Recall: Whitted-style ray tracing

how do we make this faster?

foreach pixel sample
  foreach object
    if ray hit object
      if z < min_z
        update pixel color
Idea: build a “database” of objects
Idea: build a “database” of objects
Quiz time: what’s a good “database”? 
Quiz time: what’s a good “database”? 

- desired properties
- fast construction time
- fast query time
- fast “update” (for animation)
Two ideas for building a database

- Subdivide the space (today)
  - Uniform grid, octrees, kd-trees, BSP trees

- Subdivide the objects (next time)
  - Bounding volume hierarchy
Uniform grid

- **construction:**
  build a 3D grid, for each voxel, store a linked list of overlapping object IDs

in reality we build 3D grids, the 2D grid here is for illustration
Uniform grid

- **construction:**
  build a 3D grid, for each voxel, store a linked list of overlapping object IDs

- **query:**
  traverse the grid, do intersection tests with each objects inside the grid

in reality we build 3D grids, the 2D grid here is for illustration
Uniform grid

- **construction:**
  build a 3D grid, for each voxel, store a linked list of overlapping object IDs

- **query:**
  traverse the grid, do intersection tests with each objects inside the grid

**quiz 1:** why is this faster than brute-force?

in reality we build 3D grids, the 2D grid here is for illustration
• **construction:**
  build a 3D grid, for each voxel, store a linked list of overlapping object IDs

• **query:**
  traverse the grid, do intersection tests with each objects inside the grid

**quiz 1:** why is this faster than brute-force?
**quiz 2:** what is the optimal grid size?

in reality we build 3D grids, the 2D grid here is for illustration
Uniform grid

- **construction:**
  build a 3D grid, for each voxel, store a linked list of overlapping object IDs

- **query:**
  traverse the grid, do intersection tests with each objects inside the grid

**quiz 1:** why is this faster than brute-force?  
**quiz 2:** what is the optimal grid size?  
**quiz 3:** how do we avoid testing with the same object multiple times?
Uniform grid

- **construction:**
  build a 3D grid, for each voxel, store a linked list of overlapping object IDs

- **query:**
  traverse the grid, do intersection tests with each objects inside the grid

**quiz 1:** why is this faster than brute-force?
**quiz 2:** what is the optimal grid size?
**quiz 3:** how do we avoid testing with the same object multiple times?

in reality we build 3D grids, the 2D grid here is for illustration
Mailboxing: caching intersection results

- store the intersect result of the latest test, so we don’t need to intersect with the big triangle multiple times
• **construction:**
  build a 3D grid, for each voxel, store a linked list of overlapping object IDs

• **query:**
  traverse the grid, do intersection tests with each objects inside the grid

**quiz 4:** what would be a “failure case” for uniform grid?

---

in reality we build 3D grids, the 2D grid here is for illustration
Uniform grid failure case: teapot in a stadium

a super detailed teapot with 1 billion triangles
Uniform grid: pros and cons

• desired properties
  • fast construction time ✓
  • fast query time ❌
  • fast “update” (for animation) ✓
Two-level grid

- each cell itself can be another grid

- **construction:**
  build a normal grid first, refine the grid if we detect too many primitives in one

- **query:**
  same as uniform grid!
Two-level grid

- each cell itself can be another grid

- **construction:**
  build a normal grid first, refine the grid if we detect too many primitives in one

- **query:**
  same as uniform grid!

---

Two-Level Grids for Ray Tracing on GPUs

Javor Kalojanov\textsuperscript{1} and Markus Billeter\textsuperscript{2} and Philipp Slusallek\textsuperscript{1,3}

\textsuperscript{1}Saarland University, \textsuperscript{2}DFKI Saarbrücken
\textsuperscript{3}Chalmers University of Technology

2010
Multi-level grids and octrees

- each cell itself can be another grid, and each cell inside that grid can be another grid...

- octree: a special case where the grid size is always 2x2x2
Multi-level grids: pros and cons

• Quiz time: what happens to the construction/query/update time as we increase the number of levels?

• desired properties
  • fast construction time
  • fast query time
  • fast “update” (for animation)
Do we have to split the space in the center? v.s.
Do we have to split all axes?

v.s.
Kd-tree: a grid with irregularly sized cells
Kd-tree construction
Kd-tree construction

- pick an axis first
Kd-tree construction

- pick an axis first

- divide the space into two

**quiz:** how many different ways we have for dividing the space?
Kd-tree construction

- pick an axis first
- divide the space into two
- recurse
How to choose split axis & location?

• split axis: quiz time!

• split location: quiz time!
How to choose split axis & location?

• split axis:
  round-robin? split the largest extent?

• split location:
  middle of the extent? median of geometry?

• none of these choices work really well in practice!
How to choose split axis & location?

• current state of the art: enumerate all choices, choose a split that will minimize an approximate cost of ray tracing

• the approximate cost: surface area heuristics (more about it next time)
Kd-tree construction can be done in $O(N \log N)$

(N = number of primitives)

On building fast kd-Trees for Ray Tracing, and on doing that in $O(N \log N)$

Ingo Wald†  Vlastimil Havran°

†SCI Institute, University of Utah  °Czech Technical University in Prague

2006
Kd-tree traversal

- $t^* > t_{\text{max}}$: test left node
- $t_{\text{min}} \leq t^* \leq t_{\text{max}}$: test both nodes
- $t_{\text{min}} > t^*$: test right node
Kd-tree vs multi-level grid

• Quiz: what are the pros and cons?

• desired properties
  • fast construction time
  • fast query time
  • fast “update” (for animation)
Kd-tree vs multi-level grid

- Kd-tree: faster query, okay construction, slower update time

- desired properties
  - fast construction time
  - fast query time
  - fast "update" (for animation)
Do we have to split with a vertical/horizontal line?

v.s.
BSP tree: split with arbitrary planes

Binary Space Partition
BSP trees vs kd-trees

- BSP trees:
  - notoriously difficult to implement
  - more efficient space partition, but more expensive traversal
  - more expensive construction, very difficult to update
Hybrid BSP / Kd tree

use surface area heuristics (approximate ray tracing cost) to decide which is better at each split

approximate cost: 400

approximate cost: 500

Ray Tracing with the BSP Tree

Thiago Ize
University of Utah

Ingo Wald
Intel Corp

Steven G. Parker
NVIDIA
University of Utah

2008
Recall: two ideas for building a database

- subdivide the space (today)
  - uniform grid, octrees, kd-trees, BSP trees

- subdivide the objects (next time)
  - bounding volume hierarchy
Recall: two ideas for building a database

- subdivide the space (today)
  - uniform grid, octrees, kd-trees, BSP trees

- subdivide the objects (next time)
  - bounding volume hierarchy

In practice, most modern ray tracers have moved to using bounding volume hierarchy.
Next: bounding volume hierarchy

in practice, most modern ray tracers have moved to using bounding volume hierarchy

subdivide the objects (next time)

bounding volume hierarchy