Specialized Acceleration Structures for Ray-Tracing

Warren Hunt

Bill Mark
Forward: Flavor of Research

• **Build is cheap** (especially with scan, lazy and build from hierarchy)
  – Grid build and BVH refit are really cheap

• **Many** specialized acceleration structures are required for maximal rendering performance
  – Already true for game rendering algorithms
One System, Many (Layered) Structures

• Static structures for static objects
• Structures with static topology for dynamic objects with static topology
  – Sub-division surfaces
• Structures for collections of dynamic objects or particles
  – Also used in collision detection
• Structures for clusters of highly coherent rays
  – Primary, shadows, large flat reflectors
• Structures for GI (probably not ray-tracing)
Ray-Specialized Acceleration Structures For Ray-Tracing

Warren Hunt
Bill Mark
Introduction

• Use specialization to improve rendering performance in the context of ray-tracing

• Use the **perspective transform** to accelerate (nearly) common origin ray-tracing
  - High performance for **primary and shadow** rays

• Use a simple acceleration structure for fast rebuild (uniform grid)
Motivation

• Ray-tracing lacks competitive performance for primary visibility and shadows

• Z-Buffer lacks flexibility to perform many (even point origin) visibility queries
  – e.g.: hard shadows
  – Uses the perspective transform!

• The space between these is remarkably unpopulated (IZB and ZZ-Buffer are exceptions)
Related Work

• Irregular Z-Buffer [Johnson ‘05]
  – Hard shadow algorithm that uses a perspective grid to store samples

• ZZ-Buffer [Salesin ’89]
  – Uses a screen-space lookup table to store conservative bounding volumes
What is The Perspective Transform?

• Non-affine spatial transform that happens to map lines to lines

• Lines passing through the point of projection are mapped into parallel (axis-aligned) lines in perspective space
Perspective Alignment:

Space: World
Grid: Uniform

Space: World
Grid: Perspective

Space: Perspective
Grid: Uniform
The Math Behind the Transform

- The perspective transform can be described using 3 simple equations:
  - \( x' = x/z \)
  - \( y' = y/z \)
  - \( z' = 1/z \) or \(-1/z\) (preserves order & handedness)

- Maps lines to lines:
  - \( x(z) = Az + B \)
  - \( y(z) = Cz + D \)
The Math Behind the Transform

• The perspective transform can be described using 3 simple equations:
  – \( x' = \frac{x}{z} \)
  – \( y' = \frac{y}{z} \)
  – \( z' = \frac{1}{z} \) or \( -\frac{1}{z} \) (preserves order & handedness)

• Maps lines to lines:
  – \( x(z) = Az + B \rightarrow x'(z') = -Bz' + A \)
  – \( y(z) = Cz + D \rightarrow y'(z') = -Dz' + C \)
The Perspective Singularity

- Problem: Dividing by \( z \) causes problems at zero (the transform has a singularity at zero)

- Solution: Restrict space to \( z^+ \) (or \( z^- \)) (near-plane clipping)
  - Rays are often only going in one direction anyway
  - Cubic arrangement of grids covers all of 3-space (cube-mapping)
Our System Overview:

- Use one perspective grid acceleration structure per light or camera
- Rebuild all grids every frame
Using **Perspective space:**

- Transform all geometry and rays into perspective space (restrict domain to \(z^+\) or \(z^-\))
- We use a uniform grid acceleration structure (in perspective space)
- Perform optimizations for common-origin rays (where applicable)
Point Origin not Required!

Space: World
Grid: Uniform

Space: World
Grid: Perspective

Space: Perspective
Grid: Uniform
Ray-Tracing With the Perspective Grid

- Ray-tracing with the perspective grid can be identical to ray-tracing with a “normal” grid
  - All the same tricks apply

- Additional optimizations for point-origin rays
  - Step only in z’
  - Use projected triangle intersection
System Overview (again):

- Use one perspective grid acceleration structure per light or camera
- Rebuild all grids every frame
Eye-Rays: Two-Level Grid

- Outer level: coarse, stores geometry
  - Doesn’t split in z’ (intersection is really cheap)

- Inner level: fine, stores rays
  - Matched to the cache size of the machine

- Uses back-face culling
Shadow Rays: One Grid per Light

- Fine, stores geometry and minimum depth
  - Uses minimum depth for depth-culling (details in the paper)

- Uses front-face culling
  - Increases minimum depth
  - Avoids shadow ray-launch problem
Results

• We achieve real-time visibility on a desktop PC for primary visibility and hard shadows
  – All scenes are fully dynamic
  – High resolution: 1920x1080

• We demonstrate the ability of a perspective grid to trace off-axis rays at reduced performance
Scenes:

- Courtyard: ~31k Polygons
- FForest020: ~174k Polygons
## Interactive on One Core

<table>
<thead>
<tr>
<th>Scene</th>
<th>Rays</th>
<th>FPS</th>
<th>Build</th>
<th>Mray/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courtyard</td>
<td>Primary</td>
<td>30</td>
<td>5%</td>
<td>68</td>
</tr>
<tr>
<td>FForest20</td>
<td>Primary</td>
<td>17</td>
<td>17%</td>
<td>39</td>
</tr>
<tr>
<td>Courtyard</td>
<td>P+Hard</td>
<td>8</td>
<td>10%</td>
<td>36</td>
</tr>
<tr>
<td>FForest20</td>
<td>P+Hard</td>
<td>6</td>
<td>33%</td>
<td>27</td>
</tr>
<tr>
<td>Courtyard</td>
<td>P+Soft</td>
<td>0.26</td>
<td>0%</td>
<td>5.3</td>
</tr>
<tr>
<td>FForest20</td>
<td>P+Soft</td>
<td>0.11</td>
<td>1%</td>
<td>2.3</td>
</tr>
</tbody>
</table>
### Real Time on 8 Cores

<table>
<thead>
<tr>
<th>Scene</th>
<th>Rays</th>
<th>FPS</th>
<th>Build</th>
<th>Mray/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courtyard</td>
<td>Primary</td>
<td>150</td>
<td>27%</td>
<td>342</td>
</tr>
<tr>
<td>FForest20</td>
<td>Primary</td>
<td>55</td>
<td>56%</td>
<td>125</td>
</tr>
<tr>
<td>Courtyard</td>
<td>P+Hard</td>
<td>34</td>
<td>41%</td>
<td>155</td>
</tr>
<tr>
<td>FForest20</td>
<td>P+Hard</td>
<td>14</td>
<td>78%</td>
<td>64</td>
</tr>
<tr>
<td>Courtyard</td>
<td>P+Soft</td>
<td>2</td>
<td>2%</td>
<td>41</td>
</tr>
<tr>
<td>FForest20</td>
<td>P+Soft</td>
<td>.5</td>
<td>4%</td>
<td>15</td>
</tr>
</tbody>
</table>
Results Comparison (In the paper)

- As fast or faster than MLRTA for primary visibility (our times include build)
- Faster than other fully dynamic ray-tracers for eye and hard shadow rays
- Competitive with a commercial software rasterizer for triangle rate and transform rate
Conclusions: Take Away Messages

- A continuum of structures/algorithms exist between ray-tracing and rasterization
  - Grid $\rightarrow$ ... $\rightarrow$ Perspective Grid $\rightarrow$ ... $\rightarrow$ Z-buffer
- Perspective Grid is a very effective, drop in hard-shadow algorithm using ray-tracing
  - Also competitive with z-buffer for primary visibility
- Many special case acceleration structures are often faster than a single generic one
Future Work

- Multi-threaded grid build
- Lots of applicable shadow-mapping research
- The perspective grid performs relatively poorly for off-axis rays:
  - Largely due to “teapot in a stadium” effects
  - Addressed with adaptive acceleration structures