DEPARTMENT OF
INFORMATION SYSTEMS
AND COMPUTER SCIENCE

Uniform Grids

Because Pairwise Checking is Expensive

Lecture Time!

- Collision Checks: AABB++
- Grids: Uniformity for Easy Cell Access
- Issues: Weaknesses
- Implementation: Lots of Lists

Early Exit

- Pairwise collision checks are expensive
- AABB's can be used for an early exit
- But even the early exit checks will add up if you are performing overlap checks for hundreds or thousands of objects

Demo

- Brute force pairwise collision check using AABB's
  - Each shot is checking for collision with each brick
  - Look at that horrible framerate!

Consider

- Given typical object behavior in real-time games, overlap checks are more likely to return false (not colliding)
- This is why AABB's "early exit" functionality is so useful
  - More likely to trigger and prevent the more expensive overlap checks from running
  - AABB overlap checks make it obvious if objects are not colliding

Earlier Exit

- Human sight allows us to easily tell if objects are colliding or not
- If an object is nowhere near another, obviously they aren't colliding
- We only look closely (perform a more expensive check) if the objects are near enough that they occupy a common region

Earlier Exit

- Is it possible to skip a pairwise collision check entirely?
- If so, what would be the conditions?
- More importantly, how would you make the program skip the check?
  - This also means skipping the AABB check!

Hmmm...

- Human sight allows us to easily tell if objects are colliding or not
- If an object is nowhere near another, obviously they aren't colliding
- We only look closely (perform a more expensive check) if the objects are near enough that they occupy a common region

Uniform Grid

- World space can be overlaid with a regular grid, effectively dividing the world into uniform-sized cells
- Each object "occupies" cells that it overlaps with
- Perform pairwise collision checks between objects that share a common cell

Back to the Demo

- Number of pairwise collision checks between X shots and Y bricks?
  - It could be worse --- what if this was your air hockey homework, but with thousands of pucks?
- But it's clear that you should only check for shot-to-brick collision when a shot is near enough to a brick
  - And only check for collision with that brick

Sneak Peek

- But what if the game implemented a uniform grid and only performed pairwise collision checks between objects that shared a common cell?
- By eliminating a significant number of collision checks, game performance improves significantly

Cell "Overlap" Check

- Assume that your world space is the screen only
  - Grid cells can exist outside the screen, but let's ignore them for now
- Given a point, how do you know which cell it is in?

Cell "Overlap" Check

- Screen size: 800x600
- Grid cell size: 20x20
- Points: (356, 288) (555, 111) (700, 400) (0, 0)
  - Note: (0, 0) is upper-leftmost pixel
  - You may assume [0, 0] is upper-leftmost cell

Cell "Overlap" Check

- But objects are more than just a point
- What if you had a circle?
- What if you had a rectangle?
- What if you had a polygon?

Since We Mentioned AABB's

- Screen size: 800x600
- Grid cell size: 20x20
- AABB#1 min: (234, 432) max: (246, 442)
- AABB#2 min: (111, 333) max: (333, 555)

That's a Lot of Overlap

- Need to use an appropriate cell size!
- There are four issues related to cell size that can hamper performance:
  - The grid is too fine
  - The grid is too coarse (with respect to object size)
  - The grid is too coarse (with respect to object complexity)
  - The grid is both too fine and too coarse

Too Fine

- If the cells are too small, a large number of cells must be located and updated for a given object
  - Takes both extra time and extra space
  - Gets worse if a moving object is involved

Objects Too Small

- If the objects are small and the grid cells are large, there will be many objects in each cell
  - This will result in a much higher number of pairwise collision checks
  - Worst-case scenario if all objects occupy one cell

Objects Too Complex

- Even with an appropriate grid cell size, objects with complex shapes could still cause problems
  - Better to reduce the grid cell size and break the object's shape into smaller pieces

Too Fine and Too Coarse

- If the objects are of greatly varying sizes... tough luck
  - Might need another spatial partitioning method (like quadtrees)

Implementation

- Cell size is generally large enough to fit the largest object at any rotation
  - Just need to make sure that objects will overlap only four cells at most (in 2-D)
  - Note: No rotation means you can afford a smaller cell size

Implementation

- Easiest way to implement involves assigning a list to each cell
  - As objects move about, update these lists with references to these objects
  - NOT copies of objects as this will consume too much memory

Implementation

- Also have to make sure that a pair of objects is only tested ONCE for collision per frame
  - Another set of lists, one for each object?
  - A bit array, each bit for a pair of objects?
  - Sometimes we're already guaranteed that a test won't be repeated
    - Depending on what needs to be tested, what the collision response is, and when the response is applied

Homework

- Duplicate the relatively lag-free demo
  - There are 3000 bricks
  - A maximum of 2000 shots can be present on-screen
  - Collision check is only to see if a shot hit a brick

Homework

- You may use different colors
- As long as the size is the same for an object type, you can change the object sizes
- Make sure speed of moving objects does not exceed its size since this may cause objects to pass through each other and not register collisions

Homework

- Note that you also have to perform other optimizations
  - Objects that are invisible or off-screen shouldn't be drawn
  - Figure out what should go into the uniform grid lists (hint: not everything)