"If a tree falls in a forest and no one is around to hear it, does it make a sound?"
If no one is around the tree, no one cares!
Lecture 6
Interest Management
aka Relevance Filtering
aka Data Distribution Management
Continuous state update:
each event triggers updates to all other players
Periodic state update:
consolidated state updates sent to players periodically
Problem: cannot scale to a large number of players
Idea: only need to update another player $p$ if the update matters to $p$. 
Question: which update matters to which player?
The Aura-Nimbus Information Model
Aura
Nimbus / Area of Interest (AOI)
(space where a player can perceive)
Update of $p$ matters to $q$ if the aura of $p$ intersects nimbus of $q$. 

Updated 10 Sept 2012
The Publish/Subscribe Communication Model
Entity **publishes updates**
Players **subscribe to entities**
Multicast: send a message to a set of subscribers
Group: a channel to publish messages
A client can subscribe to/join a group to start receiving messages from that group.
A client can unsubscribe from/leave a group to stop receiving messages from that group.
Anyone can send a message to a group (need not be a subscriber).
a group

a subscriber
Distance-based Interest Management
Update of $p$ matters to $q$ if $p$ and $q$ are within certain distance from each other
Naive $O(n^2)$ implementation

each player is a group

for each player $p$

for each player $q$

if $p$ and $q$ are close

add $p$ to $q$’s subscriber

add $q$ to $p$’s subscriber
Possible to use advanced algorithms / data structure to improve the performance, but
observation: it is OK to send extraneous updates to a player
Cell-based Interest Management
Approximate distance-based IM with rectangular cells
Each cell is a group.
Naive $O(n)$ implementation

each cell is a group

for each player $p$

for each nearby cell $c$

if $p$’s AOI overlaps with $c$

add $p$ to $c$’s subscribers

add $p$ to $c$’s publishers
Large cell: More extraneous messages.
Small cell: Large management overhead.
The white player will receive many messages he/she is not interested in.
**Idea**: adaptive cell size: partition the cells into smaller ones as needed.
Quad Tree: Partitioning a cell into four smaller cells until entity density is small enough.

Each leaf node is a group
Publish/subscribe decision is done hierarchically.

if overlaps with parent, check if overlaps with children
Cell-based IM does not consider occlusion common in FPS games
Visibility-Based Interest Management
Update of $p$ matters to $q$ if $q$ can see $p$, and $pq$ are within certain distance from each other
Without considering visibility
With visibility constraint, updates from white entities are not sent.
Ray Visibility
Interest Management
Object-to-Object Visibility

1. Expensive
2. Frequent re-calculations.

but gives exact visibility.
Update of $p$ matters to $q$ if $q$ can see $p$'s cell, and $pq$ are within certain distance from each other.
Object-to-Cell Visibility
Object-to-Cell Visibility

1. Less expensive
2. Less frequent re-calculations
3. Less accurate
When player moves, still need to recompute visible cells.
Update of $p$ matters to $q$ if $q$'s cell can "see" $p$'s cell, and $pq$ are within certain distance from each other
i.e., there exists a point in $p$’s cell that can see a point in $q$’s cell, and $q$ is near $p$. 
Cell-to-Cell Visibility
Cell-to-Cell Visibility

1. Much Less expensive
2. Calculate once!

but even less accurate.
Computing Cell-to-Cell Visibility
Check if there exist two points, one in each cell, that can see each other (can draw a line without passing through occlusion)
Trivial case: if two cells are adjacent and the boundary is not completely occluded.
Build a graph of cells -- connect two vertices if they share a boundary and is visible to each other.
if two cells are not-adjacent, then for them to be visible to each other, there should exist a path between them, and ...
consider the non-occluded boundaries along path..
The set of points on the left $L$ and right $R$ can be separated by a line.
The set of points on the left $L$ and right $R$ can be separated by a line.
Linearly Separable Point Sets
We can model this problem as a set of linear equations.

\[(x_1, y_1), (x_2, y_2)\]

\[ax + by - c = 0\]
Find a solution \((a, b, c)\) for the following:

\[ ax_1 + by_1 - c > 0 \text{ for all } (x_1,y_1) \text{ in } L \]
\[ ax_2 + by_2 - c < 0 \text{ for all } (x_2,y_2) \text{ in } R \]

The line that separates is \(ax + by - c = 0\)
Two non-adjacent cells are visible to each other if there exists a path between them, and the set of points constituting the L and R sides of the portals between cells are linearly separable.
We can break into smaller cells if occlusion is not aligned with boundary of cells.
(Irregular) triangular cells can adapt to any polygonal occlusions.
Note: Rendering engine usually computes visibility information, which we may be able to reuse in the Interest Management module.
Generalized Interest Management
Update of $p$ matters to $q$ if $q$ is “interested” in $p$ based on a set of attributes
Example: Interested in
(i) objects around avatar
(ii) buildings in a region
(iii) the opponent’s avatar
Subscription can be based on any attribute (not just position)
We can view each object as publishing into a k-dimensional space (each attribute is a dimension) call update region.
A subscription specifies a region in the same space.

Messages from an update region $u$ is sent to a subscription region $s$ if $s$ and $u$ overlaps.
Example in 2D with rectangular aura (update region) and nimbus (subscribe region)
How to test if two regions overlap in k-dimensional space?
Dimensional Reduction

If two regions overlap, then they overlap in each of the individual dimension.
Naive $O(nm)$ implementation

Each entity is a group

For each update region $p$
  
  For each subscribe region $q$
    
    For each dimension $d$
      
      Check if $p$, $q$ overlap in $d$-th dimension

    If $p$ and $q$ overlap in every dimension
      
      Send published message of $p$ to $q$
Sort-based DDM Algorithms
For each dimension,

Step 1: Sort all end points and put into a list $L$. 
Step 2: Scan from left to right. Remember all active subscription regions $S$ and all active update regions $U$. 
Active Subscriptions: S1
Active Subscriptions: S1, S2
Active Subscriptions: S1, S2
Active Update Regions: U1
We can determine the overlaps when we process the endpoint of a region.
Active Subscriptions: S2
Active Update Regions: U1
S1 overlaps U1
Active Subscriptions: none
Active Update Regions: U1
S2 overlaps U1
If we encounter the endpoint of a subscription region, then it overlaps with all active update regions.

If it is the endpoint of an update region, then it overlaps with all active subscription region.
O((n + m)\log (n + m))
for sorting

O(n + m)
to scan
Note: storing overlap information still takes O(nm) since in the worst case there are O(nm) overlaps.
Temporal Coherence

Changes to value of an attribute is small between two consecutive time steps.
$O((n+m) \log (n+m))$ to pre-sort the data

$O(n + m)$ for sorting (insertion sort)

$O(n + m)$ to scan
Only regions that are swapped during insertion sort need to update their overlap set.