DEPARTMENT OF
INFORMATION SYSTEMS
AND COMPUTER SCIENCE

Quadtrees

Hierarchy + Collision Checking

Lecture Time!

- Size: Matters
- Quadtrees: Multiple Layers of Uniform Grids?
- Implementation: Lists Again?

Size Does Matter

- Uniform grids can still work for games with objects of varying sizes
  - They just don't work that well

Space Patrol Luluco, episode 5

Size Does Matter

- Logic dictates that larger objects run a higher risk of colliding with other objects
  - Large objects occupy more cells in in a uniform grid

Size Does Matter

- Logic also dictates that small objects are less likely to be colliding with other small objects but they are more likely to collide with larger objects anyway
  - Small objects occupy fewer cells, possibly those also occupied also by large objects

Back to Uniform Grid

- Assume only one cell to cover the entire game world
  - In other words, ye olde brute force pairwise collision checking
  - We will call it grid A
- Assume another uniform grid, this time with 4 cells
  - We will call it grid B

Back to Uniform Grid

- Assume that objects can only:
  - Occupy only a single cell in grid B (it's relatively small) or
  - Occupy the single cell in grid A (it's relatively large... or poorly placed)
- Assume both grids A and B are active
  - For an object in each case above, what condition/s must another object satisfy in order to be considered for pairwise collision checking with that object?

Back to Uniform Grid

- Let's go further and assume another uniform grid with 16 cells
  - We will call it grid C
- An object can only occupy a single cell in only one of the three grids
- Assume grid C is now also active
  - How should we handle collision checking?

Quadtrees

- The *quadtree* is a tree-based, axis-aligned hierarchical spatial partitioning method of an area of 2D world space
- As the name suggests, each parent node in the quadtree has four children
- The root node is generally the smallest axis-aligned square that covers the entire game world

Quadtrees

- The area occupied by the root node is subdivided into four smaller equal-size squares (aka cells or *quads*)
- These four smaller areas are the child nodes of the root node
- These child nodes are also subdivided in the same way, and so on
  - Typically stops when the tree reaches a maximum depth or when the squares become smaller than a certain size

Quadtrees

- A node in a quadtree usually contains the following information (or there should be an easy way to derive this information):
  - Center (center point of the AABB representing the node)
  - Half-width (or "radius" of the AABB)
  - Pointers to its 4 children nodes and to its parent node
  - List of objects in this node (same as a cell in uniform grid)

Quadtrees

- Remember that there are many ways to implement a tree
- Since we know that each node in a quadtree has exactly 4 children (or 0 if the node is at a certain depth), you can also use an array-based implementation for the quadtree
  - This removes the need for pointers to children nodes but requires you to work with indices instead

Quadtrees

- An object can only be placed in a node that covers them completely
  - If an object is larger than a particular node or it is overlapping that node's edge, that object should not be placed in that node's list
  - It is possible for a very small object to be placed in the root node if that object is situated in or near the center of the world

Example

- Assume a quadtree of depth 3 (like our theoretical example earlier with grids A, B, and C)
- World size is 320x240
  - Size of root node? (Note: Square shape preferred)
  - Size and number of cells in second layer?
  - Size and number of cells in third layer?

Example

- Note: Changing world size to 320x320 to keep things simple; assume (0, 0) is upper-left corner of world
  - Object #1 is a rectangle of size (30, 20) and its center is located in (120, 130)
  - Object #2 is a rectangle of size (240, 100) and its center is located in (159, 160)
  - Object #3 is a rectangle of size (60, 20) and its center is located in (70, 40)
- Which cell should each object be placed?

Creating the Quadtree

- While it is possible to dynamically create quadtree nodes whenever they are needed, let us assume an easier static implementation
- Quadtree creation can be done using a recursive function
  - Start by defining the root node

Creating the Quadtree

- Root node parameters:
  - Center is the center of the screen
  - Half-width is half the screen width or height, whichever is greater
  - Pointers to parent and child nodes may be necessary except in an array-based implementation
  - List is initially empty (obviously)

Creating the Quadtree

- Each child node's parameters:
  - Center is +/- half of the root's half-width to root’s center’s x and y
    - Ex. + to x and - to y to get the upper-right quad
  - Half-width is half of the root's half-width
    - Solved for that already; see above

Creating the Quadtree

- Each child node is another quadtree with different parameters for its "root"
  - Center and half-width already defined (see previous slide)
  - Pointers to parent and child nodes may be necessary except in an array-based implementation
  - List is initially empty (obviously)

Creating the Quadtree

- You may then pretend that a node is the root for its own children
  - Refer to the previous 2 slides
- But you have to stop somewhere eventually
  - Limit depth to 8 (6 for octrees used in 3D)

Updating the Quadtree

- Given an object, where should it be placed in the quadtree?
- Updating the quadtree lists can also be done recursively
  - Start by setting the root node as the "current" node

Updating the Quadtree

- At the current node, check if the object can be contained completely in one of its child nodes
  - How?
  - Assume that all objects can be contained completely in the root node

Updating the Quadtree

- If it cannot be contained in any of the child nodes OR you are already at one of the leaf nodes, add it to the list of the current node
  - The leaf check should be done first :P
- If it can AND you are NOT at one of the leaf nodes, set that child as the "current" node and repeat the process
  - Refer to the previous slide

Updating the Quadtree

- An object should also contain a pointer to or the index of the quadtree node it occupies
  - This allows you to:
    - Easily remove the object from its quadtree node list if it has to change nodes and
    - Skip adding the object to a list that it is already in
  - Remember: An object should only occupy ONE node

Collision Checking

- For each object, collision checking involves traversing the quadtree either towards the root (easier) or towards the leaves:
  - Start by setting the node occupied by the object to be the "current" node
  - Do a pairwise collision check with each object in the list of the current node

Collision Checking

- For each object, collision checking involves traversing the quadtree… (continued):
  - Assuming traversal towards the root, set the parent node as the "current" node and go back to the previous step
    - Stop if you're at the root node already

Collision Checking

- While the quadtree helps immensely, there are still redundant/useless collision checks that you should NOT perform

Bananya, episode 10

Collision Checking

- Given an object, when going through the list in the node that the object itself occupies:
  - That object should NOT check for collision with itself
  - That object should also NOT check for collision with other objects located in an "earlier" address or index in the overall list of objects

Homework

- Create a program that can simulate a LOT of "air hockey pucks" (1000+)
  - Circle information will be obtained via standard input
  - First line contains number of circles N
  - Each of the next N lines contains the circle's x-coordinate, y-coordinate, and radius (in that order)

Homework

- It's pretty much the physics collision homework on steroids
  - Each circle should have a mass equal to its area (radius * radius * pi)
    - Hint: You should also store the reciprocal of the mass of each circle for easy access (the mass itself is usually not stored)
  - Friction toggle is OPTIONAL

Homework

- All circles should be the same color
- A circle can be selected by left-clicking it
  - The selected circle should have a different color
  - This will also deselect any previously selected circle
- The selected circle can be accelerated using keyboard input
  - By holding down WASD or directional keys

Homework

- Note that you may have to perform other optimizations
  - Reduce number of arithmetic operations per frame
- If you notice circles "melding" into each other, it is highly likely that there is something wrong with the way you handle quadtree traversal
  - Double-check using brute force