CS 419: Production Rendering

Geometric Data Structures: Regular Grids

Eric Shaffer
Objectives

- Be able to use geometric data structures to accelerate ray-tracing
  - Know how to compute/intersect a bounding box
  - Be able to implement a regular grid to accelerate ray tracing
Bounding Boxes

- Complex geometric models require expensive tests for ray intersections
- Boxes are fast to test for ray intersection
- Can implement a “quick rejection” test on the bounding box of a complex object
  - Usually, lots of rays will miss an object
Axis-Aligned Bounding Box

- Box is defined by
  - min point $p_0 = (x_0, y_0, z_0)$
  - max point $p_1 = (x_1, y_1, z_1)$

- Box is $[x_0, x_1] \cap [y_0, y_1] \cap [z_0, z_1]$

- How can we efficiently compute the box?
  - Imagine you are given a triangle mesh... what is the bounding box for all those triangles?
Box-Ray Intersection Test

- Box is defined by slabs along each axis
- We will look at 2D case and generalize to 3D
- Ray misses the box when the slab intersection intervals do not overlap
- How would you test this?
Box-Ray Intersection Test

- Check if largest entering t value is less than smallest exiting t value
Box-Ray Intersection Test

- But ray can also miss if smallest exiting $t$ value is negative
Geometric Data Structures

- We can do more than just use bounding boxes
- What about a data structure that partitions space?
  - And we only test objects in the sections of space the ray traverses
- Lots of choices
  - Bounding Volume Hierarchy (BVH)
  - Octree
  - BSP-Tree
  - KD-tree
  - Regular grid
A regular grid is an axis-aligned box
Subdivided into smaller axis-aligned boxes called cells
Each cell has the same shape and size
This is why it’s called a regular or uniform grid
Regular Grids

- Each cell stores a list of objects that
  - Are contained in the cell
  - Are partially contained in the cell
  - Might be in the cell….

- Why might? What insertion procedure would result in that condition?
Tracing Rays through Regular Grids

- You can march a ray through the grid
  - Only test for intersections against objects in the cells you march through
- You can also terminate the march on the closest hit
  - Is closest = first?
Constructing a Regular Grid

- Procedure is simple:
  - Compute bounding box of all objects
  - Divide up the bounding box into cells
  - Insert the objects into the grid

- Implementation detail:
  - Underlying data structure should be a hash table
  - Hash functions should map cell (i,j,k) to array index
  - Only allocate space for a cell if it is occupied
A Hash Function for 3D Points

- Convert a point \( p=(x,y,z) \) into \((i,j,k)\) indices
  - In other words, map \( p \) to grid cell address
- Convert \((i,j,k)\) to binary
- Interleave the bits of the 3 binary numbers
- This generates a Morton Code
  - Also known as a Z-order curve
- Example: \((5,9,1)\)
  - \((0101,1001,0001)\)
  - 010001000111 = index 1095 in a linear array
- Where does the Z-order curve place neighbor grid cells in the array? Why is this important?
How Many Cells?

- Want to avoid having large number of objects in a single cell
- Isotropic cells are probably best for uniformly distributed objects
  - We’ll try to make cells as cube-like as we can
- $n_x$, $n_y$, $n_z$ are the number of cells along each axis
- $w_x$, $w_y$, $w_z$ is the length of the grid along each axis
- Let $n$ be the number of objects we have to store
- $m$ allows you to pick how many cells per object
  - $m=2$ yields 8ish cells per object, good number
  - why use +1?

\[
\begin{align*}
  s &= \sqrt[3]{\frac{w_x w_y w_z}{n}} \\
  n_x &= \left\lceil \frac{m w_x}{s} \right\rceil + 1 \\
  n_y &= \left\lceil \frac{m w_y}{s} \right\rceil + 1 \\
  n_z &= \left\lceil \frac{m w_z}{s} \right\rceil + 1
\end{align*}
\]
Inserting Objects into the Grid

- Compute the bounding box for the object
- Intersect the box with the grid
  - Insert object into all cells overlapped by box

(a)         (b)
Inserting Objects into the Grid

- Compute the bounding box for the object
- Intersect the box with the grid
  - Insert object into all cells overlapped by box

(a)         (b)
Example: Computing x-axis index for a point \( p \)

\[
f(p_x) = \frac{(p_x - p_{0x})}{(p_{1x} - p_{0x})}
\]

\[
i_x = \left\lfloor n_x f(p_x) \right\rfloor
\]

Note that indices must be clamped in range \([0, n_x - 1]\)
Traversal

- Find start point of a ray through the grid
  - If ray origin is inside grid
    - If so, find the starting cell
  - If origin is outside grid
    - Intersect ray with grid bounding box
      - If ray hits the grid bounding box
        - Find starting grid cell
      - Else, return 0 intersections
Traversals: Marching Along the Ray

- Intersection of rays with cell faces can be unequally spaced
- They are equally spaced in the x, y, and z directions
- We can compute parametric increments across the cell
  - Figure out, based on smallest t, which face the ray next hits
Traversal: Marching Along the Ray

(a) \( t_{x_{next}} \) \( t_{y_{next}} \)

(b) \( t_{x_{next}} \) \( t_{y_{next}} \)
Theoretical Performance

- Grid will have $O(\sqrt[3]{n})$ cells in each cardinal direction.
- Rays tested on $O(\sqrt[3]{n})$ objects.
- In practice, $O(\log n)$ tests per ray.
Empirical Performance

- From book...10 years ago
- 400x400 image using 1 ray per pixel, no shadows
  - Uniform grid m=2
  - 450 MHz Mac G3(!?) with 512 MB RAM

<table>
<thead>
<tr>
<th>Number of Spheres</th>
<th>Render Times in Seconds</th>
<th>Grid Speed-Up Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>With Grid</td>
<td>Exhaustive</td>
</tr>
<tr>
<td>10</td>
<td>1.5</td>
<td>2.5</td>
</tr>
<tr>
<td>100</td>
<td>2.0</td>
<td>16</td>
</tr>
<tr>
<td>1000</td>
<td>2.7</td>
<td>164</td>
</tr>
<tr>
<td>10,000</td>
<td>3.8</td>
<td>2041</td>
</tr>
<tr>
<td>100,000</td>
<td>4.7</td>
<td>22169 (6 hours)</td>
</tr>
<tr>
<td>1,000,000</td>
<td>5.2</td>
<td>forget it!</td>
</tr>
</tbody>
</table>
Empirical Performance
A hit on an object may not be in the cell

Need to check the intersection point falls in cell
  - If not, keep marching

Will we end up testing same object multiple times?
  - Use mailboxing in which
    - Each ray gets an integer id
    - Store most recent checked ray id and test result with object...like caching
Uniform Grids and Scene Complexity

- Work best for uniformly distributed objects
  - Objects all have similar complexity
- Poor performance for non-uniform complexity in scene
  - “Teapot in a stadium” problem – Eric Haines
  - Can nest grids or use a tree-based structure