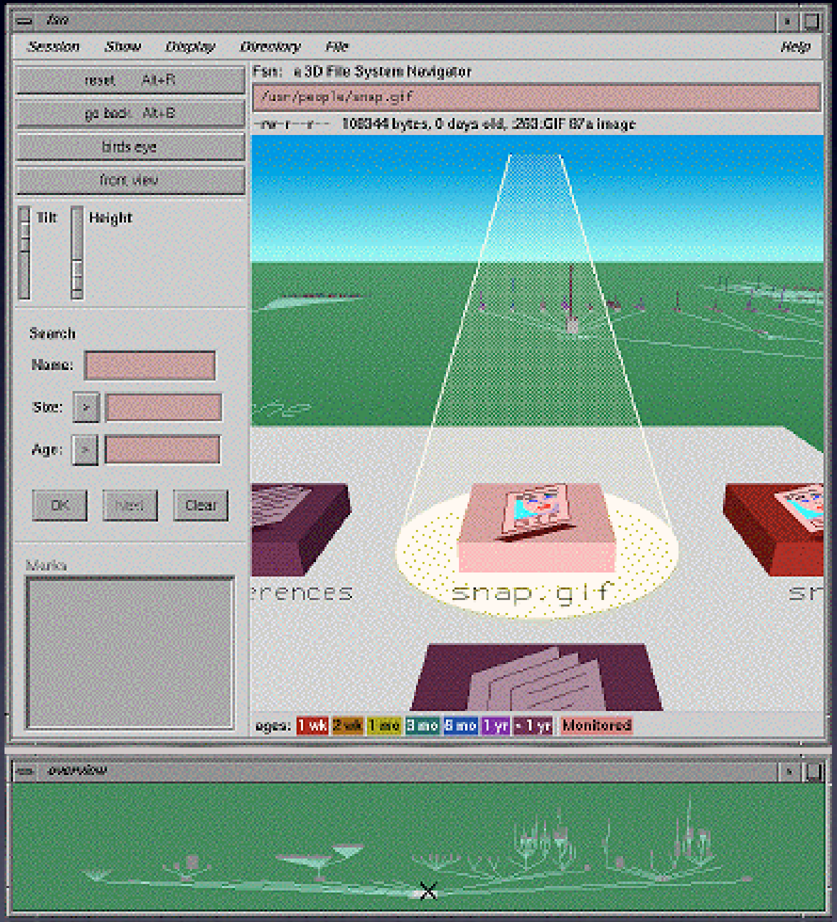
Visibility Preprocessing for Interactive Walkthroughs

The Setting

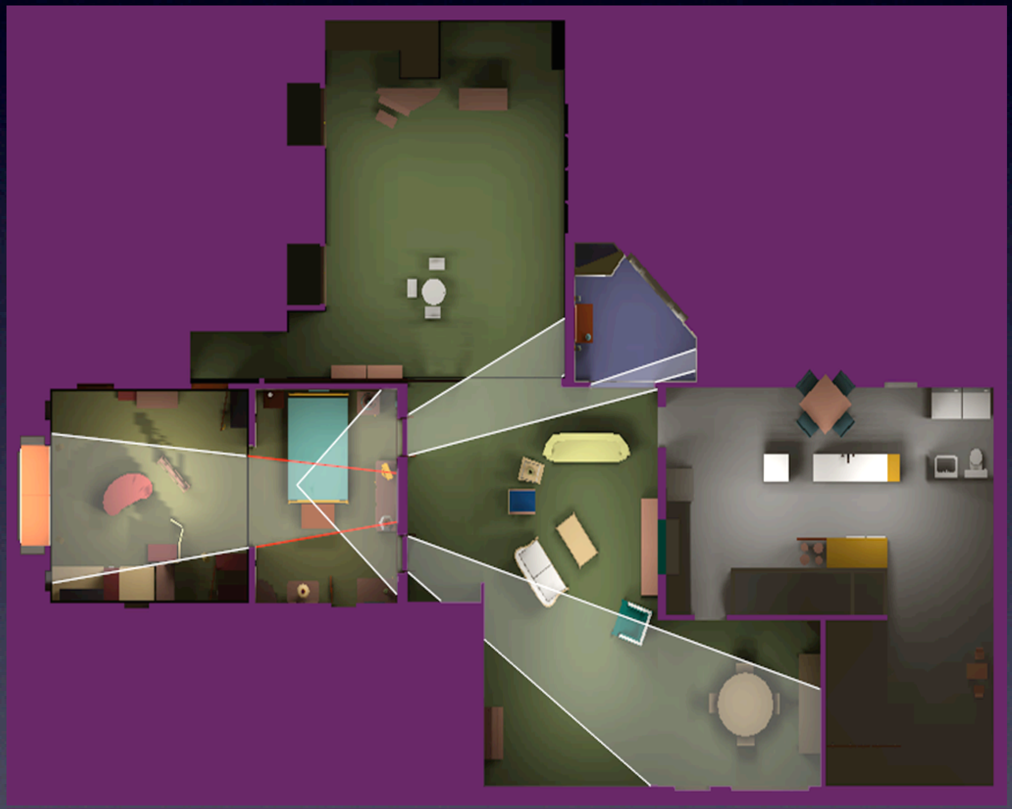
- 1991
- Second generation graphics hardware (SGI)
- Hardware transform, polygon rasterization, z-buffer



Application

Visibility Problem

- How do we ensure that the front-most polygon is used to shade the pixel?
- For efficiency - how can we shade as few fragments per pixel as possible (ideally only 1)?



Precomputed Visibility

- Many existing solutions for exact visibility
  - But can be complex, in terms of data and implementation
- BSPs - potentially O(n^2) new polygons
- Quad-tree, Octree - common in raytracers, not so great for interactive applications

Precomputed Visibility

- But we have a hardware Z-buffer now
- Maybe we don't have to be exact

Precomputed Visibility

- Existing visibility approximations
  - Quadtree/octree frustum culling
  - Portal shadow volumes
  - Discrete sampling

Goals

- Conservative
- But not too conservative
- Reasonable precomputation time
- Reasonable precomputed data size
- Allow further view frustum culling at runtime

Outline of Approach

- Precomputation
  - Spatial subdivision (cell and portal)
  - Conservative visibility (sightlines)
- Runtime
  - View cone culling

Assumptions

- Axial faces - not necessary, but simplifies implementation
- 2D - extension to 3D described but not tested
- Ignore small/insignificant objects

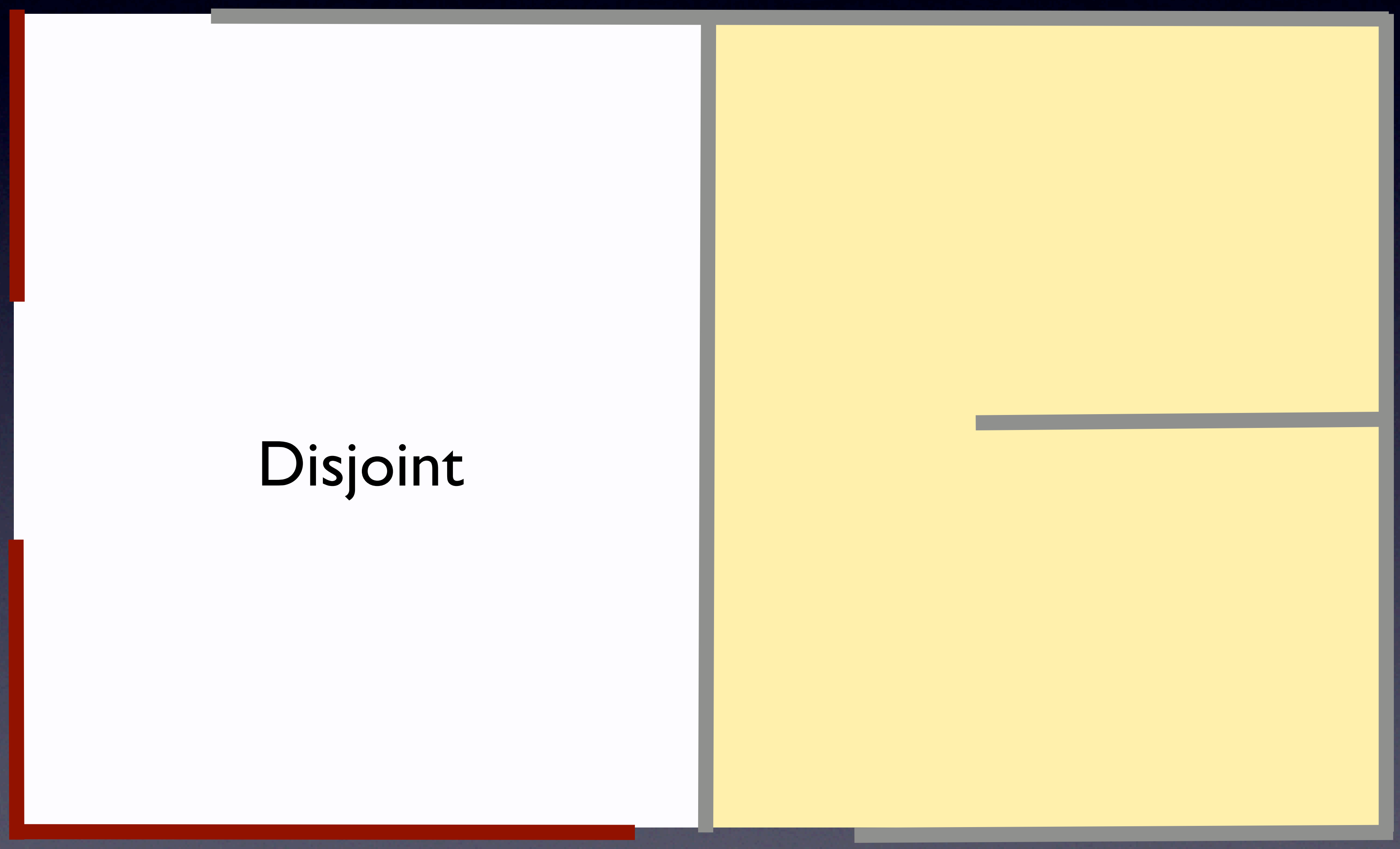
Spatial Subdivision

- Requirements
  - Convex cells
  - Point location support
  - Portal enumeration
  - Neighbor finding

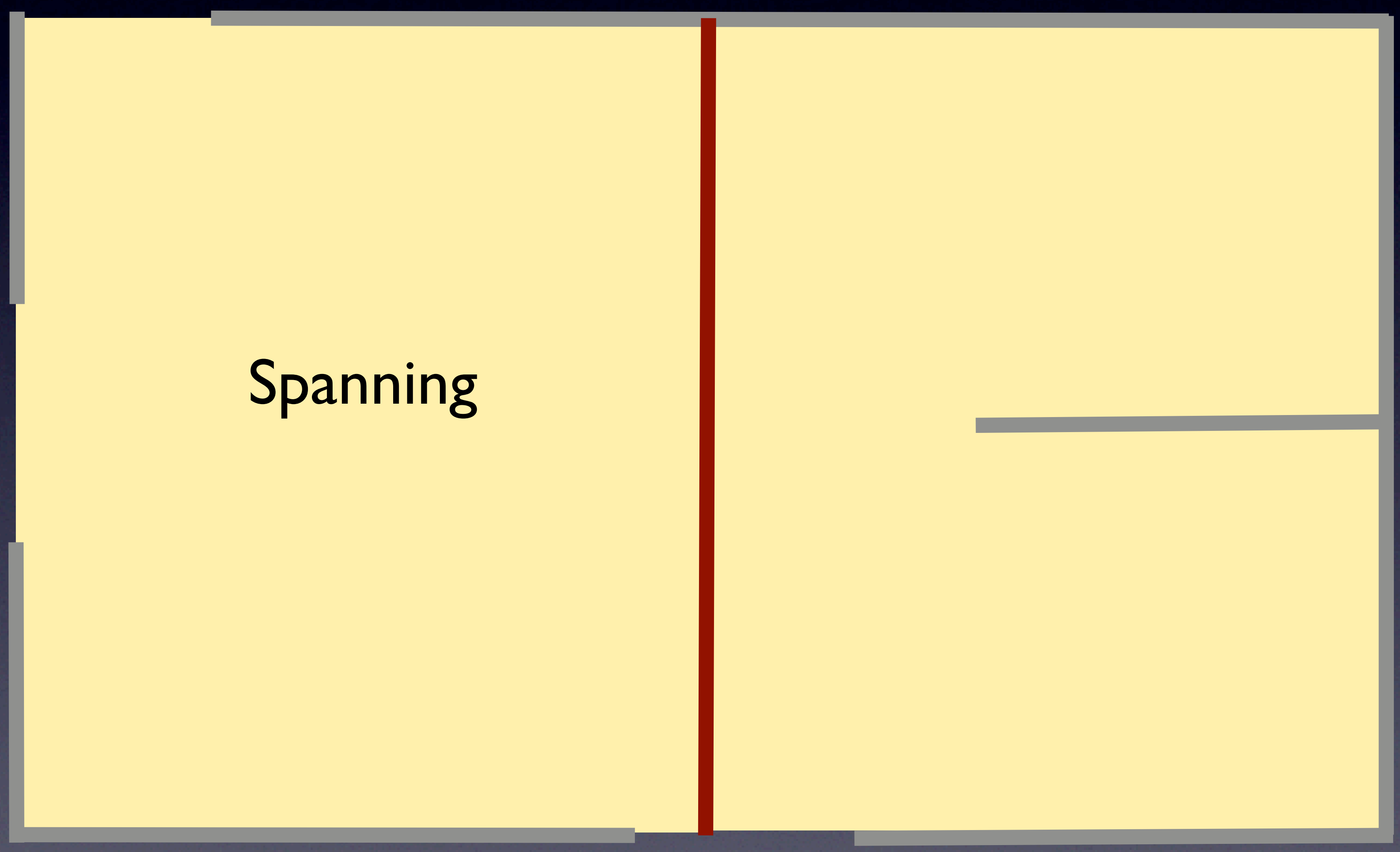
Spatial Subdivision

- Axis aligned faces and portals
- k-D tree

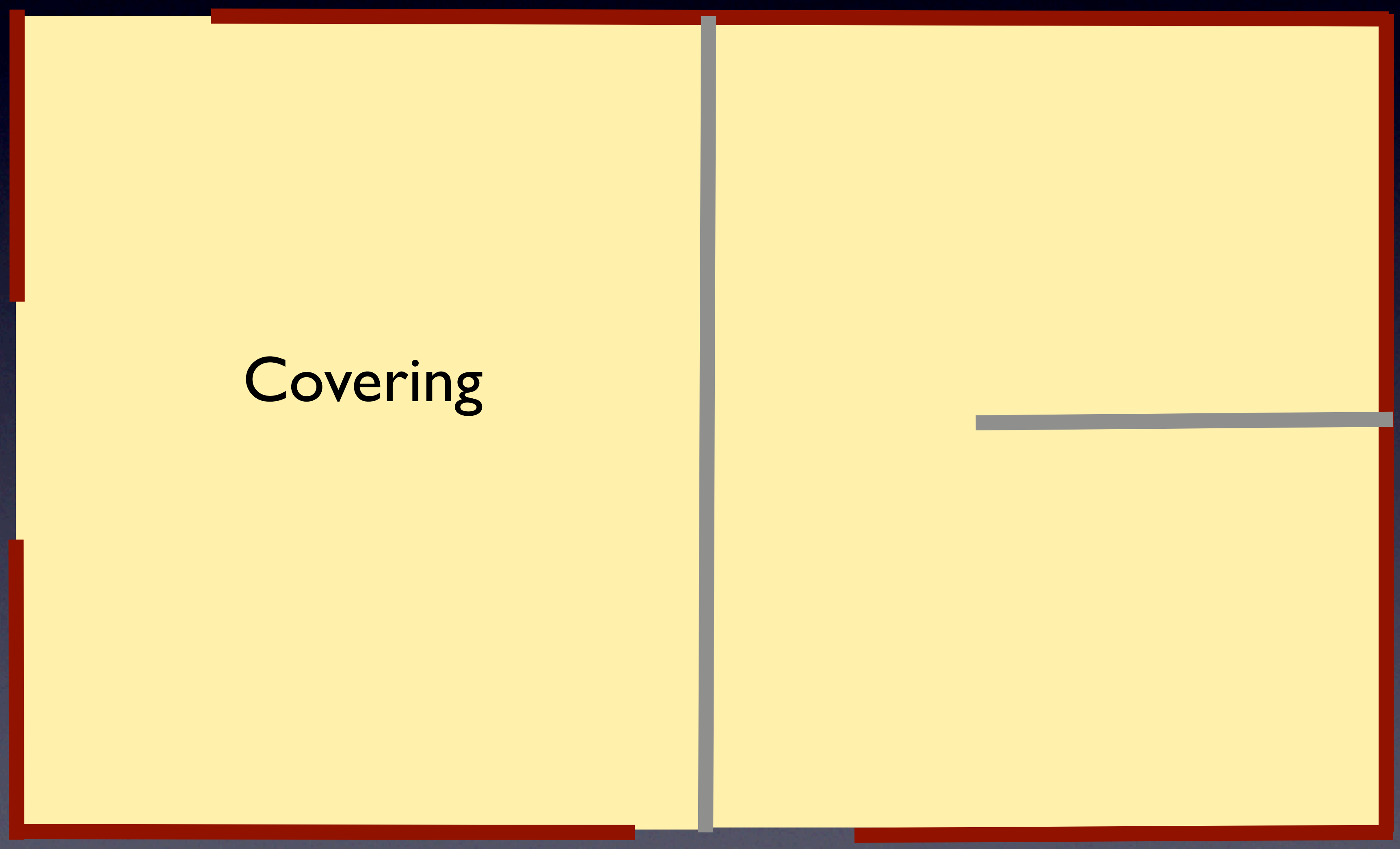
Spatial Subdivision



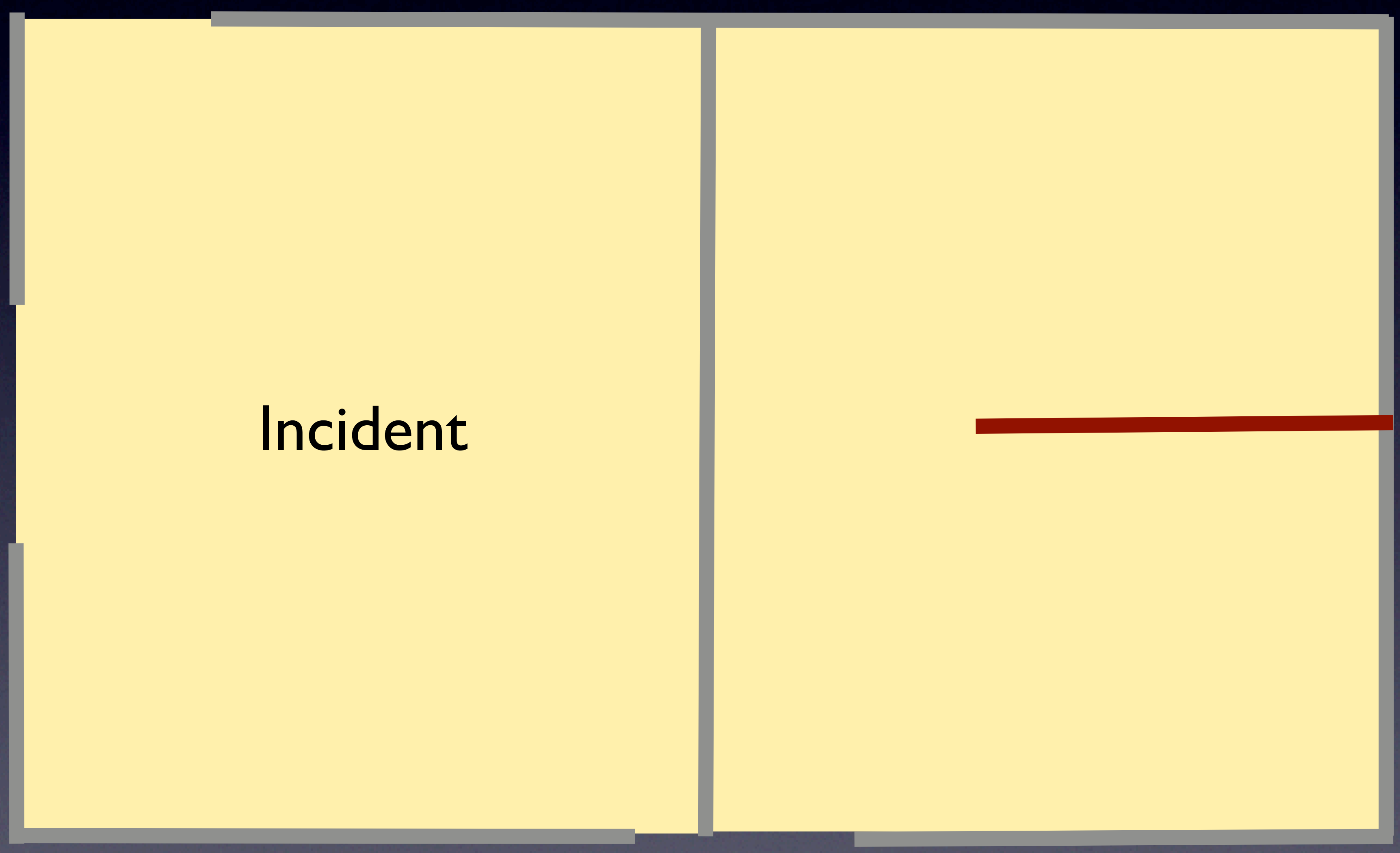
Spatial Subdivision



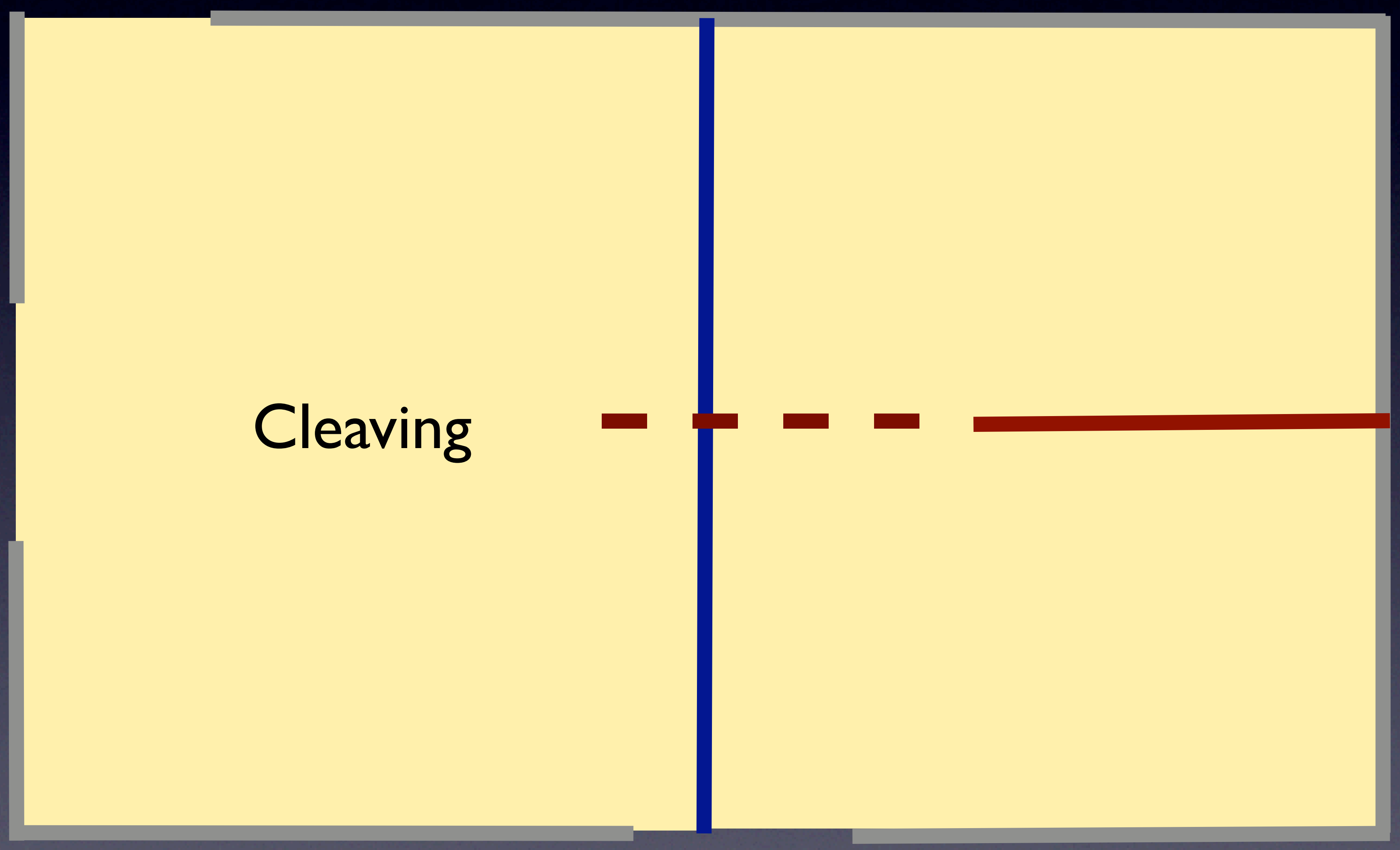
Spatial Subdivision



Spatial Subdivision



Spatial Subdivision

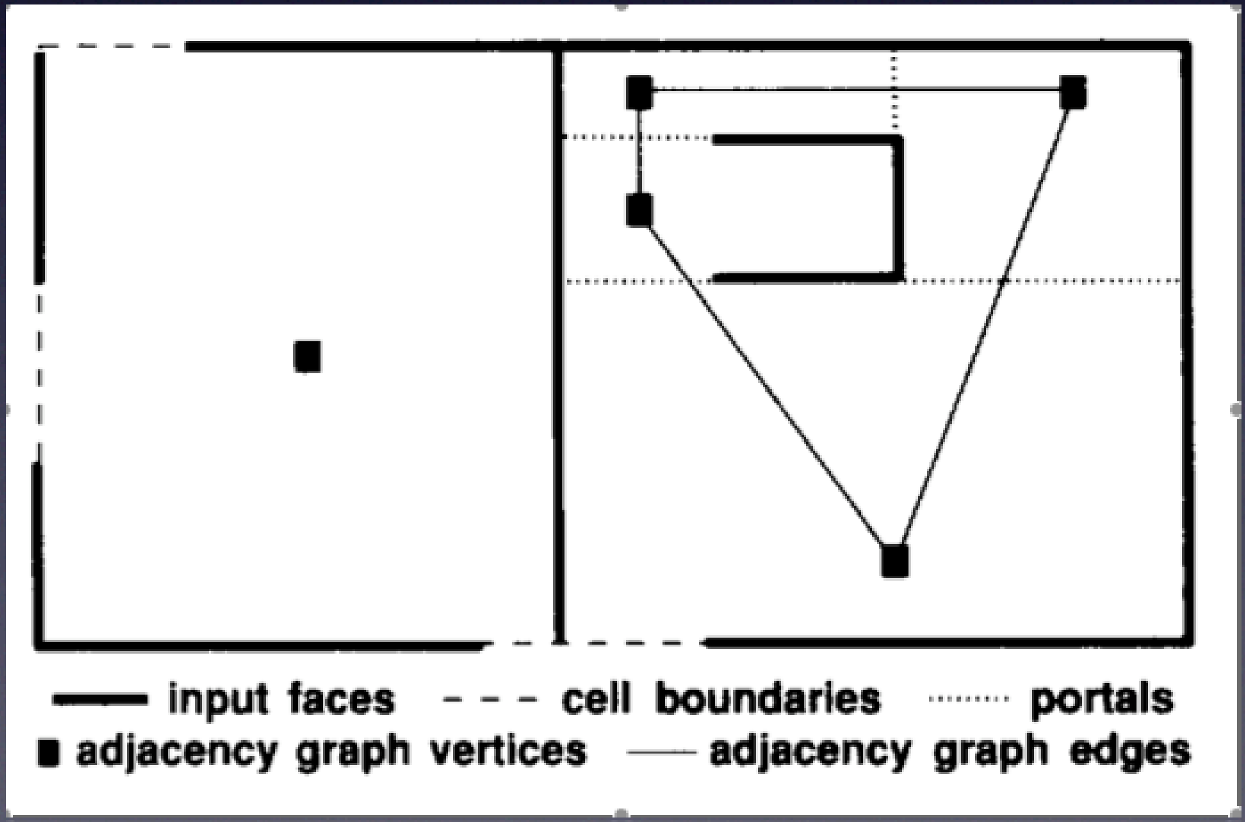


Spatial Subdivision

- Recursively subdivide leaf nodes
  - If no incident or spanning faces, stop
  - If any spanning faces, split on median spanning face
  - Else, split along median, sufficiently obscured, minimum cleaving face

Spatial Subdivision

- Store (portal, neighbor) pairs at leaf nodes
- Effectively have adjacency graph



Finding Sightlines

- Problem: Given set of portals forming path from one portal to another, how do determine if there is a sightline that passes through all those portals.

Finding Sightlines

- Key insight: orient portals
- All portal left end points must be on positive side of the line
- All portal right end points must be on negative side of the line

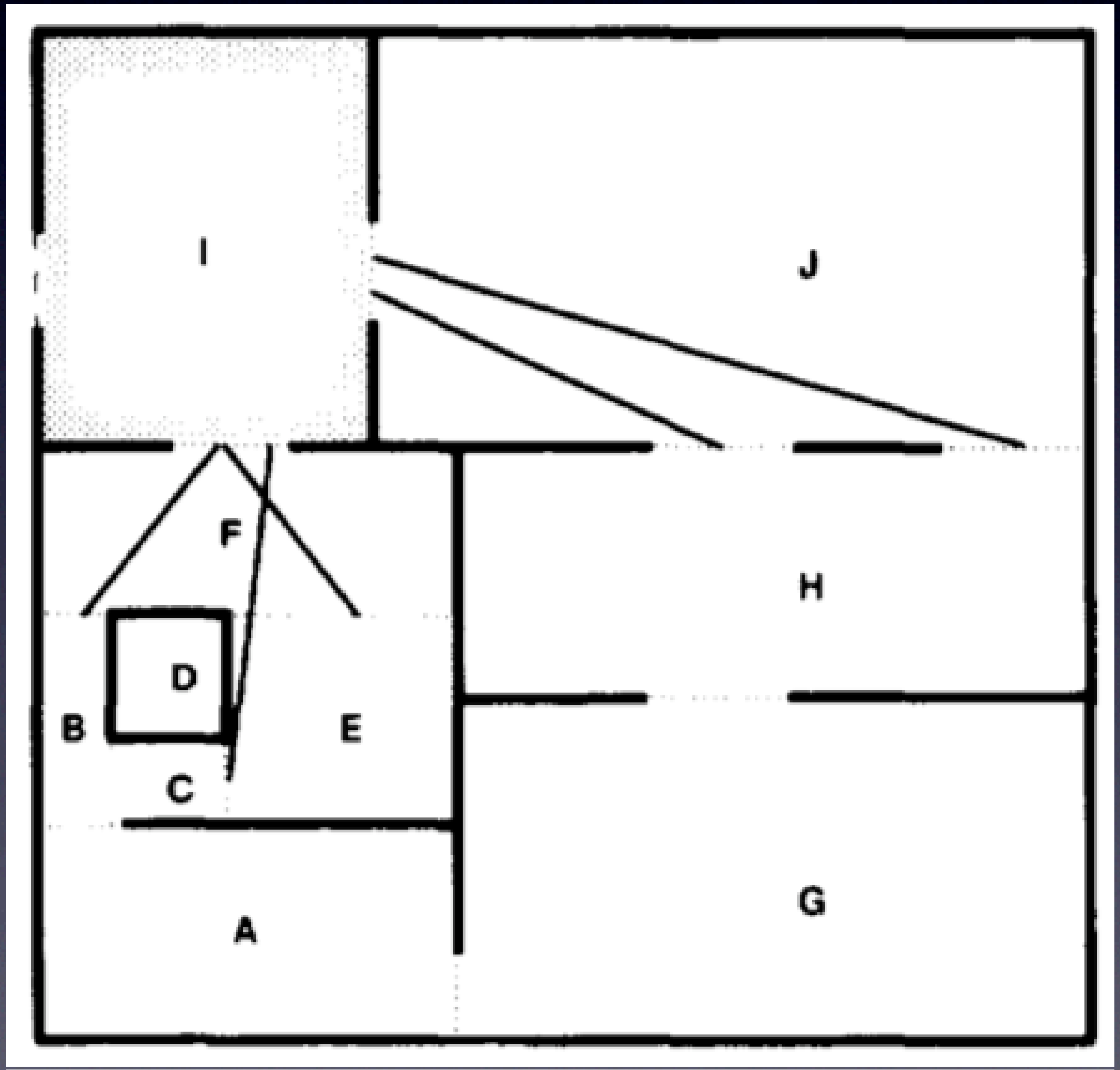
Finding Sightlines

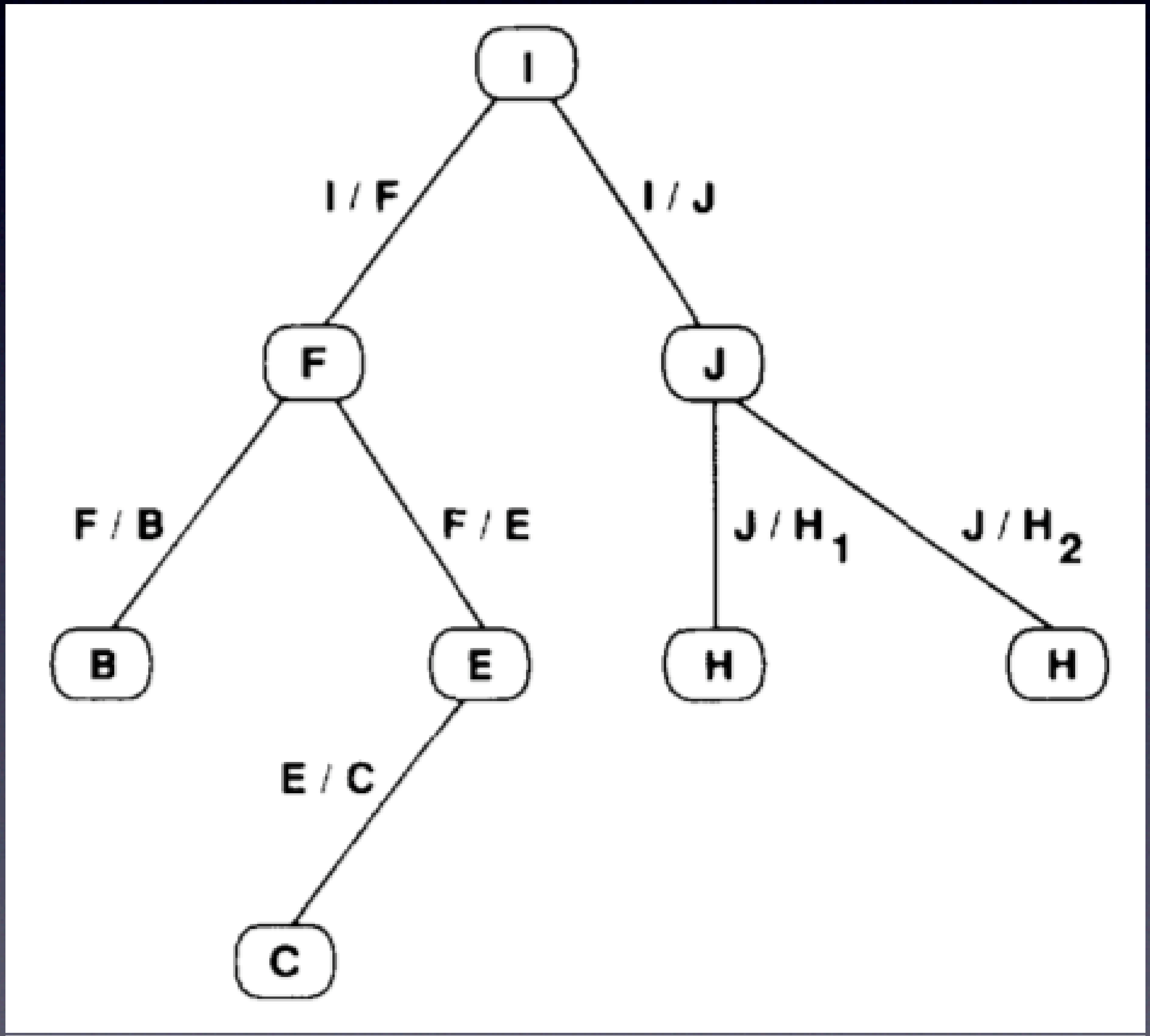
- Unknown line: **S**
- Constraints:
  - **S** . **L** >= 0 for all portal left points
  - **S** . **R** <= 0 for all portal right points
- This is a linear programming problem.

Cell to Cell Visibility

- Given a cell and the sightline algorithm, we need to find, for each cell, all cells visible from it
- Use adjacency lists to traverse graph depth first
- At each cell, recurse only if sightline test is positive

Cell to Cell Visibility



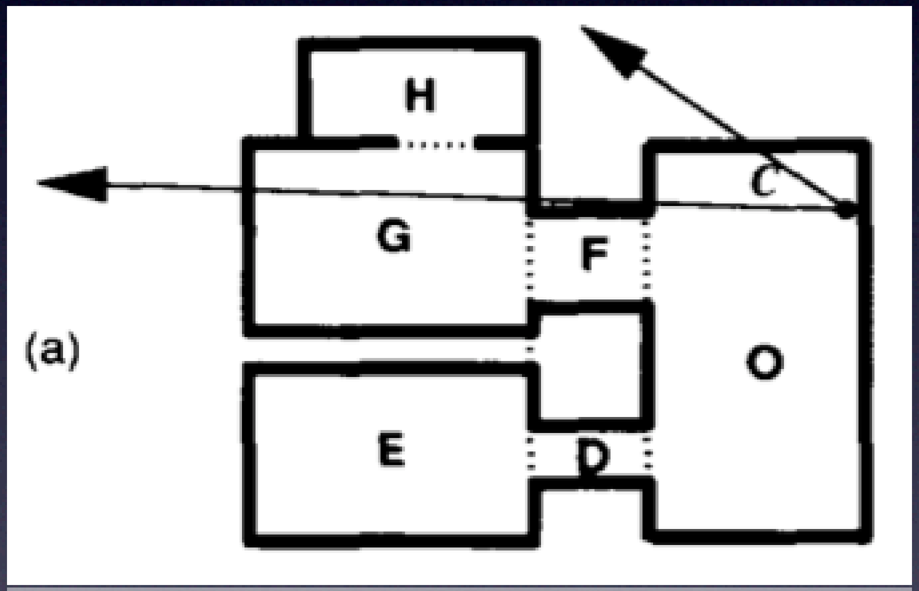


Eye to Cell Visibility

- Cell to cell visibility clearly a superset of eye to cell visibility
- So we can just cull based on the view frustum
- A few methods are presented, increasing in accuracy

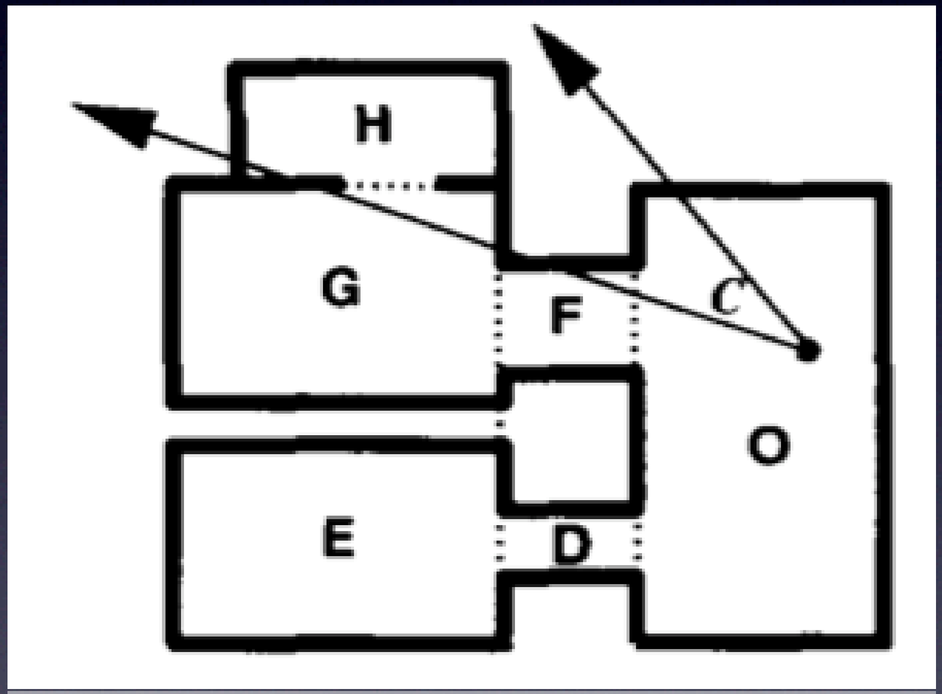
Eye to Cell Visibility

- Disjoint cell
  - For each potentially visible cell
  - Discard if cell intersection with frustum is empty



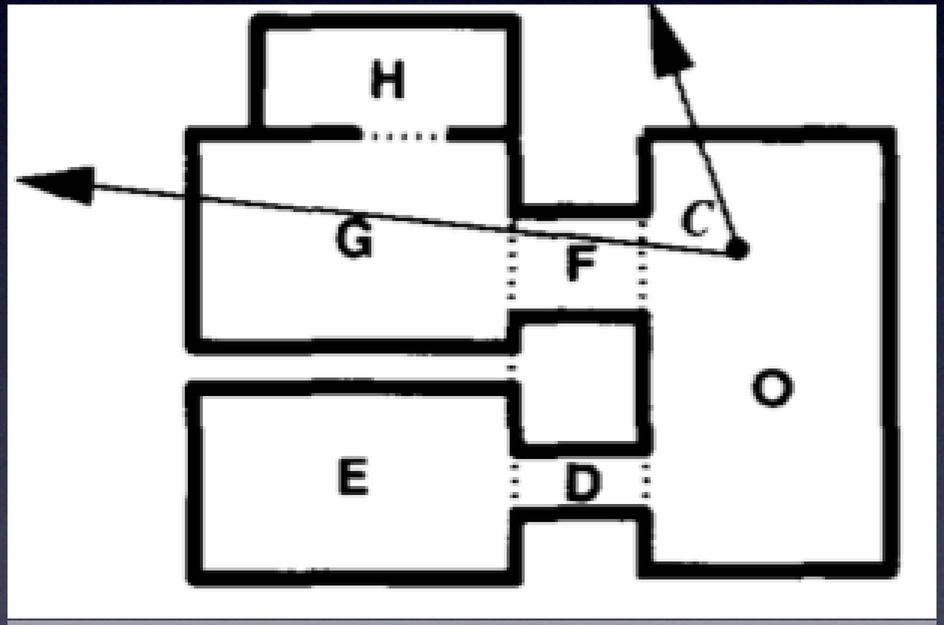
Eye to Cell Visibility

- Connected Component
    - DFS on stab tree
    - Only recurse at cell if it intersects with view frustum



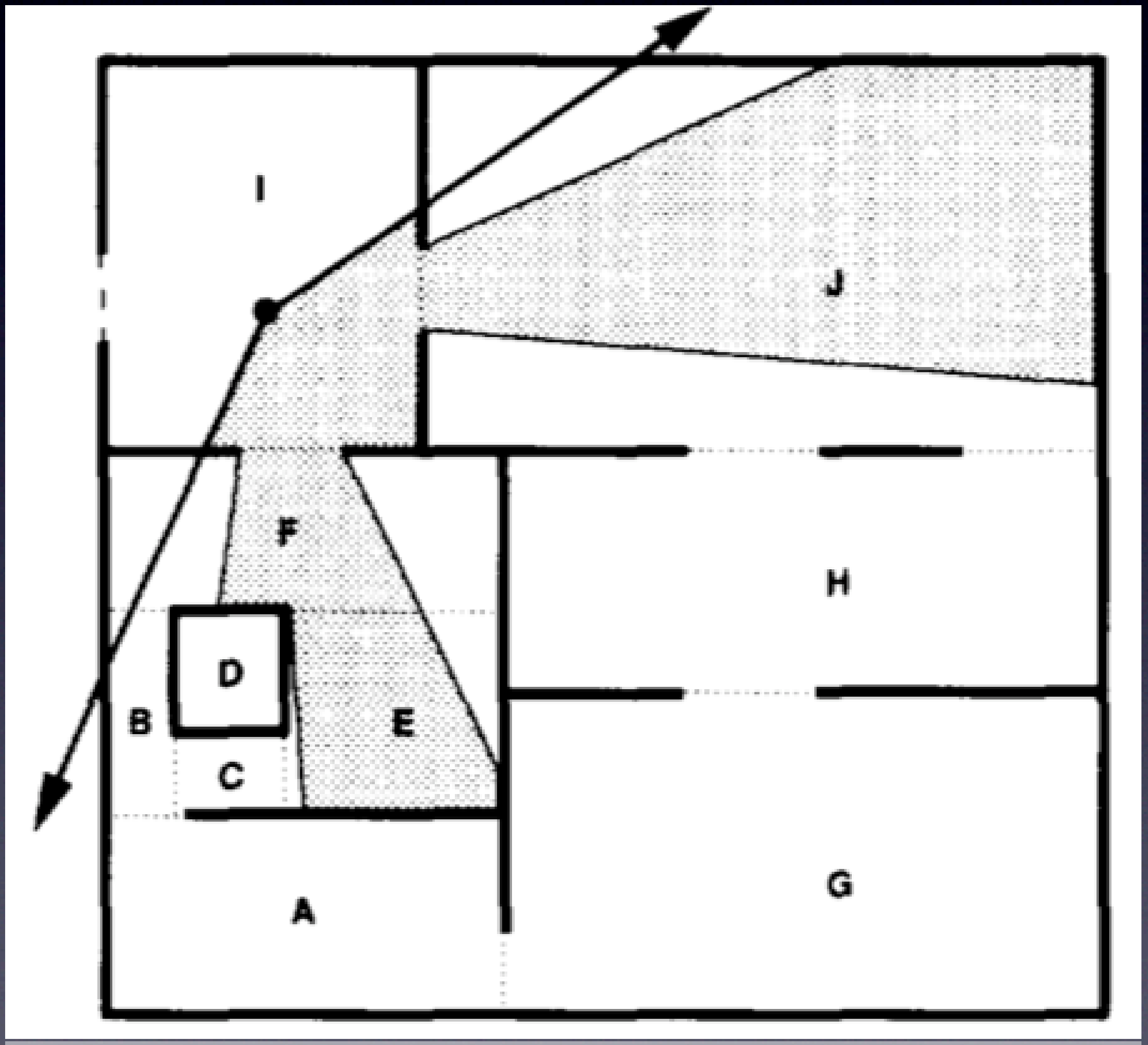
Eye to Cell Visibility

- Incident Portals
  - DFS on stab tree
  - Recurse down edge if portal intersects view frustum

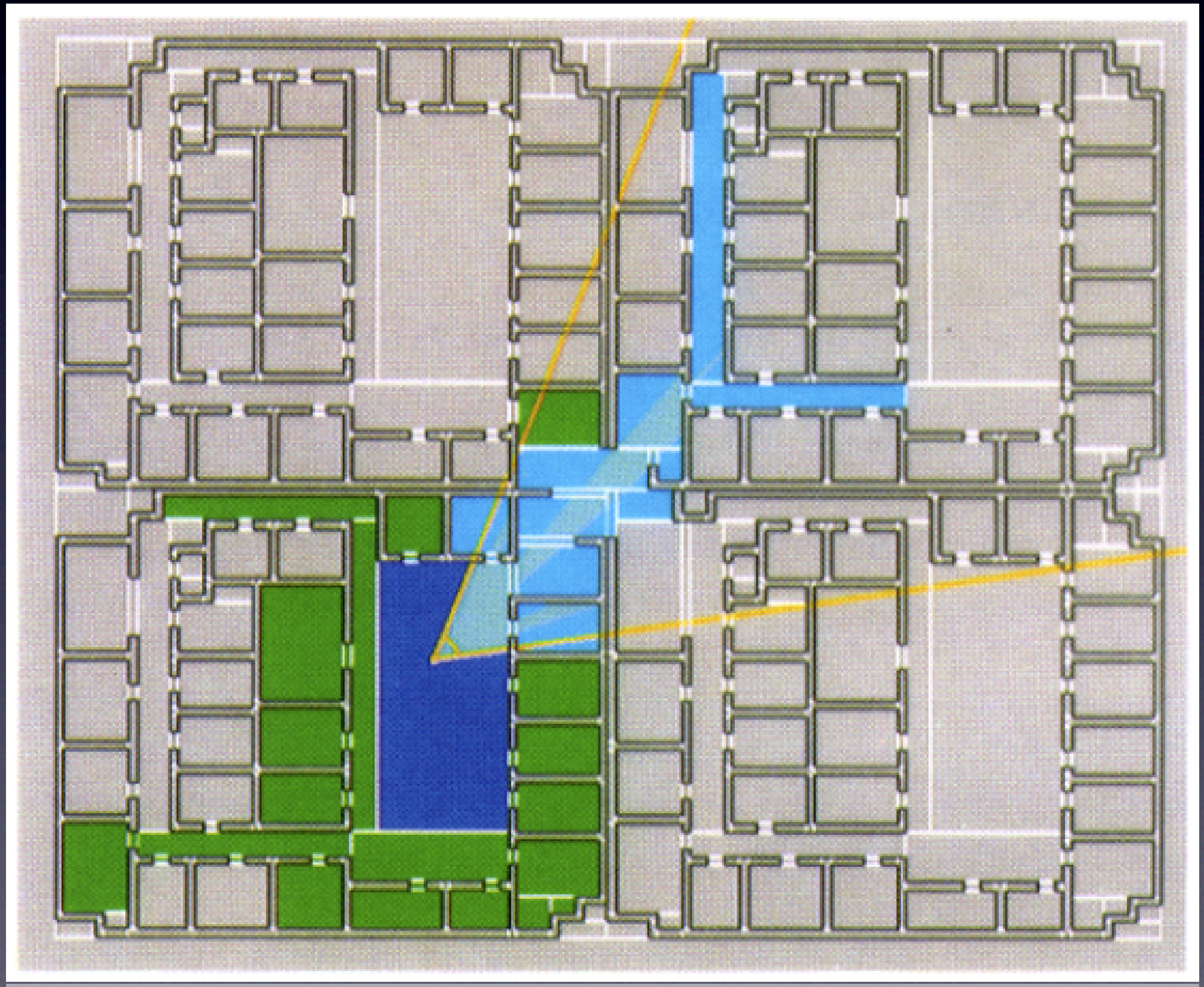


Eye to Cell Visibility

- Exact
  - DFS traversal of stab tree
  - For each additional portal check for a sightline that
    - passes through the portals
    - passes through the eye
    - lies in half-spaces defined by frustum



Results



Results

culling method	360° view cone		60° view cone	
	vis. area	*reduction factor*	*vis. area*	*reduction factor*
none (cell-to-cell vis.)	8.1%	10x	8.1%	10x
disjoint cell	8.1%	10x	3.1%	30x
connected component	8.1%	10x	2.4%	40x
incident portals	8.1%	10x	2.2%	40x
exact eye-to-cell	4.9%	20x	1.8%	50x
exact visible area	2.1%	50x	0.3%	300x

Extension to 3D

- Still assume axial faces
- Portals can be non-convex
  - Bounding box - looser approximation of VS
  - Decompose - possible combinatorial explosion

Extension to 3D

- Sightlines
  - Stab sequence of n axis-aligned quads
  - Another paper describes how to do this in O(n lg n)

Discussion

- Does solve their problem, and easily in 2D.
- Spatial subdivision relies on axial faces, but rest of the algorithms don't.
- Not general.
- No dynamic scenes.

Discussion

- Efficiency very data dependent.
  - Not thoroughly tested.
- Long precomputation, potentially significant storage.
- Inefficient - no way to reuse paths
- What about other types of portals? E.g. mirrors?

Discussion

- Front-to-back BSP drawing developed soon after
- Cell to cell visibility used in Quake
  - Stab tree storage very expensive
  - Visibility set storage expensive, requires compression
  - Simple bounding box cell-frustum culling
  - Expensive precomputation for 3D, even 5 years later

Questions?