# *Data Structures & Algorithms for Geometry*

- Agenda:
  - Quiz #2
  - Space partitioning
    - Uniform grids
    - Hierarchical grids
    - Quadtrees / Octrees
    - k-d trees
    - Tree traversal
  - Assignment #3

# *Space Partitioning Overview*

- BVs and BVHs reduce *comparison* costs.
  - Collision comparisons
  - Visibility comparisons
  - etc.

# *Space Partitioning Overview*

- BVs and BVHs reduce *comparison* costs.
  - Collision comparisons
  - Visibility comparisons
  - etc.
- Space partitions reduce *search* costs
  - Find all objects near another object
  - Find all objects inside the view frustum
  - Find all objects along a ray
  - etc.

# *Uniform Grid*

- Divide the world into fixed size, uniform regions.
- Store each object in the bucket for each region that it overlaps.
- What is the most important parameter of the grid?

# *Cell Size*

- Too big → too many objects in each cell
- Too small → each object overlaps many cells
- Too big *and* too small → if the size of objects varies a lot, large objects may overlap many cells while lots of small objects are in a single cell

# Grid Representation

- Obvious implementation: m-by-m-by-m array of linked lists.
  - Divide each coordinate by the cell size to find the array element, search the array for the desired object.
- Problems?

# *Grid Representation*

- Obvious implementation: m-by-m-by-m array of linked lists.
  - Divide each coordinate by the cell size to find the array element, search the array for the desired object.
- Problems?
  - Worst case search time is O(n).
  - If *m* is large, can use a **lot** of memory.
    - Even worse, many of the lists might be empty!

# *Grid Hash Table*

- Use smaller array to store buckets.
  - Divide coordinates by grid size (like before)
  - Use a hash function on the grid coordinates
  - Find the object at the bucket specified by the hash value
- Problems?

# Grid Hash Table

- Use smaller array to store buckets.
  - Divide coordinates by grid size (like before)
  - Use a hash function on the grid coordinates
  - Find the object at the bucket specified by the hash value
- Problems?
  - Good hash functions for grid data are hard to make
  - Usual collision handling problems

# Static Data Optimization

- If all data is static store objects in a large array
  - Group the objects in the array so that objects in the same sell are together in the array
  - In each cell store the index of the first object and the number of objects.
    - This works with both the previous representations
- Saves storage space, but does *not* help search time.

# *Implicit Grids*

- Store an array of n lists for each axis.
  - Objects are placed the lists for the regions of each axis that they overlap.
  - Uses n+m+p buckets instead $n \times m \times p$

# Grid Element Selection

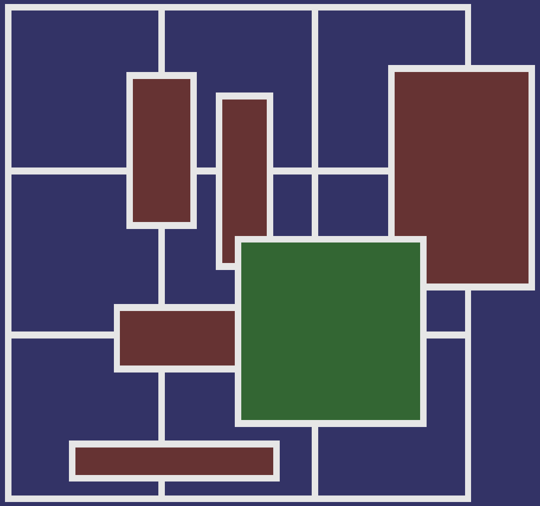
- No matter how big the grids are, an object can overlap 4 elements.
  - How do we decide where to store the object?

# Grid Element Selection

- No matter how big the grids are, an object can overlap 4 elements.
  - How do we decide where to store the object?
- Make the cells just larger than the largest object
  - Each object can only overlap 4 cells (8 cells in 3D)
- Place each object in the cell that its minimum corner lies in.

# *Uniform Grid Object-Object Intersection*

- Since each object can overlap 4 cells:
  - Test the four cells that the object might overlap
  - Test the five cells that might contain objects that overlap the cells the original object might overlap.
- Some tests can be avoided if the test object doesn't overlap all 4 cells.
  - If *all* objects are being tested, only the 4 cells each object overlaps need testing.



# *Hierarchical Grids*

- Fixed number of levels in the hierarchy
- Cells in level n+1 are usually half the dimensions of cells in level n.
  - Sound familiar?

# *Hierarchical Grids*

- Fixed number of levels in the hierarchy
- Cells in level n+1 are usually half the dimensions of cells in level n.
  - Sound familiar?  Like mipmaps, perhaps?
- Store objects at the level in the tree where the cell size is just larger than the object size
  - This gives the 4-cell overlap property.

# *Hierarchical Grid Object-Object Intersection*

- Object-object intersection tests requires checking all cells (up and down) in the hierarchy that the object might intersect.
  - If all objects are being tested, only the current level and the larger-cell levels need be tested.

# Quadtrees

- 2D tree hierarchy
  - Start with the axis-aligned bounding square.
    - Must be a square, not a general AABB
  - Subdivide along each axis into four subsquares.
  - Repeat subdivision process on each subsquare until:
    - A predefined maximum level is reached
    - The square contains fewer the some threshold number of points / objects.
  - http://www.cs.wustl.edu/~suri/cs506/projects/quad.ht

# *Octrees*

- 3D extension of quadtrees.
  - Start with axis aligned bounding cube.
- Octrees are a great structure, but...
  - Can be **major** memory hogs
    - Complete 10-level tree >150 **million** nodes
  - Traversal can be tricky
  - Objects must be carefully assigned to nodes

# *k-d Trees*

- Cousins of octrees and BSP trees.
- Each node in the tree picks an axis to split along.
  - If the selection axes are the X/Y/Z axes, three levels of a k-d tree are like one level of an octree.
  - Each node can pick *any* axis to split along.
    - Care must be taken to prevent the subspaces from being to “oblong” (a.k.a., thin)

# References

http://www.cs.cmu.edu/~awm/animations/kdtree/

- Links to more animations and other resources.

http://en.wikipedia.org/wiki/K-d_tree

- The wikipedia entry is *very* good.

# *Ray-based Tree Traversal*

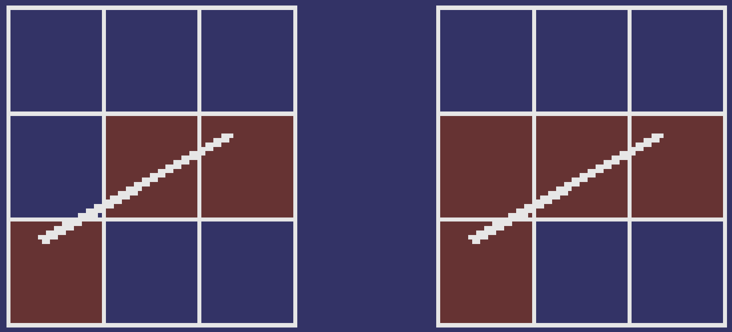
- Given a partitioning scheme, how can we visit all nodes along a line?

# *Ray-based Tree Traversal*

- Given a partitioning scheme, how can we visit all nodes along a line?
- For quadtrees, octrees, and k-d trees, it's fairly simple.
  - Use line equation $S(t) = A + td$
  - Calculate $t$ where the splitting planes intersect line.
  - If $0 \leq t < t_{max}$, search the node on that side of the split.
  - Repeat until no nodes are in range (or you reach the leaf nodes).

# *Ray-based Grid Traversal*

- Much like drawing anti-aliased lines.
  - Need to visit every cell intersected, not just the ones “most” intersected.



# *Ray-based Grid Traversal (cont.)*

- Can calculate the distance to the next x-boundary ($\Delta tx$) intersection and the next y-boundary ($\Delta ty$) intersection.
  - The closest intersect determines which neighbor cell to visit next.
  - At each step add $\Delta tx$ to *tx* or $\Delta ty$ to *ty* and subtract from the other delta.

$$\Delta tx = M \frac{\sqrt{dx^2 + dy^2}}{dx}$$

$$tx = (x_{max} - x_{initial}) \frac{\sqrt{dx^2 + dy^2}}{dx}$$

# *Next week...*

- BSP trees, part 1
- Assignment #3, part 1 due

# Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.
- OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.
- Khronos and OpenGL ES are trademarks of the Khronos Group.
- Other company, product, and service names may be trademarks or service marks of others.