Enhancing Traditional Rasterization Graphics with Ray Tracing

March 2015
Introductions

- **James Rumble**
  - Developer Technology Engineer
  - Ray Tracing Support

- **Justin DeCell**
  - Software Design Engineer
  - Ray Tracing Visualizer
Overview

- PowerVR Ray Tracing Introduction:
  - What is Ray Tracing?
  - PowerVR Ray Tracing Pipeline
  - Hybrid Ray Tracing
- Fast Ray Traced Soft Shadows
- Conclusion
What is Ray Tracing?
Tracing Rays

Forward Ray Tracing vs Backward Ray Tracing

- Rays used to model real-world behaviour of light.

- Forward Ray Tracing:
  - Rays traced from the light source.
  - Many rays never reach our eyes.

- Backward Ray Tracing:
  - Rays traced from the “eye” towards light.
Tracing Rays

Forward Ray Tracing vs Backward Ray Tracing

- Rays used to model real-world behaviour of light.

- **Forward Ray Tracing:**
  - Rays traced from the light source.
  - Many rays never reach our eyes.

- **Backward Ray Tracing:**
  - Rays traced from the “eye” towards light.
Tracing Rays

Forward Ray Tracing vs Backward Ray Tracing

- Rays used to model real-world behaviour of light.

- **Forward Ray Tracing:**
  - Rays traced from the light source.
  - Many rays never reach our eyes.

- **Backward Ray Tracing:**
  - Rays traced from the “eye” towards light.
Tracing Rays
Forward Ray Tracing vs Backward Ray Tracing

- Rays used to model real-world behaviour of light.

- Forward Ray Tracing:
  - Rays traced from the light source.
  - Many rays never reach our eyes.

- Backward Ray Tracing:
  - Rays traced from the “eye” towards light.
Tracing Rays
Forward Ray Tracing vs Backward Ray Tracing

- Rays used to model real-world behaviour of light.

- **Forward Ray Tracing:**
  - Rays traced from the light source.
  - Many rays never reach our eyes.

- **Backward Ray Tracing:**
  - Rays traced from the “eye” towards light.
Tracing Rays

Forward Ray Tracing vs Backward Ray Tracing

- Rays used to model real-world behaviour of light.

- **Forward Ray Tracing:**
  - Rays traced from the light source.
  - Many rays never reach our eyes.

- **Backward Ray Tracing:**
  - Rays traced from the “eye” towards light.
Tracing Rays

Forward Ray Tracing vs Backward Ray Tracing

- Rays used to model real-world behaviour of light.

- Forward Ray Tracing:
  - Rays traced from the light source.
  - Many rays never reach our eyes.

- Backward Ray Tracing:
  - Rays traced from the “eye” towards light.
What is Ray Tracing?

**Backwards Ray Tracing**

- Primary rays fired from “eye” through pixels of the screen.
- Find closest triangle intersection.
What is Ray Tracing?

Ray Tracing provides the ability for the shading of one object to be aware of the geometry of other objects in the scene.
PowerVR Ray Tracing Pipeline
PowerVR Ray Tracing Pipeline

Scene Hierarchy Building Phase
- Vertex Processing
- Hierarchy Generation
- Memory
  - Geometry Data
  - Scene Hierarchy

Ray Traversal Phase
- Primary Ray Generation
- Intersection Tests
- Ray Shaders
- Accumulation
Scene Hierarchy Generation

Geometry Submission

Scene Hierarchy Building Phase

- Vertex Processing
- Hierarchy Generation

Ray Traversal Phase

- Primary Ray Generation
- Intersection Tests
- Ray Shaders
- Accumulation

Memory

- Geometry Data
- Scene Hierarchy

Geometry Submission

- Scene Components
- Scenes
Scene Hierarchy Generation

Scene Components

Windshield
- Program:
  - Vertex Shader
  - Ray Shader
  - Uniforms
- Geometry Data
- Buffers
- Textures

Car Body
- Program:
  - Vertex Shader
  - Ray Shader
  - Uniforms
  - Geometry Data
  - Buffers
  - Textures

Headlights
- Program:
  - Vertex Shader
  - Ray Shader
  - Uniforms
  - Geometry Data
  - Buffers
  - Textures

Chrome
- Program:
  - Vertex Shader
  - Ray Shader
  - Uniforms
  - Geometry Data
  - Buffers
  - Textures

Tyres
- Program:
  - Vertex Shader
  - Ray Shader
  - Uniforms
  - Geometry Data
  - Buffers
  - Textures
Scene Hierarchy Generation

**Scenes**

- Contain groups of components.
- Ray traversal occurs within scene.
Scene Hierarchy Generation

**Vertex Processing**

**Scene Hierarchy Building Phase**
- Vertex Processing
- Hierarchy Generation

**Memory**
- Geometry Data
- Scene Hierarchy

**Ray Traversal Phase**
- Primary Ray Generation
- Intersection Tests
- Ray Shaders
- Accumulation

---

facebook.com/imgtec    @PowerVRInsider | #idc15
Scene Hierarchy Generation

Vertex Processing

- Vertex shader used to transform geometry to world space.
Scene Hierarchy Generation

Hierarchy Generation

Scene Hierarchy Building Phase
- Vertex Processing
- Hierarchy Generation

Memory
- Geometry Data
- Scene Hierarchy

Ray Traversal Phase
- Primary Ray Generation
- Intersection Tests
- Ray Shaders
- Accumulation

Hierarchy Generation
Scene Hierarchy Generation

**Hierarchy Generation**

- Organises triangles based on their size and position.
- Application developer does not need to worry about handling this.
- Simple “Build” command.
Scene Hierarchy Generation

Hierarchy Generation

- Organises triangles based on their size and position.
- Application developer does not need to worry about handling this.
  - Simple “Build” command.
Scene Hierarchy Generation

Hierarchy Generation

- Organises triangles based on their size and position.
- Application developer does not need to worry about handling this.
- Simple “Build” command.
Scene Hierarchy Generation

Organises triangles based on their size and position.

Application developer does not need to worry about handling this.

Simple “Build” command.
Scene Ray Traversal

Scene Hierarchy Building Phase
- Vertex Processing
- Hierarchy Generation
- Memory
- Geometry Data
- Scene Hierarchy

Ray Traversal Phase
- Primary Ray Generation
- Intersection Tests
- Ray Shaders
- Accumulation
Scene Ray Traversal

Primary Ray Generation

Scene Hierarchy Building Phase
- Vertex Processing
- Hierarchy Generation

Memory
- Geometry Data
- Scene Hierarchy

Ray Traversal Phase
- Primary Ray Generation
- Intersection Tests
- Ray Shaders
- Accumulation

Scene Hierarchy Building Phase

Primary Ray Generation

Intersection Tests

Ray Shaders

Accumulation
Scene Ray Traversal

Primary Ray Generation

- **Frame Shaders:**
  - Run once per pixel to emit primary rays into the scene.
  - Common case – fire a ray per-pixel over a full-screen quad.
  - Alternatively – Frame shader can be setup as any m x n dimension.

```
Scene Ray Traversal

Intersection Tests

Scene Hierarchy Building Phase
- Vertex Processing
- Hierarchy Generation

Memory
- Geometry Data
- Scene Hierarchy

Ray Traversal Phase
- Primary Ray Generation
- Intersection Tests
- Ray Shaders
- Accumulation
Scene Ray Traversal

Intersection Tests

- Simple approach – each ray tested against all triangles:
  - Brute force.
  - \(m \times n\) intersection tests (\(m\) – rays, \(n\) – objects).
- Use accelerated data structure to reduce intersection tests.
  - Application developer does not need to worry about handling this.
  - Accelerated triangle and box testing.
Scene Ray Traversal

**Intersection Tests**

- Simple approach – each ray tested against all triangles:
  - Brute force.
  - \( m \times n \) intersection tests (\( m \) – rays, \( n \) – objects).
- Use accelerated data structure to reduce intersection tests.
  - Application developer does not need to worry about handling this.
  - Accelerated triangle and box testing.
Scene Ray Traversal

**Intersection Tests**

- Simple approach – each ray tested against all triangles:
  - Brute force.
  - $m \times n$ intersection tests ($m$ – rays, $n$ – objects).
- Use accelerated data structure to reduce intersection tests.
  - Application developer does not need to worry about handling this.
  - Accelerated triangle and box testing.
Scene Ray Traversal

Intersection Tests

Simple approach – each ray tested against all triangles:

- Brute force.
- \(m \times n\) intersection tests (\(m\) – rays, \(n\) – objects).

Use accelerated data structure to reduce intersection tests.

- Application developer does not need to worry about handling this.
- Accelerated triangle and box testing.
Scene Ray Traversal

Ray Shaders and Accumulation

Scene Hierarchy Building Phase

- Vertex Processing
- Hierarchy Generation
- Memory
  - Geometry Data
  - Scene Hierarchy

Ray Traversal Phase

- Primary Ray Generation
- Intersection Tests
- Ray Shaders
- Accumulation
Scene Ray Traversal

Ray Shaders and Accumulation

- **Ray Shaders:**
  - Defined for each component and invoked when a ray intersects a triangle.
  - Defines material properties of component.
  - Can emit further rays causing additional intersection tests.

```plaintext
accumulate(0.85, 0.85, 0.75);
reflect(incidentDirection, normal)
```
Pipeline Summary

Scene Hierarchy Building Phase
- Vertex Processing
- Hierarchy Generation
- Memory
- Scene Hierarchy
- Geometry Submission
  - Scene Components
  - Scenes

Ray Traversal Phase
- Primary Ray Generation
- Intersection Tests
- Ray Shaders
- Accumulation

Geometry Data

Scene Components
Scenes

Pipeline Summary

Scene Hierarchy Building Phase
- Vertex Processing
- Hierarchy Generation
- Memory
  - Geometry Data
  - Scene Hierarchy

Ray Traversal Phase
- Primary Ray Generation
- Intersection Tests
- Ray Shaders
  - Accumulation

Geometry Submission
- Scene Components
- Scenes
Pipeline Summary

Scene Hierarchy Building Phase
- Vertex Processing
- Hierarchy Generation
- Memory
  - Geometry Data
  - Scene Hierarchy

Ray Traversal Phase
- Primary Ray Generation
- Intersection Tests
- Ray Shaders
- Accumulation

Geometry Submission
- Scene Components
- Scenes
Hybrid Ray Tracing
How do we use it?

- Hybrid Shadows, Reflections, etc.
- Augmented Reality
- Production-Quality Renders
- Order-Independent Transparency
- Ambient Occlusion
- Asset creation / compression
- Global Illumination
- Physics & Collision Detection
- Virtual Reality
- Lens correction, Ultra-low latency rendering, Lenticular Displays
- A.I. & Line of Sight Calculations
- Rapid photo-quality output
How do we use it?

Light Maps and Probe Baking In Unity Editor 5
How do we use it?

*Hybrid Ray Traced Graphics*
Modern raster-based game engine

Deferred shading used in many modern game engines

Render the scene into a deep frame buffer (G-Buffer)

Compute Lighting on each G-Buffer pixel (No Overdraw)

Assemble the final frame and perform screen-space effects

Lighting Buffer(s)

Lighting calculations executed in screen space

G-Buffer

Material IDs
Normals
World-Space Positions or depth buffer

Final Result Frame

Lighting calculations executed in screen space

Modern raster-based game engine

Deferred shading used in many modern game engines

Render the scene into a deep frame buffer (G-Buffer)

Compute Lighting on each G-Buffer pixel (No Overdraw)

Assemble the final frame and perform screen-space effects

Lighting Buffer(s)

Lighting calculations executed in screen space

G-Buffer

Material IDs
Normals
World-Space Positions or depth buffer

Final Result Frame

Lighting calculations executed in screen space
Ray Trace using the G-Buffer

Who needs primary rays when we have rasterization?

G-Buffer

- Material IDs: (1.0,0.0,0.0)
- Normals: (0.5,0.6,-0.2)
- World Space Positions or Depth values: (14.3,81.6,-0.4)
Ray Trace using the G-Buffer

Who needs primary rays when we have rasterization?

G-Buffer

| Material IDs | (1.0,0.0,0.0) |
| Normals      | (0.5,0.6,-0.2) |
| World Space Positions or Depth values | (14.3,81.6,-0.4) |
Hybrid Ray Tracing

Use the G-Buffer to setup your rays

- Render the scene into a deep frame buffer (G-Buffer)
- Trace Rays to determine lighting
- Assemble the final frame and perform screen-space effects

G-Buffer
- Material IDs
- Normals
- World-Space Positions or depth buffer

Lighting Buffer(s)

Final Result Frame
Fast Ray Traced Soft Shadows
Cascaded Shadow Maps

State of the art non-ray tracing technique

- Render a shadow map slice for each light at each resolution
Cascaded Shadow Maps

State of the art non-ray tracing technique

- Render a shadow map slice for each light at each resolution
Cascaded Shadow Maps

State of the art non-ray tracing technique

- Render a shadow map slice for each light at each resolution
Cascaded Shadow Maps

*State of the art non-ray tracing technique*

- Render a shadow map slice for each light at each resolution
Cascaded Shadow Maps

State of the art non-ray tracing technique

- Render a shadow map slice for each light at each resolution
Cascaded Shadow Maps

State of the art non-ray tracing technique

- More shadow detail is stored for objects near to the view

100 texels per scene unit
50 texels per scene unit
25 texels per scene unit
12 texels per scene/unit
Cascaded Shadow Maps

State of the art non-ray tracing technique

- Sample from the appropriate slices based on distance from view
Cascaded Shadow Maps
State of the art non-ray tracing technique

- Sample from the appropriate slices based on distance from view
Comparison with Cascaded Shadow Maps

State of the art non-ray tracing technique

- Sample from the appropriate slices based on distance from view
Comparison with Cascaded Shadow Maps

*State of the art non-ray tracing technique*

- Sample from the appropriate slices based on distance from view
Ray Traced Shadows

*The basic algorithm for shadowing with ray tracing*

- Shoot a ray between surface and the light:
  - If the ray hits *anything* then do nothing (region is shadowed and unlit)
  - If the ray reaches the light without hitting anything then illuminate that pixel
- Then you’re done!
Soft Shadows

*Shadows with soft edges*

- Shadows have sharp edges at noon on a clear day…
- But many situations cause soft shadows
- The region where a shadow transitions between fully and partially lit is called the penumbra.
- In the real world
  - Light sources are not infinitely small points
  - Scattering occurs between the light source and surface
A Simplified Model of Soft Shadows

*Penumbra size is based on the ratio of the distance to an occluder and the light*
A Simplified Model of Soft Shadows

Knowing the distance to an object, we can calculate the softness of its shadow.
A Simplified Model of Soft Shadows

By knowing the distance to an object, we can calculate the softness of the shadow it casts.
Algorithm: Distance-To-Occluder Soft Shadows

Analytical penumbra calculation based on distance to occluder

- Shadow rays don’t just record hit vs. miss, they also record the distance to the occluder on each hit
  - After the ray tracing pass, final pixel’s shadow density is calculated with a screen-space filter
  - Pixels in shadow select a filter kernel size based on the stored distance to the occluder
  - Pixels in light perform a limited search to identify the nearby shadows that could darken the pixel
Algorithm: Distance-To-Occluder Soft Shadows

Analytical penumbra calculation based on distance to occluder

- Easily extended to support transparent objects
  - In addition to distance, store a shadow density value for each pixel that hits an object
    - Only store the distance to the first occluder we encounter
  - Continue ray towards light until we reach the light or shadow density is saturated, i.e. pixel is fully shadowed

Discard Ray

Store distance Density: 0.5

Density: 1
Discard Ray

Store distance Density: 0.5
Ray Traced Shadows
Ray Tracing Density Buffer
Ray Tracing Distance Buffer
Penumbra Size Calculation

Distance selection – in shadow

- If current pixel in shadow, use current pixel’s distance to occluder value

```
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
  5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
  6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
  7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
```
Penumbra Size Calculation

*Distance selection – not in shadow*

- Search for shadowed neighboring pixels using “cross” pattern
- Select the maximum distance to occluder

```
  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
---+---+---+---+---+---+---+---+---+
  | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
---+---+---+---+---+---+---+---+---+
  | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
---+---+---+---+---+---+---+---+---+
  | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
---+---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   | 7 |
```

facebook.com/imgtec  @PowerVRInsider  #idc15
Penumbra Size Calculation

Analytically compute penumbra size

- Compute penumbra width ($P$), using distance to occluder ($O$), distance to the light source ($L$), and the light radius ($R$)

$$P = \frac{RO}{L}$$

Note: Given an sufficiently far away light, $R/L$ can be treated as a constant.
Kernel Width Selection

Project penumbra width into screen space
Depth Rejection

Preserve discontinuous edges

- Local differencing to compute partial derivatives with respect to screen space
  \[
  \frac{\partial z}{\partial x} = \min(|d_{1,0} - d_{0,0}|, |d_{-1,0} - d_{0,0}|) \\
  \frac{\partial z}{\partial y} = \min(|d_{0,1} - d_{0,0}|, |d_{0,-1} - d_{0,0}|)
  \]

- Compute depth delta between center pixel and pixel sampling, accept sample if within following threshold:
  \[
  |\Delta d_{x,y}| \leq p_x \cdot \frac{\partial z}{\partial x} + p_y \cdot \frac{\partial z}{\partial y} + \varepsilon
  \]
Depth Rejection

Preserve discontinuous edges

- Local differencing to compute partial derivatives with respect to screen space

\[
\frac{\partial z}{\partial x} = \min(|d_{1,0} - d_{0,0}|, |d_{-1,0} - d_{0,0}|)
\]

\[
\frac{\partial z}{\partial y} = \min(|d_{0,1} - d_{0,0}|, |d_{0,-1} - d_{0,0}|)
\]

- Compute depth delta between center pixel and pixel sampling, accept sample if within following threshold:

\[
|\Delta d_{x,y}| \leq p_x \ast \frac{\partial z}{\partial x} + p_y \ast \frac{\partial z}{\partial y} + \varepsilon
\]
Algorithm: Distance-To-Occluder Soft Shadows

Analytical penumbra calculation based on distance to occluder

**Advantages:**
- No shadow map resolution issues; shadow resolution based on screen resolution
- Perfect shadow “contact”; no “Peter Panning”
- No noise or banding
- Physically plausible penumbra size
- More efficient than cascaded shadow maps or multi-ray approaches
  - At most, casts 1 ray per pixel, often less

**Limitations:**
- Visible penumbra size is limited by the kernel size
- A separate buffer or pass is needed for each light, just like shadow maps
  - Best for daylight scenes with one dominant light, i.e. the sun
Results

4 slice, 2k, cascaded shadow maps
Results

Ray traced shadows
Results

Over blurring

Ray traced

Cascaded shadow maps
Results

4 slice, 2k, cascaded shadow maps
Results

Ray traced shadows
Results

Floating geometry/Peter Panning

Ray traced

Cascaded shadow maps
Results

4 slice, 2k, cascaded shadow maps
Results

Ray traced shadows
Results

Floating geometry/Peter Panning

Ray traced

Cascaded shadow maps
Optimization and Efficiency Analysis
Cast Fewer Rays

\( N \cdot L \) rejection

- If a surface is facing away from the light, there’s no need to cast any rays!
  - A surface is back facing with respect to the light when \( \text{dot}(N, L) \leq 0.0 \)
  - Store rejected values as 0.0 in distance buffer, 1.0 in density buffer
- In the worst case scenario, we still need to cast 1 ray per pixel
- Most cases we’ll be able to reject some number of pixels giving us shadows for < 1 ray per pixel
Rending Pipeline

*Initial implementation*

G-Buffer
- Normal RGBA32F
- Position RGBA32F
- Albedo RGBA8
- Depth 24

Ray Tracing
- Distance/Density RG16F

Filtering + Blit

facebook.com/imgtec  @PowerVRInsider  #idc15
Rending Pipeline

Better implementation

G-Buffer
- Packed Normal R32UI
- Position RGBA32F
- Albedo RGBA8
- Depth 24

Ray Tracing
- Distance/Density RG8

Filtering + Blit
Rending Pipeline

Optimized implementation

G-Buffer

- Packed Normal R32UI
- Position RGBA32F
- Albedo + N-L RGBA8
- Depth 24

Framebuffer Fetch

- Distance/Density RG8

Ray Tracing

Filtering + Blit
What is Bandwidth?

*Why you should care about bandwidth*

- Bandwidth consumption is a result of data that must be fetched from main memory (DRAM)
  - Bandwidth = (memory reads + memory writes) / time
- Texture fetches that miss the cache must read the value from DRAM
  - DRAM fetch is a high latency operation and stalls the shader
  - Each DRAM fetch consumes power

```
vec4 value = texture(uTex, uv);
```
Efficiency Analysis

Bandwidth for 1280x720 render
# Efficiency Analysis

*Bandwidth for 1280x720 render*

## Total Memory Traffic (Reads + Writes)

<table>
<thead>
<tr>
<th></th>
<th>Cascaded Shadow Maps</th>
<th>Ray Traced Shadows</th>
<th>G-Buffer Only</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>233.71 MB</strong></td>
<td>164.34 MB</td>
<td>97.01 MB</td>
<td></td>
</tr>
</tbody>
</table>
### Efficiency Analysis

*Bandwidth for 1280x720 render*

<table>
<thead>
<tr>
<th>Total Memory Traffic (Reads + Writes)</th>
<th>Cascaded Shadow Maps</th>
<th>Ray Traced Shadows</th>
<th>G-Buffer Only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>233.71 MB</td>
<td>164.34 MB</td>
<td>97.01 MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Shadowing Memory Traffic</th>
<th>Cascaded Shadow Maps</th>
<th>Ray Traced Shadows</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>136.70 MB</td>
<td>67.32 MB</td>
</tr>
</tbody>
</table>
Efficiency Analysis
Bandwidth for 1280x720 render

<table>
<thead>
<tr>
<th>Total Memory Traffic (Reads + Writes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascaded Shadow Maps</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>233.71 MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Shadowing Memory Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascaded Shadow Maps</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>136.70 MB</td>
</tr>
</tbody>
</table>

50.7% Shadowing Memory Traffic Reduction
# Efficiency Analysis

*Bandwidth for 1280x720 render*

<table>
<thead>
<tr>
<th>Cascaded Shadows</th>
<th>Ray Traced Shadows*</th>
<th>Memory Traffic Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>136.70 MB</td>
<td>67.32 MB</td>
<td>50.7%</td>
</tr>
<tr>
<td>150.94 MB</td>
<td>62.49 MB</td>
<td>58.6%</td>
</tr>
<tr>
<td>159.48 MB</td>
<td>86.51 MB</td>
<td>45.8%</td>
</tr>
</tbody>
</table>

*Excludes 61.06 MB one time setup cost for building scene hierarchy*
## Efficiency Analysis

*Cycles for 1280x720 render @ 600Mhz*

<table>
<thead>
<tr>
<th>Cascaded Shadows</th>
<th>Ray Traced Shadows</th>
<th>Total Time Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Cascaded Shadows" /></td>
<td><img src="image2.png" alt="Ray Traced Shadows" /></td>
<td><img src="image3.png" alt="Total Time Reduction" /></td>
</tr>
<tr>
<td>27,582,732 cycles</td>
<td>13,585,924 cycles</td>
<td>23.3 ms/frame 50.7%</td>
</tr>
<tr>
<td>45.97 ms/frame</td>
<td>22.64 ms/frame</td>
<td></td>
</tr>
<tr>
<td>31,643,530 cycles</td>
<td>14,145,497 cycles</td>
<td>29.2 ms/frame 55.3%</td>
</tr>
<tr>
<td>52.74 ms/frame</td>
<td>23.58 ms/frame</td>
<td></td>
</tr>
<tr>
<td>37,842,299 cycles</td>
<td>18,975,421 cycles</td>
<td>31.4 ms/frame 49.6%</td>
</tr>
<tr>
<td>63.07 ms/frame</td>
<td>31.63 ms/frame</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- Ray tracing in conjunction with rasterization
- Shown specific example using ray tracing to improve visual quality and performance
- More examples of ray tracing in action this afternoon
  - Low Overhead Probe-based Global Illumination Using Ray Tracing @ 1pm