Using Graphs: Visibility Culling

(slides based on those of David Luebke)
Motivation

- When rendering/displaying objects on a computer screen, we do not need to draw objects that are not visible.
- Thus, visibility culling = cull away non-visible objects.
- The graphics-card (z-buffer) does this but *after* transformation; we want to do it *before* it even reaches the graphics card.
Visibility Culling vs. Level-of-detail Simplification

- Level-of-detail (LOD) simplification is related to visibility culling but is **not** the same thing
  - LOD “simplifies” the visible object(s)
  - Visibility culling does not change what is rendered/displayed – it just reduces the unnecessary work of rendering non-visible geometry
Types of visibility culling

- View frustum culling
- Back face culling
- Occlusion culling
Examples of visibility culling

(potentially) visible

not-visible
Examples of visibility culling
Visibility Culling

Let’s focus on architectural models...
Portal Culling

Goal: walk through architectural models (buildings, cities, catacombs)

These divide naturally into **cells**
- Rooms, alcoves, corridors...

Transparent **portals** connect cells
- Doorways, entrances, windows...

Notice: cells only see other cells through portals
Cells & Portals

An example:
Cells & Portals

Visibility Culling Idea:
- 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
  - So cell visibility reduces to testing portal sequences for a line of sight...
Cells & Portals
Cells & Portals

Visibility Culling
Cells & Portals

Visibility Culling
Cells & Portals

Visibility Culling
Cells & Portals

Visibility Culling
Cells & Portals
Cells & Portals
Cells & Portals

*View-independent* solution: find all cells a particular cell could *possibly* see; e.g. what can C see?

C can *only* see A, D, E, and H
Cells & Portals

*View-independent* solution: find all cells a particular cell could *possibly* see; e.g. what can H not see?

H will *never* see F
Cells and Portals

How can we detect the cells that are visible from any given viewpoint?

Idea:

- Set the view box \( P \) as the entire screen
- Compare the portal \( B \) to the neighbor cell \( C \) against the current view box \( P \)
  - If \( B \) outside \( P \) – the neighbor cell \( C \) cannot be seen
  - Otherwise – the neighbor cell \( C \) is visible
    - New view box \( P = \) intersection of \( P \) and the portal \( B \)
    - For each neighbor of \( C \), depth first traverse the adjacency graph of \( C \) and recurse
Example I

1. From eye, can see to C and to F
2. From C, can see nothing more
3. From F, can see H
4. From H, can see nothing more
Example II

A mirror – still works (cool!)

eye
Extension: Portal Images

How can you reduce the number of rendered cells to at most 2?

Answer: replace cells with images... 😊
Cells and Portal Culling
Portal Image Culling
Creating Portal Images

Ideal portal image would be one sampled from exactly the current eye position
Creating Portal Images

Option 1: a large number of static reference images (~120) needed to eliminate *popping*
Creating Portal Images

Option 2: use image warping and thus require a much smaller number of reference images
Example: Comparison

Color Figure A: Visibility Errors. This image shows a single reference image being warped (from a total of six sampled across the portal). The viewpoint is at the worst location for this reference image. Observe the black areas where we have no information.

Color Figure B: Two Reference Images. An image from the same viewpoint as A, but we are warping the two nearest reference images (from a total of six) to render the desired image. The large areas that were invisible from one are visible from the second.

Color Figure C: Layered Depth Image. The LDI for the portal captures almost all of the visible detail. In addition, it takes up less storage and can be rendered faster than two full reference images. Furthermore, in the architectural domain, we can construct high-quality LDIs by using reference images sampled along a semicircle in front of each portal.

Color Figure D: Geometry. An image from the same viewpoint as A, but rendered using the model geometry for purposes of comparison with figures A, B and C. The main difference is some detail through the left doorway. Apparently these were some features that were visible from only certain viewing angles.