Lecture 11: Interactive Rendering
Chapters 7 in Advanced GI

Fall 2004
Kavita Bala
Computer Science
Cornell University

HW 1

• Questions?
Interactive Software Rendering

• Interactive
  – User-driven, not pre-scripted animation
  – At least a few frames per second (fps)

• Software
  – Major shading done in software
    ▪ Can use hardware to help

• Rendering
  – Online, not pre-computed or captured
    ▪ Eg, lightfields are pre-computed

An Oxymoron?

• Why not just use hardware?
  – The games all use it
    – It has lots of cool effects
    – Isn’t software too slow?
Fast interpolation from vertices

- Specify properties at vertices
  - Color
  - Texture coordinates
  - Surface normals, etc.
- Interpolate at each pixel in triangle

- But: average triangle size is decreasing
  - Many more visible triangles than pixels, therefore, interpolation less valuable

Fast Visibility Determination

- z-buffer: Amortized performance
  - One rendering determines visible surfaces for all pixels simultaneously
Fast Visibility Determination

- Great for some visibility queries types
  - Primary (eye) rays
  - Shadow rays (point sources)
- Not so good for other types
  - Shadow rays (area light sources)
  - Many lights
  - Reflection & refraction from curved surfaces
  - Indirect illumination
  - Adaptive sampling

Fast Shading

- Latest boards can do per-pixel shading
- Programmable
- Local shading only
  - All inputs must be provided ahead of time
  - Non-local shading can only be approximated
    - Shadows, reflections, indirect, etc
Why Software Rendering?

- Global Illumination: Non-local information
- Extremely high complexity
- Arbitrary shading models
- Portability
  - No tweaking: just works
  - No scene dependent optimizations

Hardware Vs. Software

- Hardware still has the edge due to its dedicated pipeline
- Software attractive for its scalability and flexiblility
  - If it can be made “fast enough” for interactive use
  - And handle scene and/or effects the hardware cannot handle
Ray Tracing (or Ray Casting)

• Common visibility tool for software

• Flexible

• Efficient for large models
  – Using an acceleration structure (grids, bsp, etc)

• Usually the largest computational bottleneck

• Easily parallelizable: each pixel in parallel

Interactive RT  (Parker et. al.)

• SGI Origin 2000
  – 64 processors
  – Shared memory

• Whitted-style ray tracing
  – Shadow, reflection, and refraction rays

• Non-polygonal primitives
  – Spheres and splines

© Kavita Bala, Computer Science, Cornell University
Interactive RT: (Parker et al.)

- Dynamic load balancing
  - Tasks divided by screen region
  - Sequence of larger to smaller tasks
  - Found near ideal parallel speedup

Coherent Ray Tracing (Wald et. al.)

- Highly optimized ray tracing engine for Intel-based PCs
- Carefully profiled their ray tracer
  - Discovered it was often memory bound
- Hand-crafted and tuned their code
  - Both C and assembly versions
  - Compact, cache-friendly data structures
  - Optimized for SIMD (SSE)
  - Reordered computations for better coherence
Coherent Ray Tracing (Wald et. al.)

• Optimizations
  – Separated data based on use
    ▪ Data needed for intersection stored separately
  – Used compact axis-aligned BSP structure
  – Cache aligned data
  – Works on groups of four rays at a time
    ▪ Allows for efficient use of SIMD (SSE)

• Limitations
  – Restricted to triangles only
  – Optimized for Phong shading specifically

Test Scenes I

• Test scenes: 800 triangles to >8 million

Office:
34,000 triangles, 3 lights

Conference Room:
280,000 triangles, 2 lights
Test Scenes II

- Test scenes: 800 triangles to >8 million

Berkeley Soda Hall:
1.5 to 8 million triangles

Terrain:
1 million triangles (textured)

Performance Results I

- Comparison to Rasterization-Hardware
  - Rasterization: IRIS Performer
  - RTRT: 512x512 Pixel, 1 CPU (PIII-800MHz)

<table>
<thead>
<tr>
<th>Scene (triangles)</th>
<th>Octane</th>
<th>Onyx3</th>
<th>NVidia</th>
<th>RTRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Office (40k)</td>
<td>&gt;24fps</td>
<td>&gt;36fps</td>
<td>12.7fps</td>
<td>1.8fps</td>
</tr>
<tr>
<td>Theatre (680k)</td>
<td>0.4fps</td>
<td>6-12fps</td>
<td>1.5fps</td>
<td>1.1fps</td>
</tr>
<tr>
<td>Library (907k)</td>
<td>1.5fps</td>
<td>4fps</td>
<td>1.6fps</td>
<td>1.1fps</td>
</tr>
<tr>
<td>5th floor (2.5M)</td>
<td>0.5fps</td>
<td>1.5fps</td>
<td>0.6fps</td>
<td>1.5fps</td>
</tr>
<tr>
<td>Soda Hall (8M)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.8fps</td>
</tr>
</tbody>
</table>
Performance Results II

- Comparison to Rasterization-Hardware
  - Ray tracing scales well for large environments

Optimizations (Wald et. al.)

- Also parallelized for more speed
  - Demonstrated on five dual processor PIIIs

- Times are for primary rays only
  - Adding shadows, reflections, etc. adds costs

- Considerable speedup
  - But, more is needed, especially for more complex shading such as:
    - soft shadows, glossy reflections, and indirect illumination

© Kavita Bala, Computer Science, Cornell University
Sample Images
Upshot

• Software Interactive Rendering is possible now with current machines
  – Good scaling with scene complexity
  – Greater shading flexibility

• Many more challenges still remain
  – Higher resolutions
  – Anti-aliasing
  – Fully dynamic environments
  – Global Illumination
  – Complex lighting

Interactive Global Illumination
Rendering as Sampling

- Ray tracers compute radiance at each pixel

Rendering = Sampling radiance

Coherence

- Within one frame: spatial coherence

Across many frames: temporal coherence
Interactive Global Illumination

- Need to render every pixel of every frame?

- Exploit coherence in radiance
  – object space, image space, temporal

Strategy

- Insight: radiance is mostly smooth -- use sparse sampling and reconstruction

- Radiance samples are very expensive
- Goal: reconstruct most pixels by interpolation
- Issues: discontinuities, non-linear variations
Interactive Global Illumination

- Fast feedback
- Must bridge gap in framerate
- Interactive requirements
  - Image quality
  - Responsiveness
    - Don’t make the user wait
    - Provide rapid user feedback
  - Consistency
    - Don't surprise or distract the user
    - Avoid sudden changes if possible
      - Eg, in quality, frame rate, popping, etc.

Visual Feedback Loop

- Standard visual feedback loop
  - Entirely synchronous
  - Frame-rate limited by renderer

© Kavita Bala, Computer Science, Cornell University
Modified Visual Feedback Loop

Display Process

- Automatically exploit spatial and temporal coherence
- Layered on top of an existing (slow) global illumination renderer
- Provide interactive performance
Sparse Sampling Approaches

- 4D:
  - Radiance Interpolants
  - Holodeck

- 2D: Image based
  - Post-rendering Warp
  - Render Cache
  - Edge and Point Rendering

Radiance Interpolants Bala(96,97,99)

- Radiance interpolant:
  - set of sparse samples of radiance function that allow accurate reconstruction

- Note x-axis is the axis of rays
Interpolant for Approximation

- Approximate radiance with conservative error bounds: accelerate ray tracing
  - Exploits spatial and temporal coherence
  - On-line: no preprocessing

Ray Parameterization

- Radiance is computed along a ray
- In 2D, ray can be parameterized by (a,c)
Dual Space

• Space of rays is called line space
• It is a dual space
• Every ray in world space is a point in line space
• And vice-versa

Linespace and Interpolants

• Bilinear interpolation
  \[ R = (1-a)(1-c) R_{00} + a (1-c) R_{10} + c (1-a) R_{01} + a c R_{11} \]
• Radiance interpolant: set of four radiance samples for region of line space
Data structure: Linetree

- Linetree stores interpolants for each object
- Hierarchical tree over line space
  - in 2D, quadtree
- Indexed by ray coordinates
  - Given \((a,c)\), find linetree leaf and its interpolant

\[
\begin{align*}
  \text{a-c line space} \\
  \text{(a,c)}
\end{align*}
\]

System Overview

Interpolant ray tracer

approximate pixel radiance

render pixel

Failed

render

render

Model

collect radiance samples

Base ray tracer

Interpolant

Interpolant cache

Model
### 3D Rays: 4D Parameterization

- Ray parameterized by \((a,b,c,d)\):
- 6 such pairs of faces

- Linespace is 4D
  - Every ray is point in 4D linespace
  - A box in 4D linespace corresponds to a bundle of 3D rays

### 4D Radiance Interpolants

- Interpolant associated with 4D hypercube in line space
  - Sixteen radiance samples
  - Quadrilinear interpolation

- Samples stored in 4D linetree

- Error-driven subdivision
Results: Museum Scene

1000+ ray-tracing primitives (100k-500k polygons)
195 MHz R10000

gray: interpolation success
yellow: silhouettes; green: shadows; cyan: non-linear radiance

Sparse Sampling Approaches

• 4D:
  – Radiance Interpolants
  – Holodeck

• 2D: Image based
  – Post-rendering Warp
  – Render Cache
  – Edge and Point Rendering