HW 1

- Add whatever you need to ....
  - Get color in the materials
  - Diffuse, etc.

- Only direct lighting

- Only hard shadows

- So why spheres? So that radiosity/radiance conversions etc. work out.
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

Why Software Rendering?

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

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

Rendering as Sampling

- Ray tracers compute radiance at each pixel

- Rendering = Sampling radiance
Coherence

- Within one frame: spatial coherence

- Across many frames: temporal coherence

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
Modified Visual Feedback Loop

Asynchronous interface

Display Process

• Automatically exploit spatial and temporal coherence
• Layered on top of an existing (slow) global illumination renderer
• Provide interactive performance
Aside: Frameless Rendering

• Update pixels as they are computed
  – Don’t wait for full frame to finish

Figure 1. The bottom row shows 7 frames of a 15 Hz frameless rendering sequence with 33% of the pixels updated in each frame. The middle row shows 5 frames of a double-buffering sequence updated at 5 Hz.
Sparse Sampling Approaches

• 4D:
  – Radiance Interpolants
  – Holodeck

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

Post-Rendering 3D Warp

• Render subset of frames
  – E.g, every 6th frame is rendered

• Use standard image warping techniques to compute the other frames
Aside: Pixel Reprojection

- Goal: Want image at new viewpoint
- Reproject points from input images

Assume have depth/disparity per pixel
- If pixel \((x,y)\) sees point \(P\),
  - \(P = C + tD\)
- \(C\) is camera position,
- \(D\) is direction from \(C\) through \((x,y)\)
- \(t\) is distance along \(D\)
Aside: Pixel Reprojection

- Direction $D$

\[ D = C + x \, i + y \, j + d \, k \]

$(x, y)$ = pixel

- $C$ = camera center
- $d$ = distance of image plane from $C$
- $C, d$ are known

\[ P = C_0 + t_0 \, D_0(x,y) \]

$C_0 + t_0 \, D_0 = C_1 + t_1 \, D_1$

$t_1 \, D_1 = (C_0 - C_1) + t_0 \, D_0$

$t_1 \, D_1$ defines the reprojected pixel

$C_0, C_1, D_0, t_0$ are known
Aside: Pixel Reprojection

• $D_1 = C_1 + x_1 i + y_1 j + d_1 k$

• Solve for $x_1$, $y_1$ and $t_1$

Post-Rendering 3D Warp

• Problem:

Pixels do not project to pixel centers
Multiple pixels project to same pixel in new view
Holes and missing data

Reference frame  Warped frame
The camera is moving to the left in this example.
How to fill holes?

- Warp from both past and future reference frames
  - Heuristics for combining pixel results

Problem: Post-Rendering Warp

- Must predict the locations of future frames
  - Longer predictions become rapidly less accurate
Sparse Sampling Approaches

• **4D:**
  – Radiance Interpolants
  – Holodeck

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

Render Cache (Walter et al.)

• **Interactivity is important**
  – Maintain relatively constant framerate
    • e.g., > 5 fps
  – Degrade gracefully as rendering becomes more expensive

• Cache shaded pixels as 3D colored points

• Render new image
  – Project points onto current image plane
  – Filter to reduce artifacts

• Prioritize future rendering
  – Identify problem pixels
  – Sparse sampling for limited render budget
Approach

• Data: Cloud of unordered points with:
  – 3D position, color, age, object id
• Approach: reproject points into image plane
  – Occlusion errors, holes in data

![Initial view](image1.png) ![After reprojection](image2.png)

Image Estimation

• Depth cull heuristic
  – Problem: occluded points may be visible
    ▪ Z-buffering only works within a pixel
  – Clean up using nearby depth information
• Interpolate 3x3

![Raw projection](image3.png) ![depth cull](image4.png) ![interpolation](image5.png)
Sampling

- Choose pixels for rendering: sparse sampling
- Requested pixels sent to renderer(s)
  - Results returned at some later frame

Displayed image  Priority image  Requested pixels

Render Cache Adv and Limitations

- Improved interactivity
- Independent display process
- Drawback: pixel artifacts

Ray tracing  Path tracing
Sparse Sampling Approaches

• 4D:
  – Radiance Interpolants
  – Holodeck

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

Edge-and-Point [Bala’03]

• Goal: Interactive high-quality rendering
  – Expensive shading: e.g., global illumination
  – But, mostly smooth (coherent)
**Edge-and-Point Motivation: Performance**

- Discontinuities are perceptually important
  - Artifacts are disturbing

- Finding discontinuities by sampling is expensive

**Edge-and-Point Rendering**

- Edges: important discontinuities
  - Silhouettes and shadows
- Points: sparse shading samples
Edge-and-Point Image

- Alternative display representation
- Edge-constrained interpolation preserves sharp features
- Fast anti-aliasing

System

- Edge finding
  - 3D edges
  - Rasterization
    - 2D edges
    - EPI
  - Edge-constrained interpolation
- Point cache
  - 3D points
  - Reprojection
    - 2D points
    - Reprojection
  - Edge-constrained interpolation
- Exploits temporal coherence
  - Decouples shading from display
  - Asynchronous
Silhouettes

\[ N_1 \cdot V > 0 \text{ (forward facing)} \]
\[ N_2 \cdot V < 0 \text{ (backward facing)} \]

Shadows: Hard and Soft

- Hard shadows
- Soft shadows
**Umbra and Penumbra Conditions**

- Event plane tangential to light and blocker
  \[ L \cdot N_{\text{blocker}} = L \cdot N_{\text{light}} = 0 \]
  \[ N_{\text{light}} \cdot N_{\text{blocker}} = 1 \text{ (umbral), -1 (penumbral)} \]

**Edge Finding**

- Hierarchical trees: fast edge finding
  - Interval-based
Soft Shadow Edges

Black: silhouettes,
Red: umbral edges, Blue: penumbral edges

Pixel types

- Pixels can have arbitrary edge complexity
- Classify pixels into 3 groups
  - Empty: no edges
  - Simple: can be approximated by 1 edge
  - Complex: everything else

- Typical pixel classification statistics
  - empty (85-95%), simple (4-10%), complex (1-4%)
**Edge-and-Point Image (EPI)**

- **Goal:** compact and fast
  - Store at most one edge and one point per pixel
  - Limited sub-pixel precision

- **Combine edges and points in image space**
  - View-driven, lazy evaluation

---

**Reachability**

- **Reachable samples**
  - Pixel’s 5x5 neighborhood
  - Connected without crossing any edges (or complex pixels)

- **Propagated outward from each pixel**
Results: Quality

- Global illumination
- 3 lights
- 150k polygons

© Kavita Bala, Computer Science, Cornell University

Tea Stand

- Global illumination

© Kavita Bala, Computer Science, Cornell University
Sparse Sampling Approaches

• 4D:
  – Radiance Interpolants
  – Holodeck

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

Corrective Texturing

• Start with a standard hardware rendering of scene
  – Graphics hardware very good at interactive display
  – Start with a radiosity solution

• Compare to underlying renderer
  – Apply corrections where they differ
  – Corrections applied as projective textures
Corrective Texturing

- Sparse rendered samples compared to hardware displayed results
  - Differences splatted into textures
  - More samples generated near points which had large differences
  - Samples which are likely to have changed are deprecated so that can be overwritten by future results
Corrective Texturing

- Corrective textures are dynamically assigned to objects

Comparison

<table>
<thead>
<tr>
<th>Target renderer speed</th>
<th>Sparseness ratio</th>
<th>Typical frame rates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warp</td>
<td>&lt; 1s</td>
<td>4 - 10</td>
</tr>
<tr>
<td>Corrective Tex.</td>
<td>20 - 200s</td>
<td>250 - 1000</td>
</tr>
<tr>
<td>RC</td>
<td>.5 - 10s</td>
<td>8 – 100</td>
</tr>
<tr>
<td>EPI</td>
<td>.5 - 10s</td>
<td>8 - 100</td>
</tr>
</tbody>
</table>
Comparison

<table>
<thead>
<tr>
<th></th>
<th>Hardware accelerated</th>
<th>Independent of scene complexity</th>
<th>Moving objects</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warp</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>?</td>
</tr>
<tr>
<td>Corrective Tex.</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Not really</td>
</tr>
<tr>
<td>RC</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>EPI</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Prediction

• **Hardware**
  – Speed
  – Programmability

• **Software**
  – High-complexity data sets
  – Complex GI

• **Hybrid techniques**
Dealing with High Complexity

• Many Lights

• Display systems
  – Point-based approaches

• Visibility pre-processing systems

• Image-Based Rendering

Acceleration Data Structures

CS 665
Making RT faster

Ray Tracing Acceleration Techniques

- Faster Intersections
- Fewer Ray-Object Intersections
- Fewer Rays
- Regular Rays
- Generalized Rays

• For each pixel, $O(N)$
• For each light, $k$ shadow rays
• For GI and antialiasing: many rays per pixel

Fewer Rays: Regular Rays

- Regular rays
  - Adaptive tree-depth control
  - Adaptive antialiasing

$K_t \cdot K_s \cdot K_r$, Shading
Generalized Rays

- Generalized rays represent a set of rays
  - Cone
  - Beam
- Pros
  - Good for anti-aliasing
  - Decreases number of rays
- Cons
  - More complex intersection tests
  - Reflections and refractions get hairy

Making RT faster

- Faster Intersections
- Fewer Ray-Object Intersections
- Regular Rays
- Generalized Rays

Faster Ray-Object Intersections

© Kavita Bala, Computer Science, Cornell University
Faster Ray-Object Intersections

• Object bounding volumes

• Avoid intersection tests for expensive objects: e.g., polygon sets, spline surfaces
  – Ray/sphere or ray/cuboid test is fast

Intersection: sphere

Assume sphere \( x^2 + y^2 + z^2 = 1 \)
Point of intersection \( p = O + t_{\text{intersection}} D \)
p lies on sphere
Solve \( A t_{\text{intersection}}^2 + B t_{\text{intersection}} + C = 0 \)
\( A = 1, \quad B = 2 \ (O.D), \quad C = (O.O - 1) \)

\[ t_{\text{intersection}} = \frac{-B \pm S}{2A}, \]
\[ S = \sqrt{B^2 - 4AC} \]
Intersection: cube

tNear = -inf, tFar = +inf
For each pair of planes for the x,y,z axes {
  Solve for  O[i] + D[i] t1 = Min[i]
  Solve for  O[i] + D[i] t2 = Max[i]
  What if t1 > t2? swap
    tNear = max (t1, tNear)
    tFar = min (t2, tFar)
}
if (tNear > tFar) missed box
else hit box

Tight Fit to Bounding Volume

© Kavita Bala, Computer Science, Cornell University