Video Games Technologies

11498: MSc in Computer Science and Engineering
11156: MSc in Game Design and Development

Chap. 7 — Culling

Outline

…:

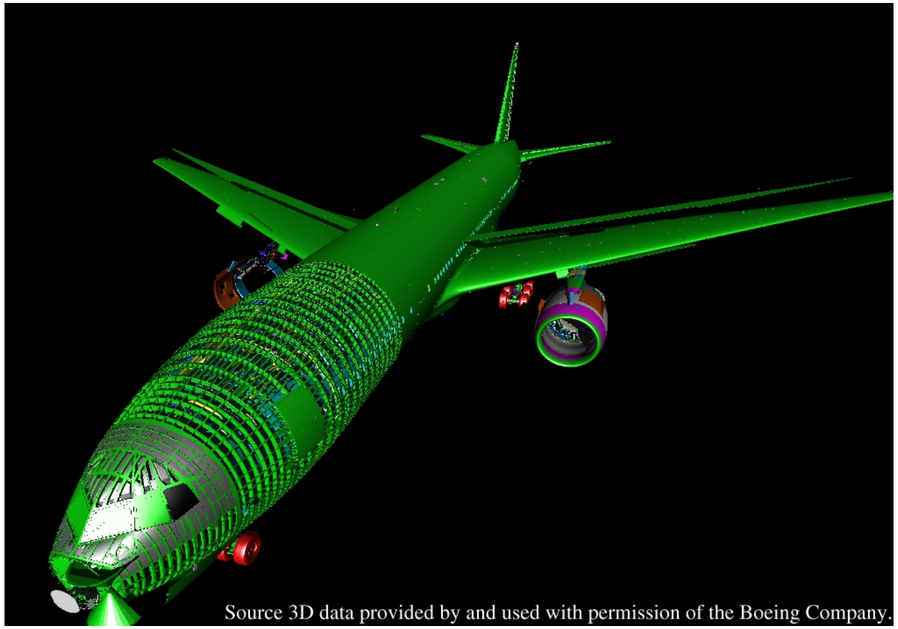
- Introduction: massive models.
- Motivation. Culling definition & types.
- Back-face culling.
- View frustum culling.
- Speeding-up techniques.
- Computation of bounding volumes.
- Occlusion culling.
- Portal culling.

Procedurally generated model of Pompeii: ~1.4 billion polygons.
Image from [Mueller06]

Massive models: trends & goal

Trends

- No upper bound on model complexity
  - Procedural generation
  - Laser scans
  - Aerial imagery
- High GPU throughput
  - At least 10-200 million triangles/second
- Widen gap between processor and memory performance
- CPU – GPU bottleneck



Boeing 777 model: ~350 million polygons.
Image from http://graphics.cs.uni-sb.de/MassiveRT/boeing777.html

Goal:

- *Output-sensitivity*: Performance as a function of the number of pixels rendered, not the size of the model

Culling: Motivation

*As any other rendering acceleration technique, the **goal** is to avoid rendering redundant geometry*

The leading idea: don't render what can't be seen

- Off-screen: *view-frustum culling*
- Occluded by other objects: *occlusion culling*

The obvious question: *why bother? (When the graphics system already gives this for granted?)*

- Off-screen geometry: *solved by clipping*
- Occluded geometry: *solved by Z-buffer*

The (obvious) answer: *efficiency*

- In fact, clipping and Z-buffering are of linear time complexity, i.e., take time linear to the number of primitives
- So, let us see a number of techniques to speed up rendering.

Culling: definition & types

*As any other rendering acceleration technique, the **goal** is to avoid rendering redundant geometry*

Definition:

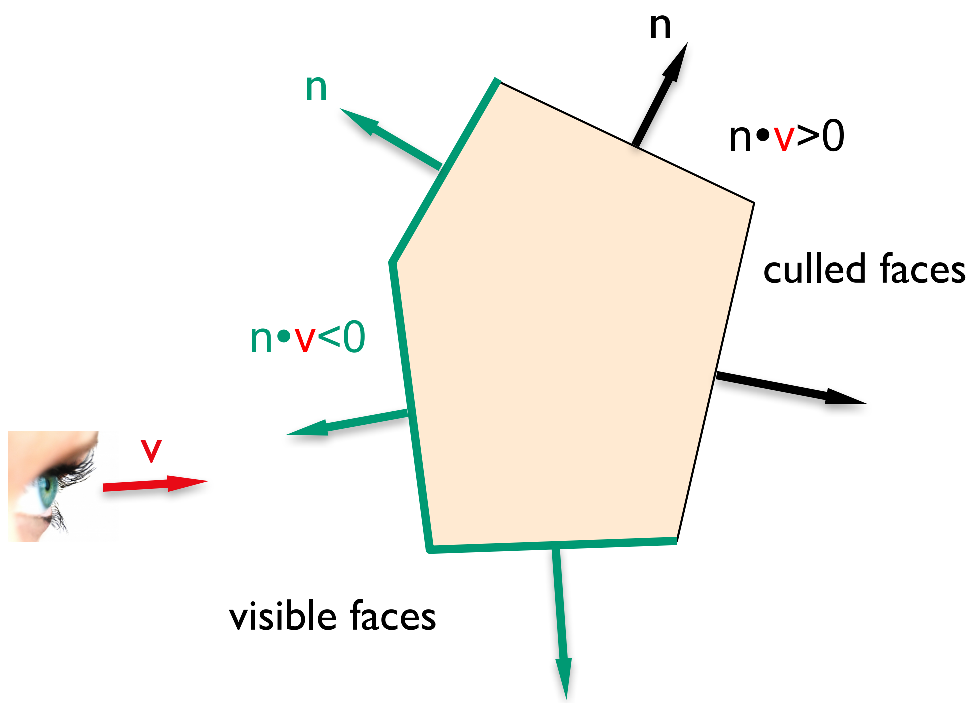
- The term 'culling' means removing from a group.
- In graphics, it means to determine which objects in the scene are *not* visible
- It is more productive to think about it as determining which objects *are* visible

Types of culling:

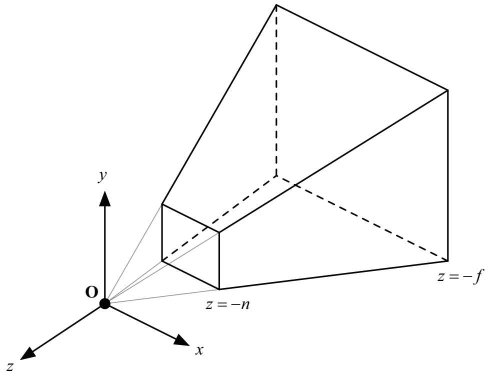
- <u>View frustrum culling</u> (in object space)
- <u>Visibility culling</u> (in object space)
  - Backface culling
  - Portal culling
  - Textured-depth mesh culling
- <u>Occlusion culling</u> (in image space)
  - Hierarchical occlusion maps (HOM)
  - Hierarchical Z-buffer
  - Occlusion planes

Back-face culling

- Backface culling is not really part of scene management – it is a lower level feature usually built into the rendering layer.
- We do not draw polygons facing the other direction
- Test z component of surface normals. If negative – cull, since normal points away from viewer.
- Or, if n•v > 0, we are not viewing the back face so polygon is obscured.

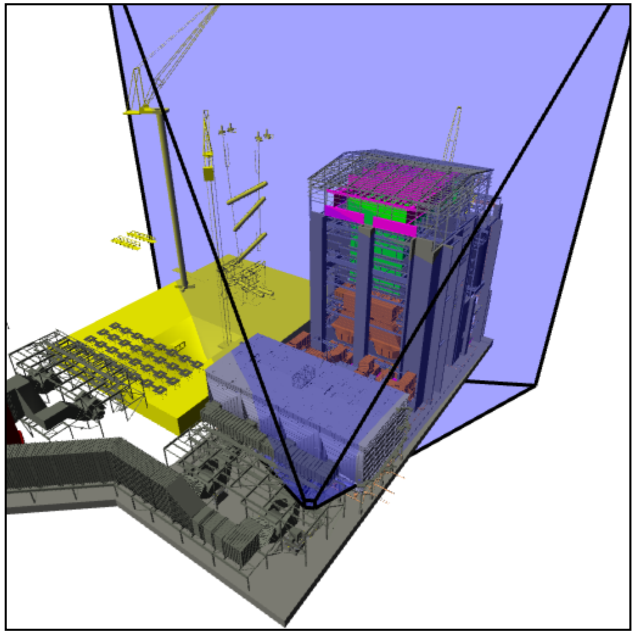


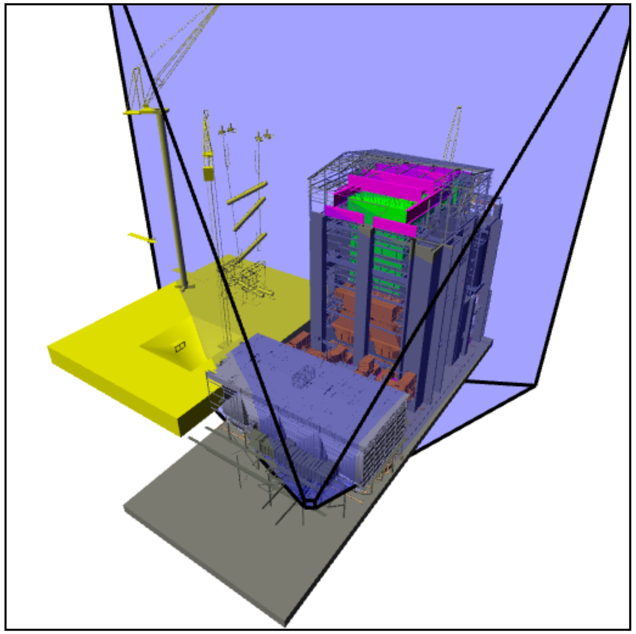
View frustum culling



- *View frustum* (view volume) is defined by six planes, namely the front, back, left, right, top, and bottom clipping planes, which together form a cut pyramid.
- *Total visibility*: If an object is entirely inside the current view frustum are drawn.
- *Partial visibility*: If it is partially inside, it is clipped to the planes, clipping its outside parts.
- *Total invisibility*: If an object is entirely outside the pyramid, it is not visible and is thus discarded.

Full model
1.7 Mtris



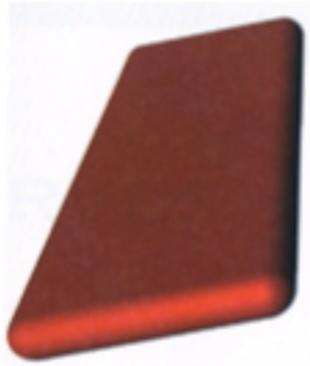


View frustum culling
1.4 Mtris

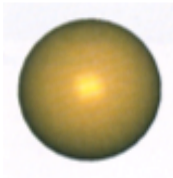
source: https://www.cs.unc.edu/~dm/UNC/COMP236/LECTURES/Lecture03.ppt

Speeding up the view frustum culling: using bounding volumes

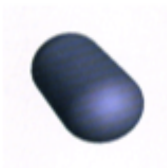
- ***Bounding volume***: Every object in the world should be enclosed in a bounding volume in order to speed up the view frustum culling. There are 3 possible results:
  - Totally visible
  - Totally invisible
  - Partially visible (may require clipping)
- ***Bounding volume types***:
  - Sphere, cylinder, hot dog / capsule / losenge
  - AABB: axis-aligned bounding box
  - OBB: oriented bounding box
  - Convex polyhedron



lozenge



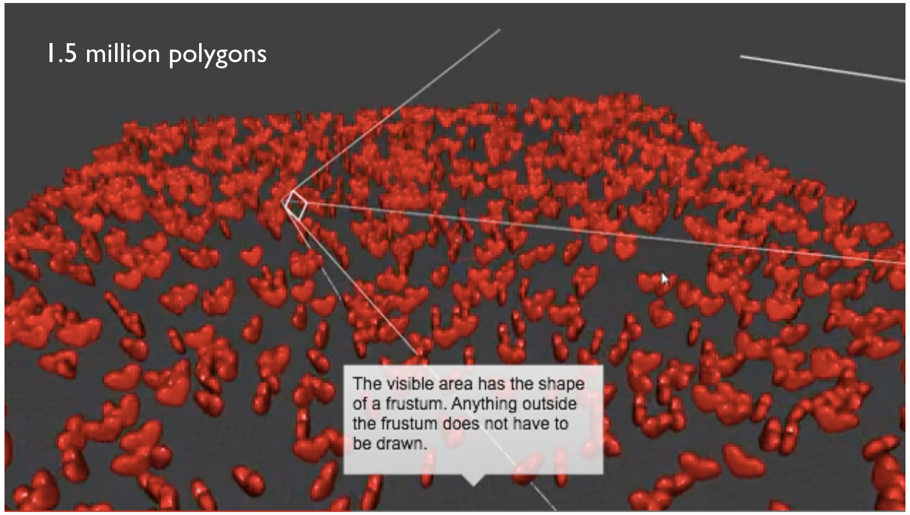
sphere

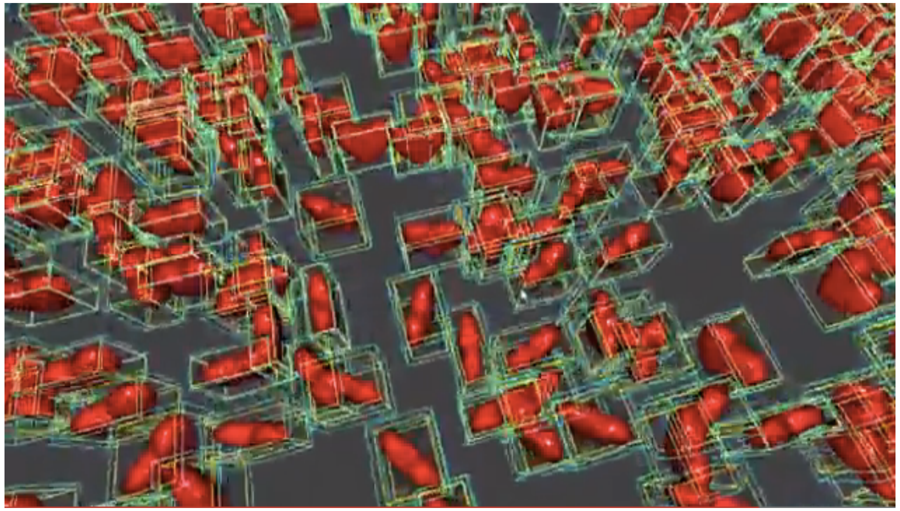


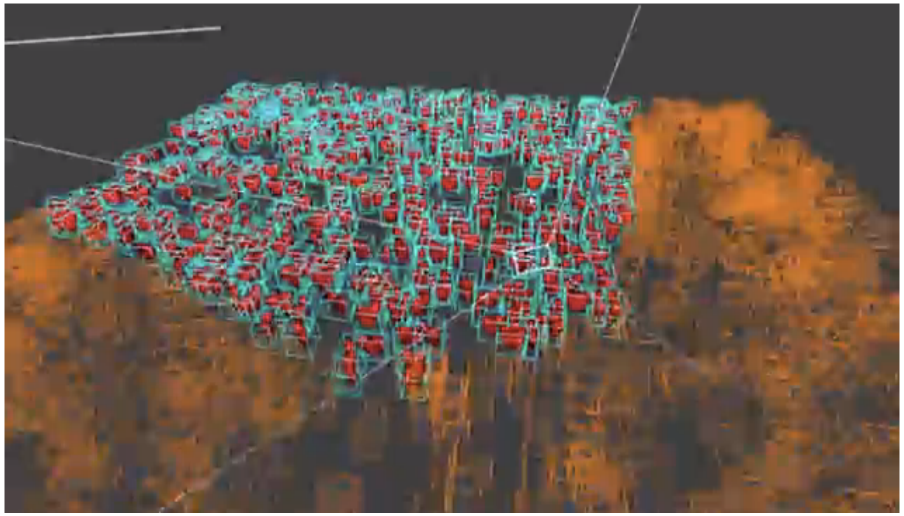
capsule

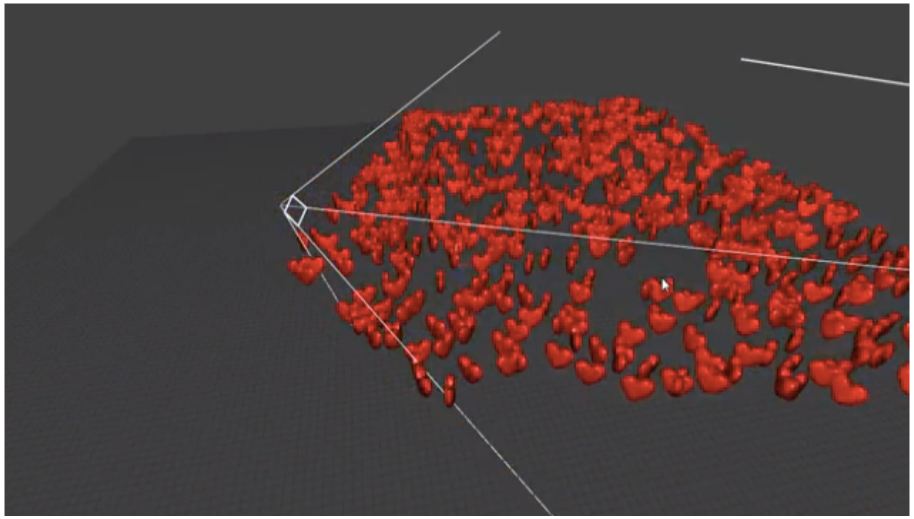
Speeding up the view frustum culling: using bounding volumes

http://youtu.be/fNa_Gh5gFWY









Computation of bounding volumes

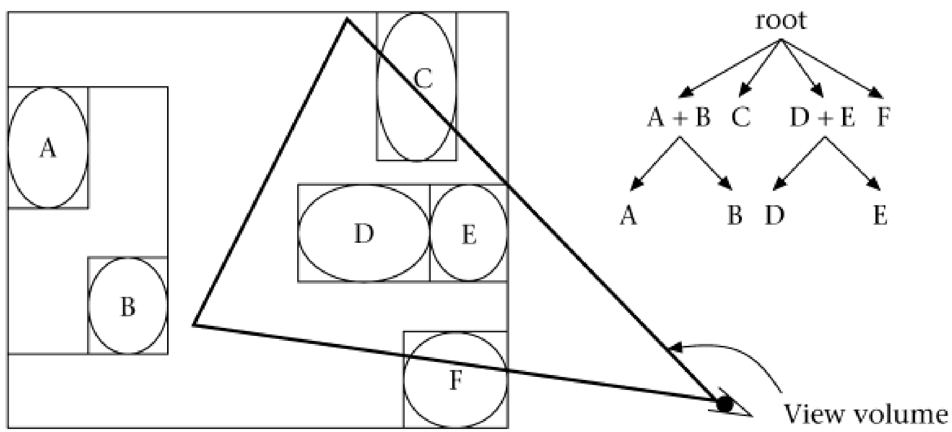
- Spheres?

  - *Step 1*: Compute average point of all vertices as the center of the bounding sphere.
  - *Step 2*: Find the farthest vertex from the center, setting the radius as the distance between them.

- Bounding boxes?
- etc.?

Speeding up further the view frustum culling: using a bounding volume hierarchy

- Compare the scene hierarchically against the view frustum
- When a bounding volume is found to be outside the view frustum then all objects inside it can be safely discarded
- If a bounding volume is fully inside then render without clipping
- What is the difference with clipping?



Portal culling

Culling service:

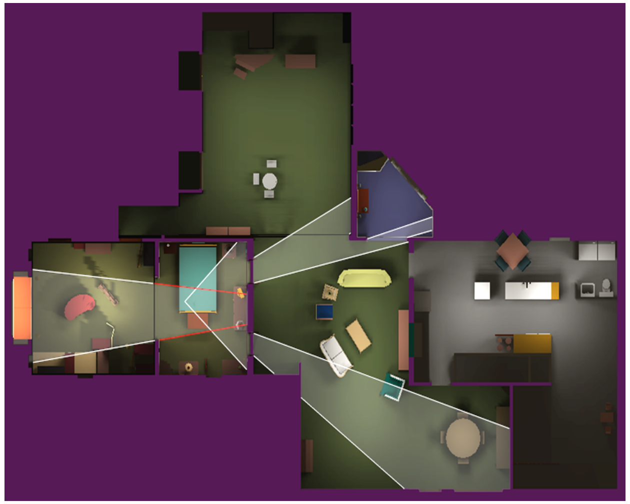
- Construct BSP tree, removing then areas of the world that cannot be seen.
- Can work in conjunction with BVH and other culling algorithms.

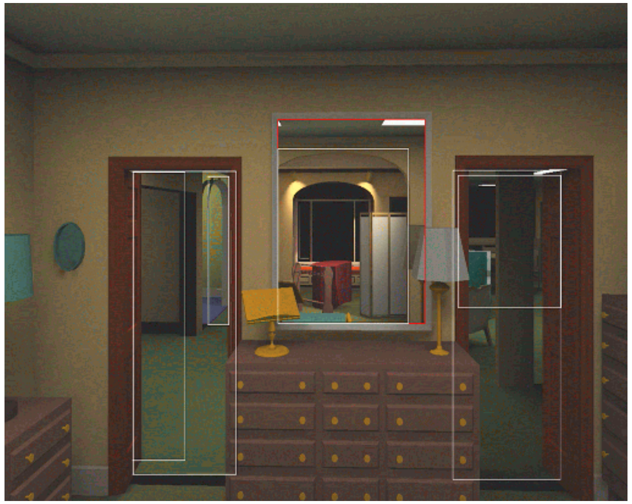
Application/Usefulness:

- To handle in-door scenes (e.g., buildings), but also applies to giant scenes like cities.

Scene modeling:

- Scene is divided into **cells** (rooms, corridors, etc.)
- Transparent **portals** connect cells (doorways, entrances, windows, etc.)
- Cells only see other cells through portals





http://www.cs.virginia.edu/~gfx/Courses/2002/RealTime.fall.02/

- *Average: culled 20-50% of the polys in view*
- *Speedup: from slightly better to 10 times*

Portal culling (contd.)

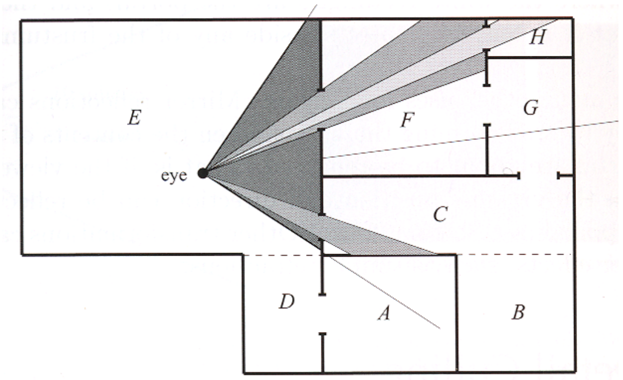
Data structure: scene as a adjacency scene graph

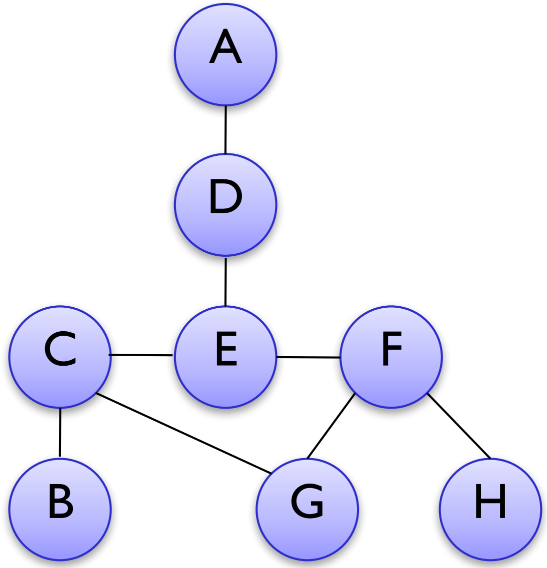
- ***Nodes***: Cells. The scene is divided into cells that usually correspond to *rooms* and *hallways* in a building connected by rectangular *portals*.
- ***Edges***: Portals. *Portals* are doors and windows that connect adjacent rooms (and/or hallways).
- Every object in a cell and the walls of the cell are stored in a data structure that is associated to the cell.
- ***Adjacency graph***: potentially visible set (PVS)

Goal: revisited

- *Quickly* eliminate large portions of the scene which will not be visible in the final image
- Not exact visibility solution, but a quick-and-dirty *conservative* estimate of which primitives are visible;
- Z-buffer& clip this for the exact solution

Cells enumerated from A to H, and portals are openings that connect the cells. Only geometry seen through the portals is rendered.





Portal culling (contd.)

The leading idea:

- Cells form the basic unit of PVS
- Create an *adjacency graph* of cells
- Starting with cell containing eyepoint, traverse graph, rendering visible cells
- A cell is only visible if it can be seen through a sequence of portals along *a line of sight*

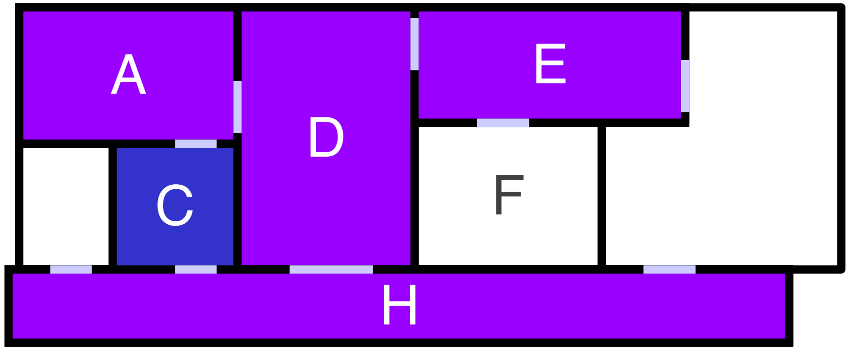
Questions:

- How is a given cell visible from a given <u>viewpoint</u>?
- How can we detect <u>view-independent</u> visibility between cells?

Solutions?:

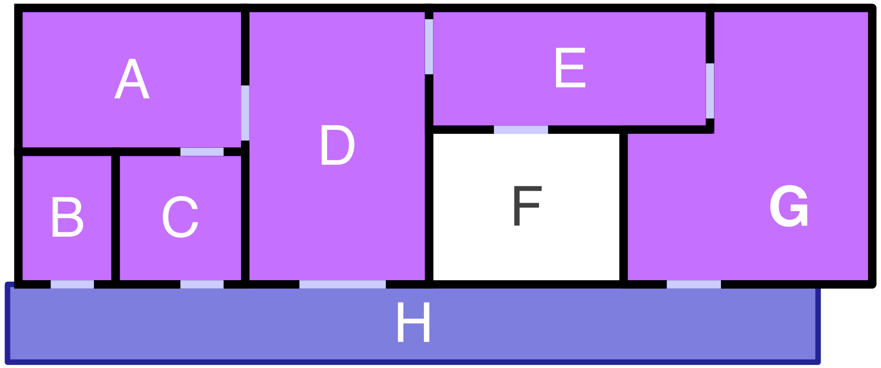
- These problems reduce to **eye-portal** and **portal-portal** visibility

view-independent visibility: example I



C can *only* see A, D, E, and H

view-independent visibility: example II



H will *never* see F

Portal culling: research contributions

http://pfportals.cs.virginia.edu/pfPortals.html

Airey (1990): view-independent only

- Portal-portal visibility determined by ray-casting
  - Non-conservative portal-portal test resulted in occasional errors in PVS
- Slow preprocess
- Order-of-magnitude speedups

Teller & Sequin (1991): view-independent + view-dependent

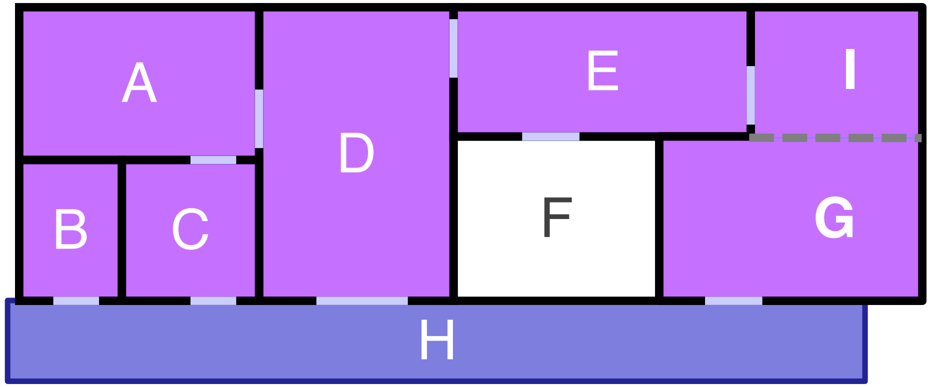
- Portal-portal visibility calculated by *line stabbing* using linear program
  - Cell-cell visibility stored in stab trees
  - View-dependent eye-portal visibility stage further refines PVS at run time
- Slow preprocess
- Elegant, exact scheme

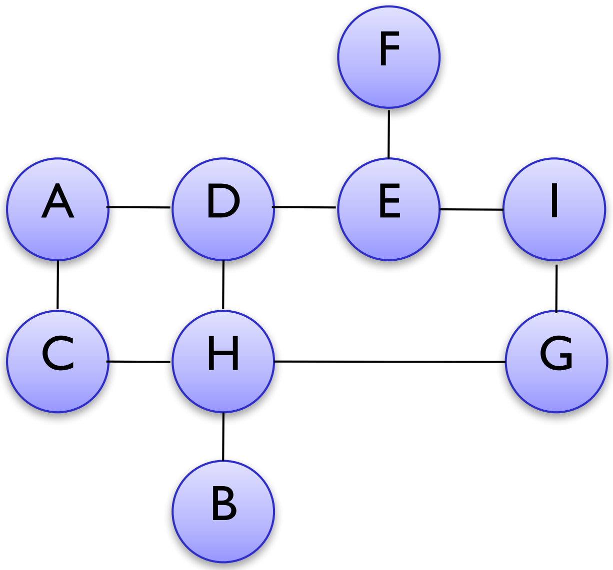
Luebke & Georges (1995): view-dependent only

- Eye-portal visibility determined by intersecting cull boxes
- No preprocess (integrate w/ modeling)
- Quick, simple hack
- Public-domain library: *pfPortals*

Cells and Portals:
Teller & Sequin algorithm (SIG'91)

- Cells form the basic unit of PVS
- Decompose space into convex cells
- For each cell, identify its boundary edges into two sets: opaque or portal
- *Pre-compute visibility among cells*
- *During viewing (eg, walkthrough phase), use the pre-computed potentially visible polygon set (PVS) of each cell to speed-up rendering*

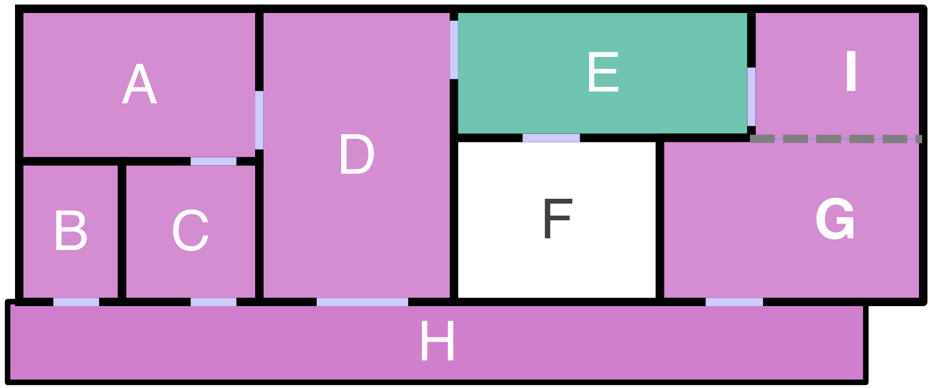


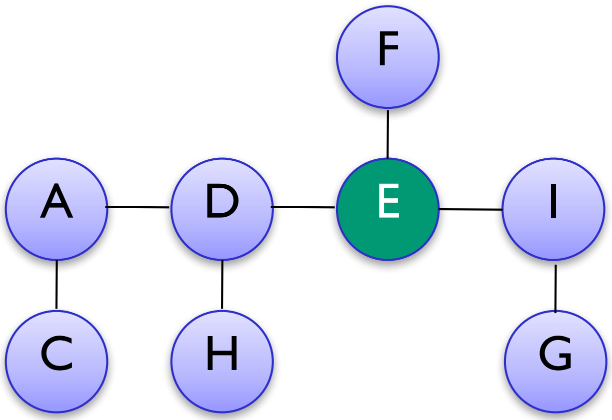


Teller & Sequin algorithm (SIG'91)
For each cell find stabbing tree

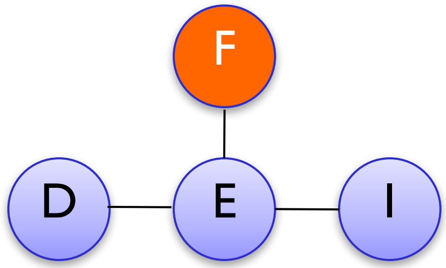
The stabbing tree of each cell is found using:

- Adjacency relationships of its neighbor cells through portals
- Sightlines (or stab lines) according to the criterion on next transparency





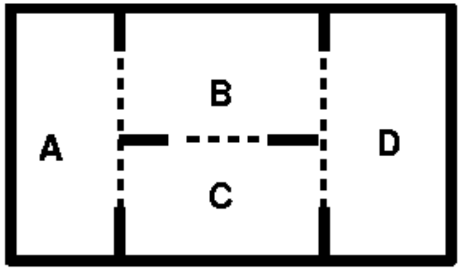
stabbing tree of cell E

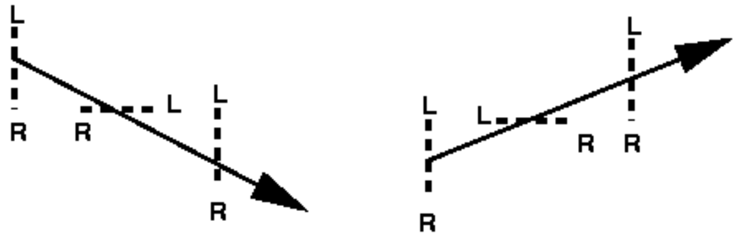


stabbing tree of cell F

Teller & Sequin algorithm (SIG'91)
Compute cells visible from each cell

http://dl.acm.org/citation.cfm?doid=122718.122725





$$S \bullet L \geq 0, \qquad \forall L \in \mathbf{L}$$

$$S \bullet R \leq 0, \qquad \forall R \in \mathbf{R}$$

```
Find_Visible_Cells(cell C, portal sequence P, visible cell set V)
    V=V ∪ C
    for each neighbor N of C
        for each portal p connecting C and N
            orient p from C to N
            P' = P concatenate p
            if Stabbing_Line(P') exists then
                Find_Visible_Cells (N, P', V)
```

Labeling L and R

A sightline can stab a portal sequence if and only if the point sets L and R are linearly separable;

Teller & Sequin algorithm (SIG'91)
Eye-to-cell visibility

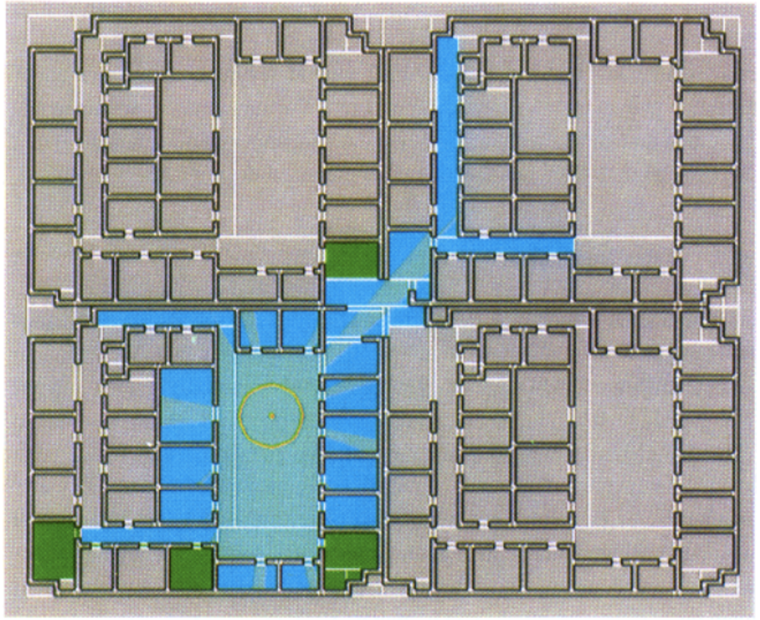
The eye-to-cell visibility of any observer is a subset of the cell-to-cell visibility for the cell containing the observer.

– This is so because the field of view of the cell-to-cell visibility procedure is implicitly 360°, while that one of the observer is less than 180°.

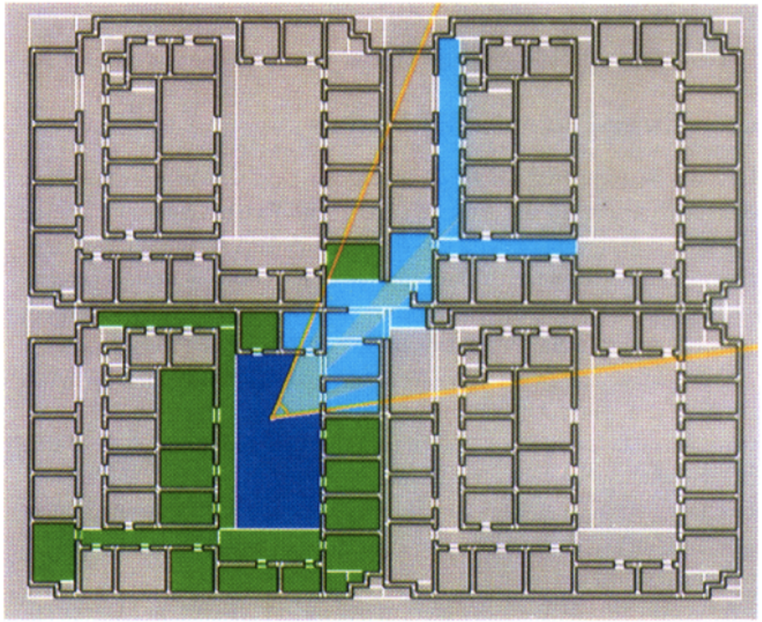
A cell is visible if all of the following are true:

– cell is in view volume (VV)
– all cells along stab tree are in VV
– all portals along stab tree are in VV
– sightline within VV exists through portals.

An observer with a 360° view cone



An observer with a 60° view cone

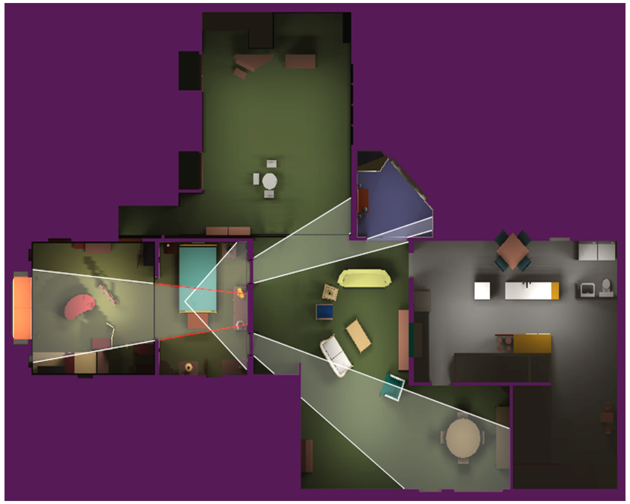


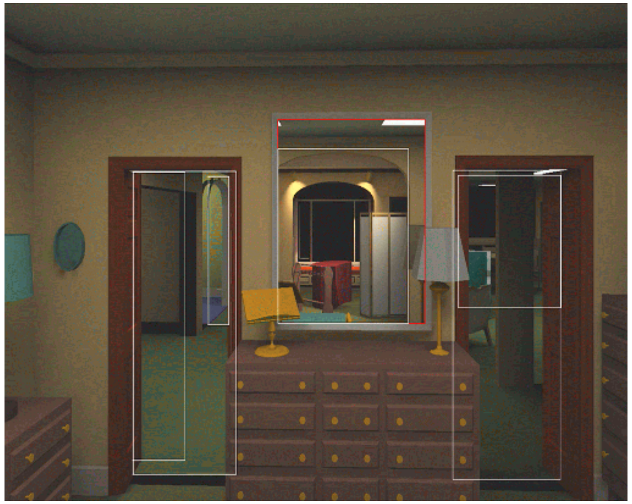
http://www.cs.tau.ac.il/~dcor/online_papers/papers/visibility-survey-ieee.pdf

Luebke & Georges algorithm (I3D'95)
Image space cells and portals

- Instead of pre-processing all the PVS calculation, it is possible to use image-space portals to make the computation easier
- ***pfPortals algorithm***: Depth-first adjacency graph traversal:
  - Render cell containing viewer
  - Treat portals as special polygons
    - If portal is visible, render adjacent cell
    - But clip to boundaries of portal!
    - Recursively check portals in that cell against new clip boundaries (and render)
  - Each visible *portal sequence* amounts to a series of nested portal boundaries
    - Kept implicitly on recursion stack

Top View Showing the Recursive Clipping of the View Volume





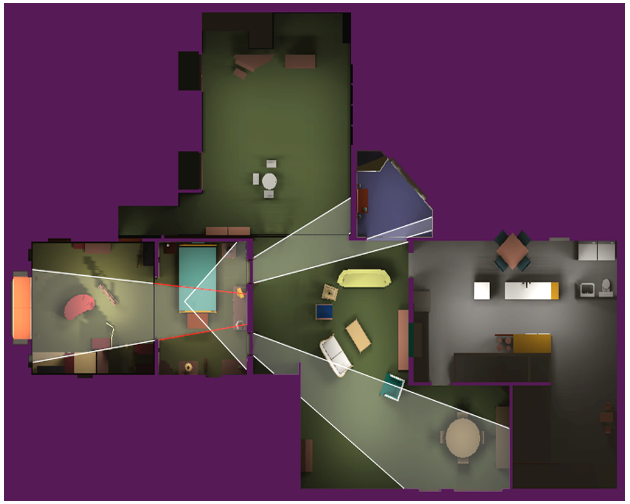
- *Average: culled 20-50% of the polys in view*
- *Speedup: from slightly better to 10 times*

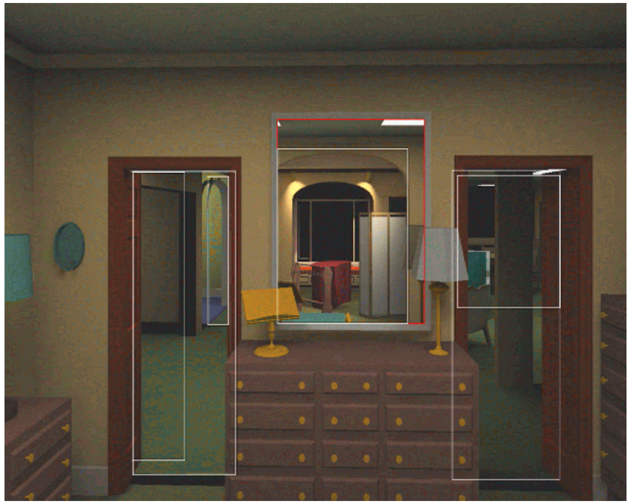
http://pfportals.cs.virginia.edu/

Portal issues

- Imposters
- Portal clipping
- Camera location
- Combining with bounding volume culling
- Moving objects
- Dynamic portals (opening & closing doors)
- Procedurally generating portals

Top View Showing the Recursive Clipping of the View Volume





- *Average: culled 20-50% of the polys in view*
- *Speedup: from slightly better to 10 times*

Further reading

http://comp.utm.my/pars/files/2013/04/Bounding-Volume-Hierarchy-for-Avatar-Collision-Detection-Design-Considerations.pdf

http://archive.gamedev.net/archive/reference/articles/article1212.html

http://www.cse.chalmers.se/~uffe/vfc_bbox.pdf

https://web.fe.up.pt/~aas/pub/Aulas/RVA/AcelRendering.pdf

http://cg.ivd.kit.edu/publications/2012/RBVH/RBVH.pdf

Summary:

…:

- Introduction: massive models.
- Motivation. Culling definition & types.
- Back-face culling.
- View frustum culling.
- Speeding-up techniques.
- Computation of bounding volumes.
- Occlusion culling.
- Portal culling.