Spatial Data Structures

5 April 2007
CMPT370
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Review last time

- Ray tracing
- Object-space vs. image-space rendering
- Backward ray tracing
  - Shadow rays
  - Reflection and refraction rays
  - Ray-surface intersection
    - Parametric surface
    - Implicit surface
    - Sphere / Quadric
    - Polygon

Ray-object intersections

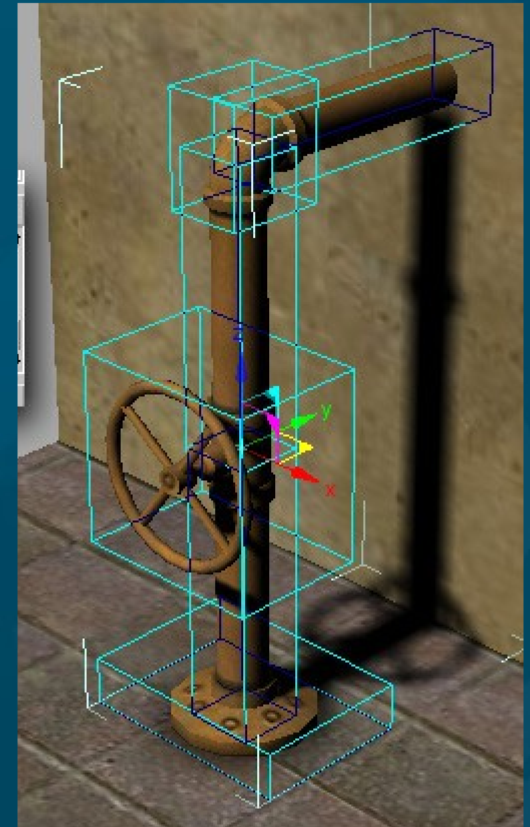
- For every ray:
  - Find all intersections with all objects
    - Choose the closest intersection
  - Cast shadow rays to every light
  - Cast reflection and refraction rays and recurse
- Significant computation work in ray-object intersections!
- Speed up ray-object intersection tests:
  - Cull away objects that won't be intersected

Spatial data structures

- Storing the geometry in a smarter way
- Applications:
  - Rendering
  - Collision detection
    - Robotics
    - Virtual world / gaming
    - Chemical / drug simulation
- Object-centred: bounding volumes
- Space-subdivision: grid, octree, BSP
- Speed-ups of 100x or more!

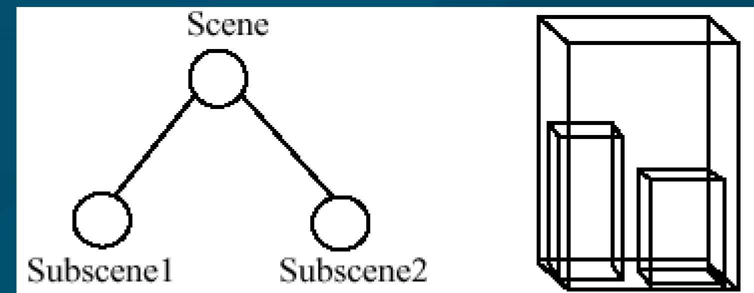
Object bounding volumes

- Object-centred data structure
- Wrap complex objects in simple ones
  - Level of detail
- If ray does not intersect bounding volume, it won't intersect the object
- Common types:
  - Axis-aligned boxes
  - Oriented boxes
  - Spheres
  - Convex hulls

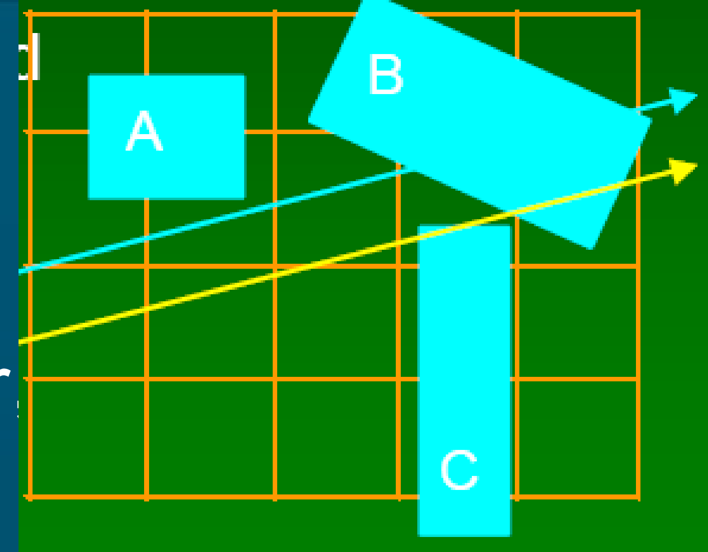


Hierarchical bounding volumes

- Straightforward bounding volumes still store objects in a flat list: O(n) intersection tests
- Use a tree structure: boxes within boxes
- Recursively test for intersections:
  - If ray misses large box, don't need to descend tree
  - If ray hits large box, recurse into smaller boxes
- Challenges:
  - Constructing full balanced tree

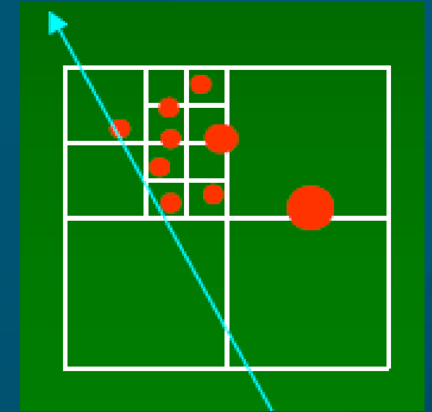


Spatial subdivision: grids



- Instead of grouping objects together, partition space (the view frustum)
- Grids: 3D array of cells that tile the space: voxels
- Each voxel keeps a list of all surfaces that intersect it
- For each voxel intersected by the ray:
  - Test intersection with each surface in the voxel
- Only good if objects are uniformly spread in space
  - Voxels too big => too many surfaces per cell
  - Voxels too small => too many empty cells to walk
- Try non-uniform cell spacing

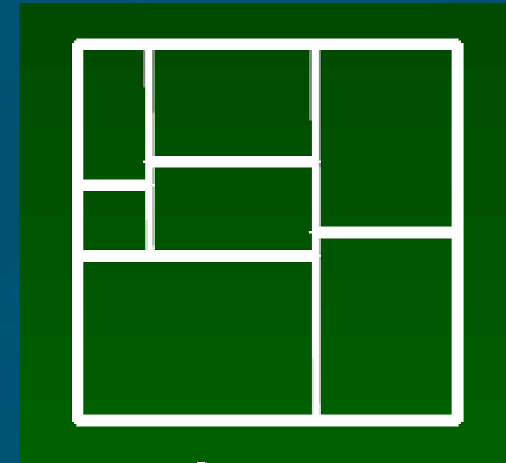
Octrees



- In 1D: binary tree
- In 2D: quadtree
- Each cell (node of tree) is a cube
- Recursively split into 8 equal sub-cubes
  - Adaptive subdivision: stop dividing based on number of surfaces in the cell
- Ray intersection: traverses tree
  - Tradeoff: tough to step to next cell along ray

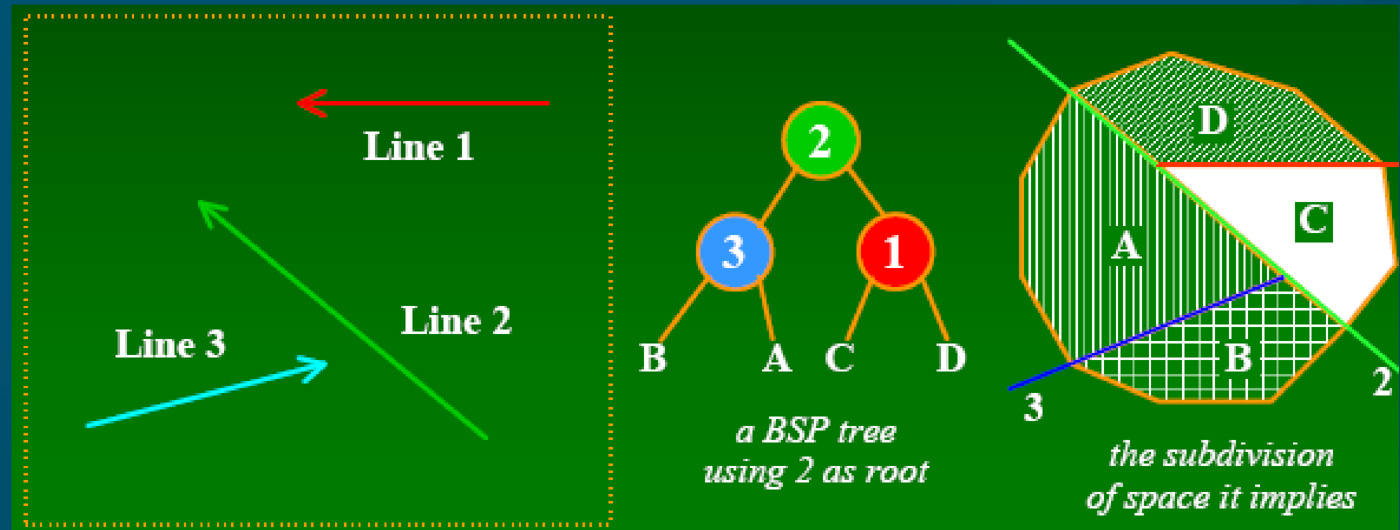
k-d trees and BSP trees

- Relaxing the rules on octrees
- k-d (k-dimensional) trees:
  - Split each cell one dimension at a time at arbitrary point within cell
- BSP (binary space partitioning) trees:
  - Split with plane of any orientation
    - In k-dims, split with hyperplane of dimension k-1
  - Used for hidden-surface removal
    - Painter's algorithm: planes oriented relative to camera

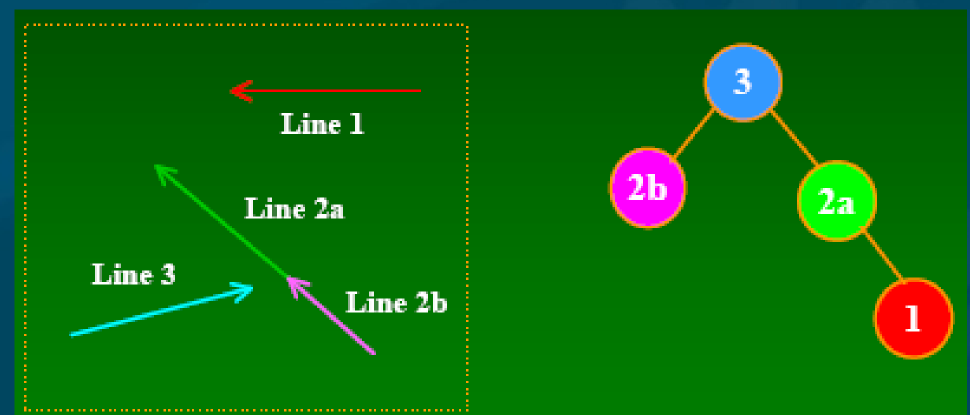


Building balanced trees

- Use the objects to guide choice of splitting plane
- Example with simple line segments



- Using Line3 as root requires splitting Line2
- Splitting gives more surfaces but often a more balanced tree



TODO

- Lab5 due next Thu 12Apr
  - Virtual world
    - Creative, interesting scene
    - Lights and materials
    - Texture map
    - Bezier evaluator or NURBS
    - Pick objects
  - Final deadline for late labs: Thu 19Apr