Rendering Complex Scenes with Memory-Coherent Ray Tracing

Matt Pharr, Craig Kolb, Reid Gershbein, Pat Hanrahan

Matt Patchin
The Goal

• Rendering Systems are challenged by three types of complexity:
  – Geometric
  – Surface
  – Illumination

• Handling all three at one had not been previously possible.
The Goal

• Generally speaking, some sacrifices were made with respect to these complexities in order to achieve reasonable run times

• Sought to rectify this by improving the coherence of the scene.
Overview

• Developed algorithms based two main ideas:
  – **Caching** — cached a subset of large geometric objects and textures in memory for fast access.
  – **Reordering** — In order to ensure coherent access, scene data and ray intersection calculations were reordered and placed into memory.
• These algorithms made it possible to efficiently compute incredibly complex images
Overview
Overview

• Disk Storage is used to manage texture, geometry, queued rays, and image samples.
• First the camera generates the eye rays to form the image and these rays are then partitioned into groups.
• The Scheduler selects groups of rays to trace. Intersection tests are performed with chosen rays, which causes geometry to be added to the cache.
• As intersections are found, shading occurs and texture maps are handled by the texture cache.
• Any new rays that are generated by this process are returned to the scheduler to be added in to the queue.
Caching Scene Data

• Cache texture and geometric information to manage scene complexity.
• Limited amount is stored in memory and lazy loading limits what is added to the cache.
• Both caches use a least recently used replacement policy.
Caching Scene Data

• Geometric Sources
  – Only use triangles.
  – Advantages
    • Ray intersection can be optimized.
    • Memory management for geometry cache is easier.
    • Other parts of the renderer can be optimized when only one type of primitive is supported.
  – Disadvantage
    • Some primitives, such as spheres, require more space to store after tessellation.
Caching Scene Data

• Geometry Cache Properties
  – Organized around voxel grids called Geometry grids.
  – All geometry in a voxel occupies a contiguous block of memory independent of geometry in other voxels. Particularly, triangles that span multiple voxels are stored independently in each of them.
  – Reasoning for this was that the spatial locality in the 3D space of the scene is tied to the spatial locality in memory. This way coherent access in 3D space generates coherent access in memory.
Caching Scene Data

• Geometry Cache Properties
  – Memory management in the geometry cache is more complicated than in the texture cache.
  – Fixed this by introducing their own allocation routines.
Reordering Computation

• The geometry and texture caching methods provide a framework for rendering larger scenes. To make overall system performance acceptable, cache misses must be minimized.

• Accomplished through dynamic reordering of ray-object intersections.

• Instead of evaluating the rays in a fixed order, all rays are placed in queues and the system chooses rays. It simultaneously tries to minimize cache misses and advance the computation.

• To this end, each ray does not depend on the results or state of other rays. This independence implies that each ray is stored with all information needed to compute its contribution to the image and the space that this requires must be minimized given the potential for massive amounts of rays.

• Both of these can be accomplished by decomposing the outgoing radiances into a simple sum of weighted incoming radiances.
Reordering Computation

• Computation Decomposition

\[ L_o(x, \omega_r) = L_e(x, \omega_r) + \frac{1}{N} \sum_{i} f_r(x, \omega_i, \omega_r) L_i(x, \omega_i) \cos \theta_i, \]
Reordering Computation

• Ray Grouping
  – One early approach tried to organize rays with nearby origins into clusters. The rays would then be sorted by direction to increase coherence and then trace them through the scene.
  – Advantage
    • Good for scenes where a majority of rays are spawned from a few locations.
  – Disadvantage
    • Fails to exploit the coherence between rays in a beam and rays that pass through the same region of space but whose origins are not close together.
Reordering Computation

• Ray Grouping
  – The approach they decide on was designed to account for spatial coherence between all rays.
  – Scene was divided into another set of voxels called the scheduling grid. Associated with each voxel is a queue of the rays currently inside it and information regarding its geometry.
  – If an intersection is found, shade and calculate spawned rays. Otherwise the ray is advanced to the next non-empty voxel and placed on the voxels ray queue.
Results

They tested their algorithms with three complex scenes.
Results

• Caching
  – Rendered the Cathedral Scene twice and Office Scene once.

<table>
<thead>
<tr>
<th></th>
<th>Cathedral</th>
<th>Cathedral Lazy</th>
<th>Indoor Lazy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running Time</td>
<td>163.4 min</td>
<td>156.9 min</td>
<td>35.3 min</td>
</tr>
<tr>
<td>Memory Use</td>
<td>431MB</td>
<td>337MB</td>
<td>316MB</td>
</tr>
<tr>
<td>% Texture Accessed</td>
<td>100%</td>
<td>49.6%</td>
<td>100%</td>
</tr>
<tr>
<td>% Geometry Accessed</td>
<td>100%</td>
<td>81.6%</td>
<td>17.8%</td>
</tr>
</tbody>
</table>

– Texture cache also performed extremely well.
Results

• Reordering
  – Lake scene was used to test performance of reordering.