
Pixel Shader: Identify closest line and linetype
Conclusions

- High quality with performance and capacity gains
- Non traditional behavior for stylization
- Can be implemented with lower shader model requirements
Questions?