CS 4732:
Computer Animation

Scene Acceleration Structures

Robert W. Lindeman
Associate Professor
Interactive Media & Game Development
Department of Computer Science
Worcester Polytechnic Institute
gogo@wpi.edu

I Want More, More, More!

- Users want ever-increasing
  - Realism
    - Graphical
    - Behavioral
    - Lighting
  - Interactivity with environments
  - Numbers of characters
- Hardware is always getting better
  - But *never* fast enough!!!

I Want More, More, More! (cont.)

- Hardware will always lag behind needs
- Stated otherwise:
  - Needs always expand to fill a performance vacuum!
- Need to better manage things
  - Visibility calculation
  - Texture (and other) mapping
  - Can fake shadows
  - Can pre-compute some reflections
  - Lots of other tricks!!!!

Bottom Line

- Graphics cards can render a lot, very fast
  - But never as much, or as fast as we'd like!
- Intelligent scene management allows us to squeeze more out of our limited resources
  - Scene graphs
  - Scene partitioning
  - Visibility calculations
  - Level of detail control

Scene Graphs

- A specification of object and attribute relationships
  - Spatial
  - Hierarchical
  - Material properties
- Transformations
- Geometry
- Easy to attach objects together
  - Riding a vehicle

Scene Graphs (cont.)

- Can use instances to save resources
  - Geometry handles instead of geometry
  - Texture handles
- To take advantage of GPUs, reducing the amount of shader (cg) and texture switching is preferred

Geometry Sorting and Culling

- Keys to scene management
  - Render only what can be seen
  - Render at a satisfactory, perceivable fidelity
  - Pre-process what you can
  - Use GPU as efficiently as you can
- First-level
  - View-frustum culling
  - Back-face culling
  - Bounding volumes
- One or more ***acceleration structures*** can be used

Acceleration Structures

- Many structures exist
  - Appropriateness depends on the scene, and the game (e.g., dynamic objects)
- Geometry partitioning
  - Bounding boxes/spheres/capsules
- Space partitioning
  - Uniform Grid
  - Quad/Oct Tree
  - Binary-Space Partitioning (BSP) trees
  - k-d trees
- Speed up of 10x, 100x, or more!

Acceleration Structures (cont.)

- Hierarchical bounding structures
  - Test if parent is visible
    - If not, then none of its children are
    - If so, then recursively check the children
- Could use information about your application to optimize approach
  - Many interior levels have cells and portals
  - No need to solve the general problem, just the specific one

Acceleration Structures - Geometry Partitioning

- Bounding boxes/spheres/capsules
- Axis-Aligned Bounding Boxes (AABB)
- Oriented Bounding Boxes (OBB)
- Discrete Oriented Polytope (DOP)
  - Polytope: 2D = polygon, 3D = polyhedron
  - *k*-DOP: *k* planes in a DOP
  - Common: 6-DOP (AABB), 10-DOP, 18-DOP, 24-DOP
- Bounding-Volume Hierarchies (BVHs)

Acceleration Structures - Space Partitioning

- Uniform Grids
  - Split space up into equal sized (or an equal number of) cells
- Quad (Oct) Trees
  - Recursively split space into 4 (8) equal-sized regions
- Binary-Space Partitioning (BSP) trees
  - Recursively divide space along a single, arbitrary plane
- *k*-dimensional trees (k-d trees)
  - Recursively split along axes

Bounding Volumes

- Objects could have fairly complex shapes
- Wrap complex objects in simple ones
  - Boxes (axis-aligned, or oriented)
  - Spheres
  - Capsules
  - Finite intersections or unions of above
- Do bounding volumes collide?
  - No = do nothing
  - Yes = Calculate intersection points, forces, etc.

Selection of Bounding Volumes

- Effectiveness depends on
  - Probability that bounding volume is contacted, but not enclosed object (tight fit is better)
  - Expense to calculate intersections with bounding volumes and enclosed objects

Hierarchical Bounding Volumes

- Simple bounding volume testing can require O(n) intersection tests
- Use a tree structure instead
  - Larger bounding volumes contain smaller ones
  - Sometimes naturally available (*e.g.*, human figure)
  - Sometimes difficult to compute
- Often reduces complexity to O(log(n))

Object Collision Algorithm

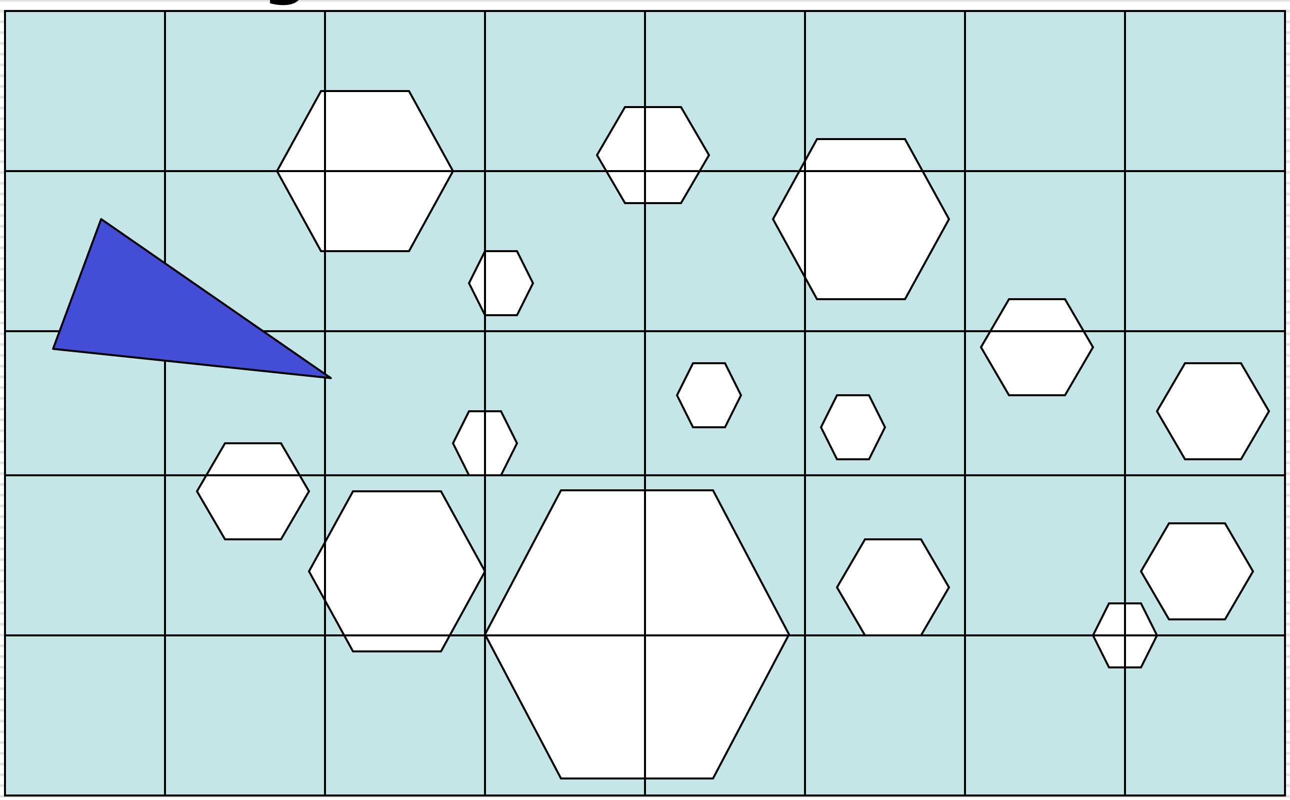
- Recursively descend tree
- If no intersection with bounding volume, no collision
- If intersection with bounding volume, recurse with enclosed volumes and objects
- Maintain near and far bounds to prune further
- Overall effectiveness depends on model and constructed hierarchy

Spatial Subdivision

- Bounding volumes enclose objects recursively
- Why not divide the space instead?
- For each segment of space, keep list of intersecting surfaces or objects
- Basic technique
  - Regular grids
  - Octrees (axis-aligned, non-uniform partition)
  - BSP trees (recursive Binary Space Partitions)

Regular Grids

- 3D array of voxels, list of surfaces intersecting cell

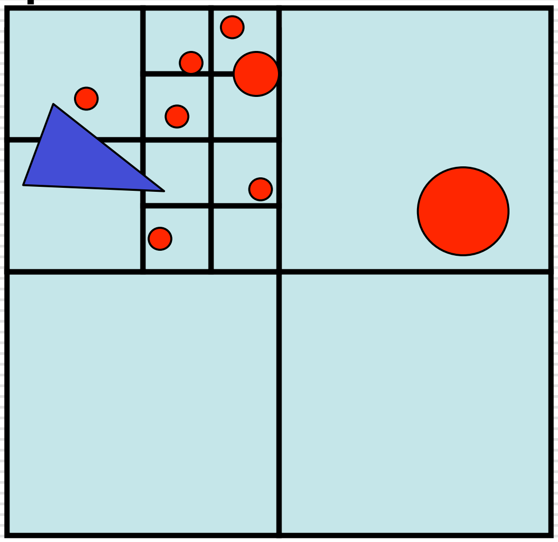


Assessment of Grids

- Poor choice when world is non-homogeneous
- Size of grid?
  - Too small: too many surfaces per cell
  - Too large: too many empty cells to traverse
- Non-uniform spatial subdivision more flexible
  - Can adjust to objects that are present

Quadtrees

- Generalization of binary trees in 2D
  - Node (cell) is a square
  - Recursively split into 4 equal sub-squares
  - Stop subdivision based on number of objects
- More difficult to step to next cell



Octrees

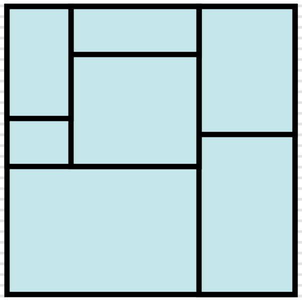
- Generalization of quadtree in 3D
- Each cell may be split into 8 equal sub-cells
- Internal nodes store pointers to children
- Leaf nodes store list of surfaces
- Adapts well to non-homogeneous scenes

Assessment for Collision Detection

- Grids
  - Easy to implement
  - Require a lot of memory
  - Poor results for non-homogeneous scenes
- Octrees
  - Better on most scenes (more adaptive)
- Alternative: nested grids
- Spatial subdivision expensive for animations
- Hierarchical bounding volumes
  - Natural for hierarchical objects
  - Better for dynamic scenes

Other Spatial-Subdivision Techniques

- Relax rules for quadtrees and octrees
- K-Dimensional tree (K-D Tree)
  - Split at arbitrary interior point
  - Split one dimension at a time (Horiz./Vert.)



- Binary space partitioning tree (BSP Tree)
  - In two dimensions, split with any line
  - In K dimensions, split with K-1-dimensional hyperplane
  - Particularly useful for painter's algorithm
  - Can also be used for ray tracing

BSP Trees

- Inherent spatial ordering given viewpoint
  - Left subtree: in front, right subtree: behind
- Problem: finding good space partitions
  - Proper ordering for balanced tree
- http://symbolcraft.com/graphics/bsp/

Cell-Portal Visibility

- Keep track of which cell the object is in
- Somehow enumerate all reachable regions
- Cell-based
  - Preprocess to identify the potentially visible set for each cell

Putting it all Together

- The "best" solution will be a combination
  - Static things
    - Oct-tree for terrain
    - Cells and portals for interior structures
  - Dynamic things
    - Quick reject using bounding spheres
    - BVHs for objects
- Balance between pre-computation and run-time computation

Reduce, Reuse, Recycle!

- These approaches can be used all over the place in graphics and animation
  - Ray tracing (e.g., intersections)
  - Collision detection
  - Visibility calculation
  - Behavioral animation

References

- http://www.cs.wisc.edu/graphics/Courses/679-f2003/