IMGD 3000 - Technical Game Development I:
Scene Management

by

Robert W. Lindeman

gogo@wpi.edu

Overview

- Graphics cards can render a lot, very fast
  - But never as much, or as fast as we'd like!

- Intelligent scene management allows us to squeeze more out of our limited resources
  - Scene graphs
  - Scene partitioning
  - Visibility calculations
  - Level of detail control

Scene Graphs

- A specification of object and attribute relationships
  - Spatial
  - Hierarchical
  - Material properties
- Transformations
- Geometry
- Easy to attach objects together
  - Riding a vehicle

Scene Graphs (cont.)

- Can use instances to save resources
  - Geometry handles instead of geometry
  - Texture handles

- To take advantage of GPUs, reducing the amount of shader (cg) and texture switching is preferred

Geometry Sorting and Culling

- Keys to scene management
  - Render only what can be seen
  - Render at a satisfactory, perceivable fidelity
  - Pre-process what you can
  - Use GPU as efficiently as you can

- First-level
  - View-frustum culling
  - Back-face culling
  - Bounding sphere

- One or more *acceleration structures* can be used

Acceleration Structures

- Hierarchical bounding structures
  - Test if parent is visible
    - If not, then none of its children are
    - If so, then recursively check the children

- Could use information about your application to optimize approach
  - Many interior levels have cells and portals
  - No need to solve the general problem, just the specific one

Acceleration Structures

- [ ] **Many structures exist**
  - Appropriateness depends on the scene, and the game (e.g., dynamic objects)

- [ ] **Space partitioning**
  - Uniform Grid
  - Quad/Oct Tree
  - Binary-Space Partitioning (BSP) trees
  - k-d trees

- [ ] **Geometry partitioning**
  - Bounding boxes/spheres/capsules

Acceleration Structures - Space Partitioning

- Uniform Grids
  - Split space up into equal sized (or an equal number of) cells

- Quad (Oct) Trees
  - Recursively split space into 4 (8) equal-sized regions

- Binary-Space Partitioning (BSP) trees
  - Recursively divide space along a single, arbitrary plane

- *k*-dimensional trees (k-d trees)
  - Recursively

Acceleration Structures - Object Partitioning

- Bounding boxes/spheres/capsules
- Axis-Aligned Bounding Boxes (AABB)
- Oriented Bounding Boxes (OBB)
- Discrete Oriented Polytope (DOP)
  - Polytope: 2D = polygon, 3D = polyhedron
  - k-DOP: k planes in a DOP
  - Common: 6-DOP (AABB), 10-DOP, 18-DOP, 24-DOP
- Bounding-Volume Hierarchies (BVHs)

Cell-Portal Visibility

☐ Keep track of which cell the viewer is in

☐ Somehow enumerate all the visible regions

☐ Cell-based
  - Preprocess to identify the potentially visible set (PVS) for each cell

☐ Point-based
  - Compute at runtime

☐ Trend is toward point-based, but cell-based is still very common
 - Why choose one over the other?

Visibility of Cells

- Point-based algorithms compute visibility from a specific point
  - Which point?
  - How often must you compute visibility?

- Cell-based algorithms compute visibility from an entire cell
  - Union of the stuff visible from each point in the cell
  - How often must you compute visibility?

- Which method has a smaller potentially visible set?

- Which method is suitable for pre-computation?

Potentially Visible Set (PVS)

☐ PVS: The set of cells/regions/objects/polygons that can be seen from a particular cell
  - Generally, choose to identify objects that can be seen
  - Trade-off is memory consumption vs. accurate visibility

☐ Computed as a pre-process
  - Have to have a strategy to manage dynamic objects

☐ Used in various ways:
  - As the only visibility computation - render everything in the PVS for the viewer's current cell
  - As a first step - identify regions that are of interest for more accurate run-time algorithms
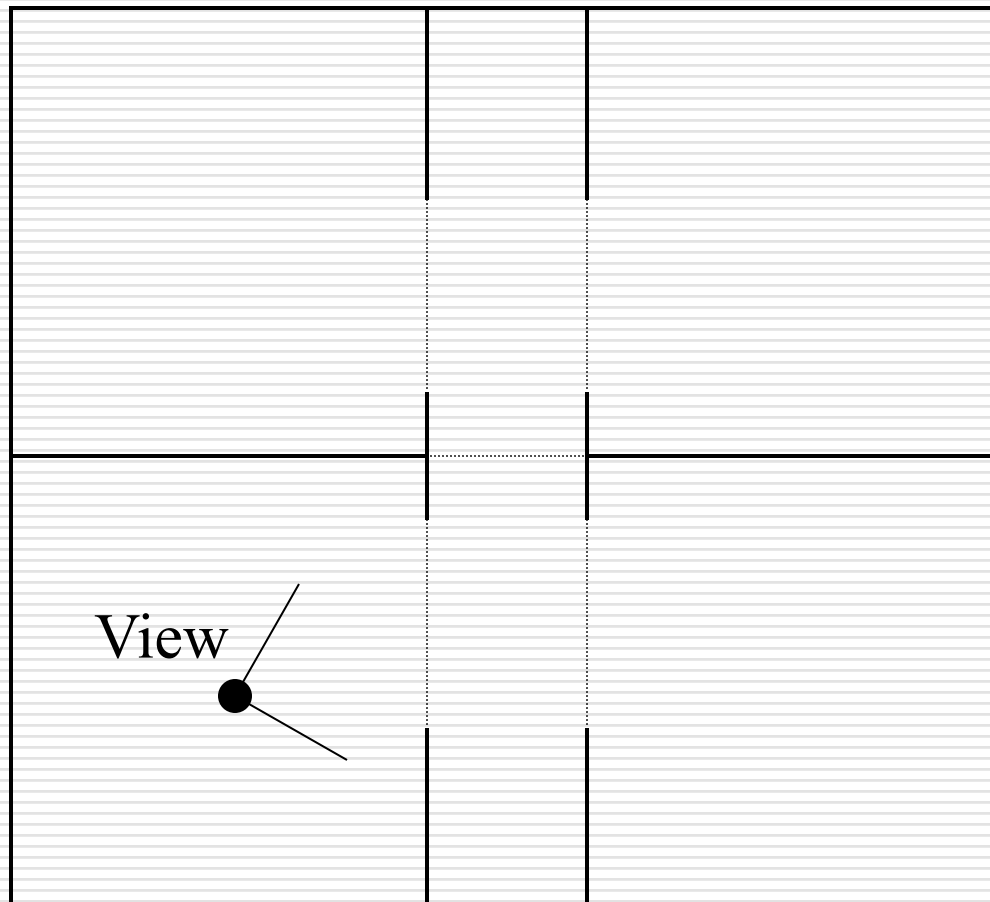
Cell-to-Cell PVS

☐ Cell A is in cell B's PVS if there exists a *stabbing line* from a portal of B to a portal of A

- ■ S*tabbing line:* a line segment intersecting only portals
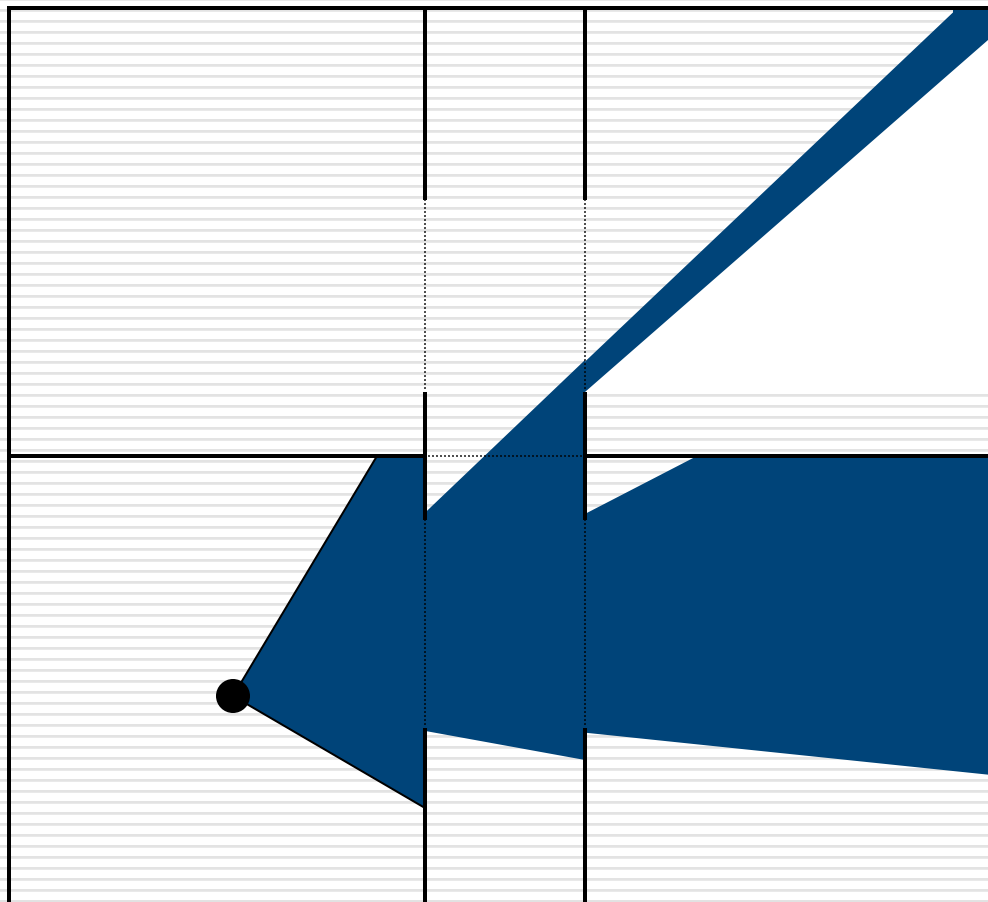- ■ Neighbor cells are trivially in the PVS
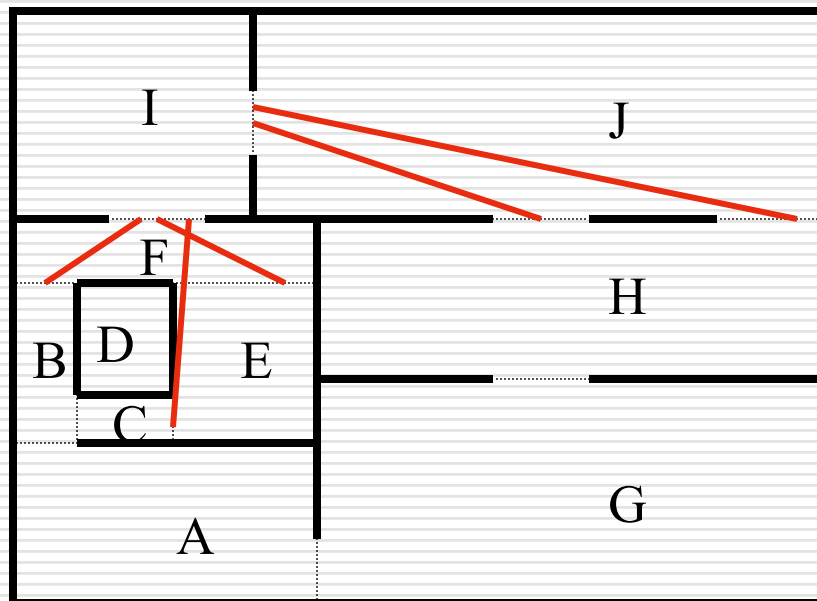
PVS for I contains:
B, C, E, F, H, J

Eye-to-Region Example (1)

View

Eye-to-Region Example (2)

Putting it all Together

- The "best" solution will be a combination
  - Static things
    - Oct-tree for terrain
    - Cells and portals for interior structures
  - Dynamic things
    - Quick reject using bounding spheres
    - BVHs for objects

- Balance between pre-computation and run-time computation

References

- http://www.cs.wisc.edu/graphics/Courses/679-f2003/