The Visibility Problem in Walkthrough Applications

Daniel Cohen-Or and Shuly Lev-Yehudi

Computer Science Department

Tel-Aviv University

Virtual Reality Applications

- The user “walks” interactively in a virtual polygonal environment
- <u>The goal</u>: to render an updated image for each view point and for each view direction in interactive frame rate
- **Visibility Computation:** Selecting the set of polygons from the model which are visible from a fixed viewpoint or potentially visible within a region

Cells and Portals

based on:

Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets

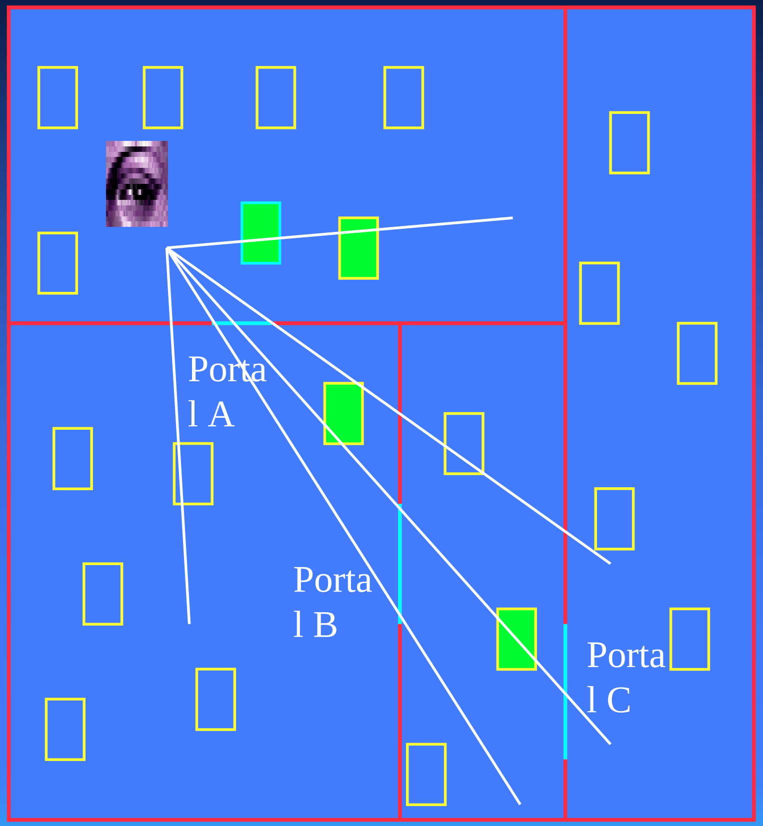
David P. Luebke and Chris Georges
University of North Carolina
1995

Cells and Portals

- ◆ Goal: Interactive walkthrough in architectural models (buildings, cities)
- ◆ These divide naturally into cells
  - - *rooms, alcoves, corridors*
- ◆ Transparent portals connect cells
  - - *windows, doors, entrances*

✷ Cells can only "see" other cells through the portals

Cells and Portals

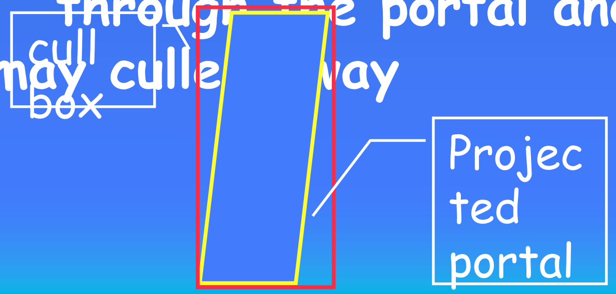


Cells and Portals - The Idea

- Build an adjacency graph of cells
- Starting with the cell containing viewpoint, traverse graph, rendering visible cells
  - A cell is only visible if it can be seen through a sequence of portals
  - need a line of sight
- So cell visibility reduces to testing portals sequences...

Cells and Portals - The Algorithm

- Project the vertices of each portal into screen-space and take 2D axial bounding box - called cull box
- Objects whose projection falls entirely outside of the cull box are not visible through the portal and may culled away



Cells and Portals - The Algorithm

- As each successive portal is traversed, its box is intersected with the aggregate cull box

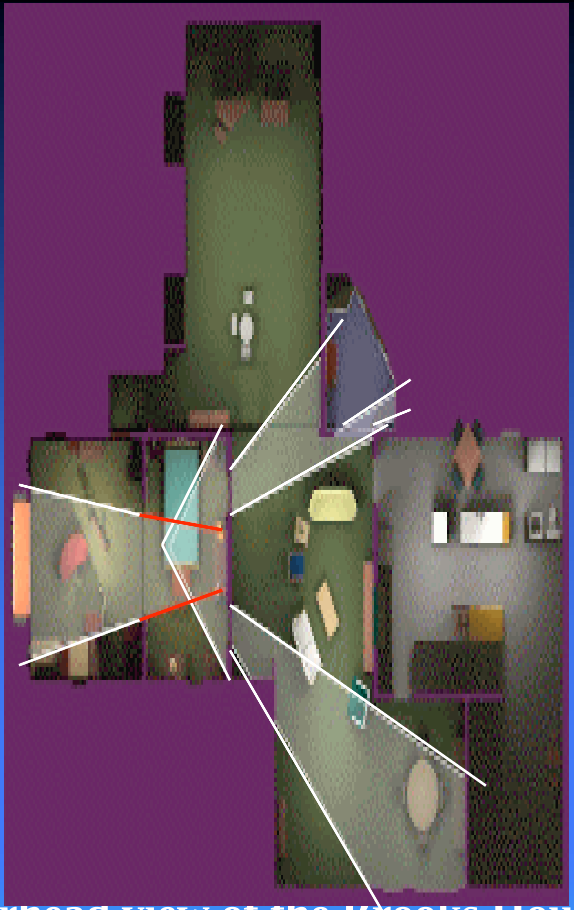
Aggregate cull box:

intersected cull box of all portals in the sequence

Cells and Portals - The Algorithm

View from the master bedroom of the Brooks House showing cull boxes for portals (white) and mirrors (red)

Cells and Portals - The Algorithm

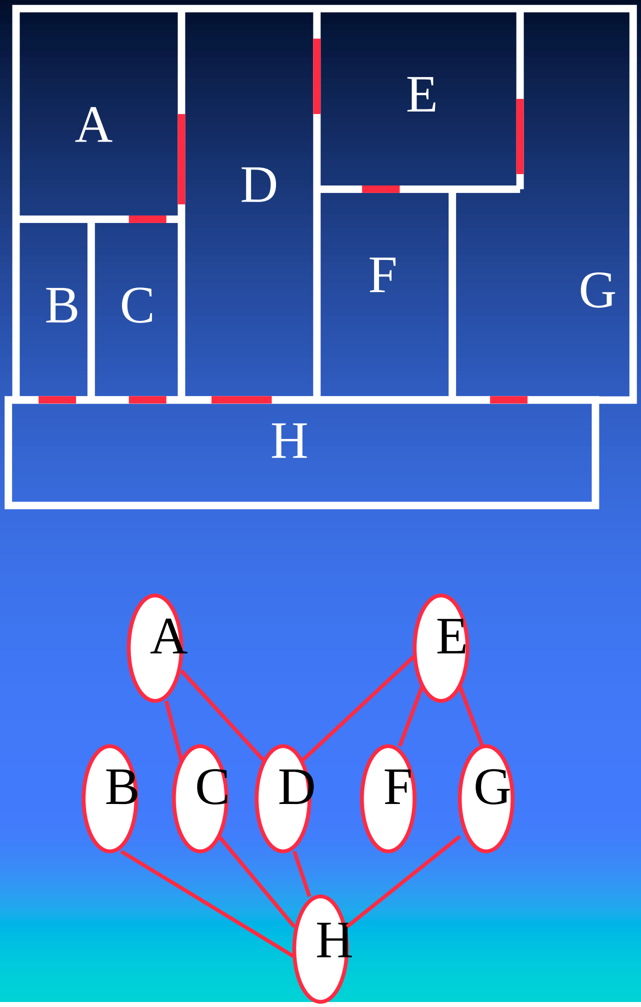


Overhead view of the Brooks House, showing portal culling frustums active in previous picture (mirror frustum shown in

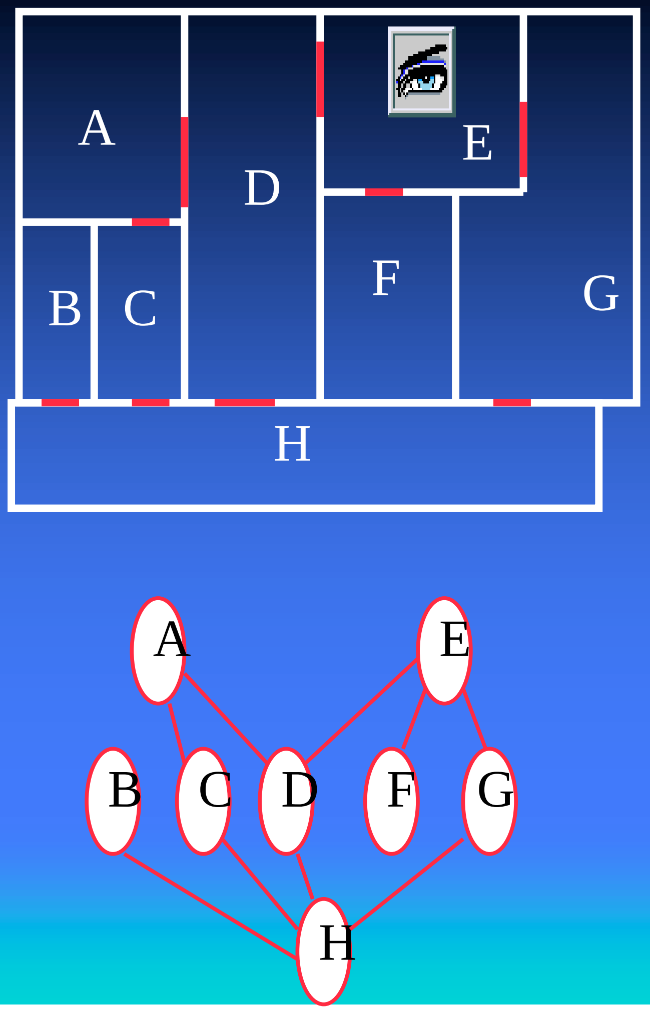
Cells and Portals - The Algorithm

- Cell's visibility: if the projected bounding box of an object within the cell intersects the aggregate cull box, the object is potentially visible
- Since a single object may be visible through multiple portal sequences, each object is tagged as it is rendered. (to avoid rendering objects more than once per frame)

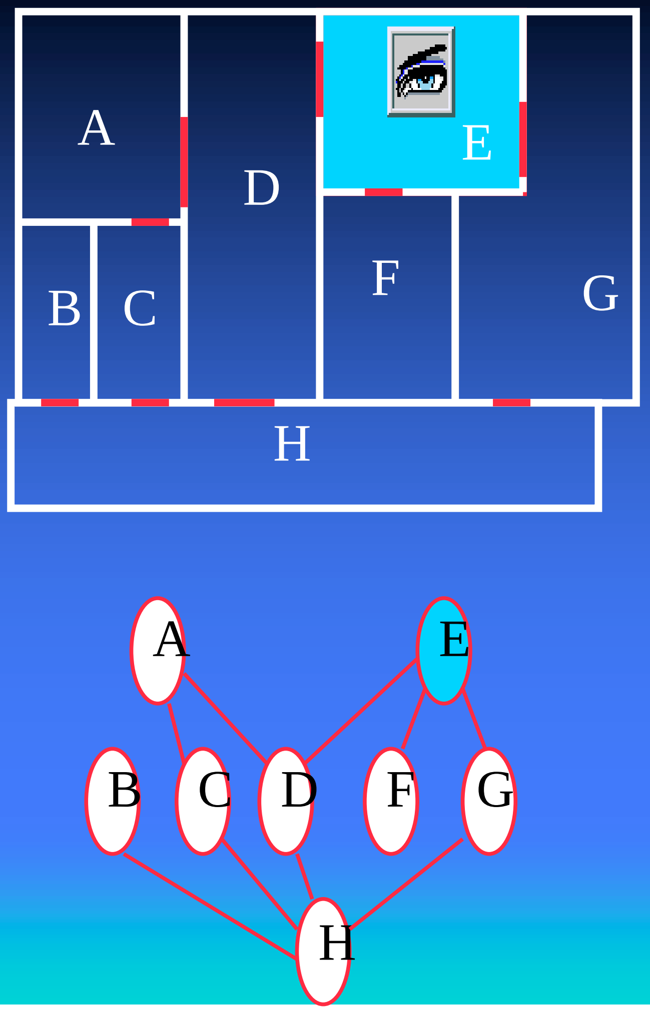
Cells and Portals - Example



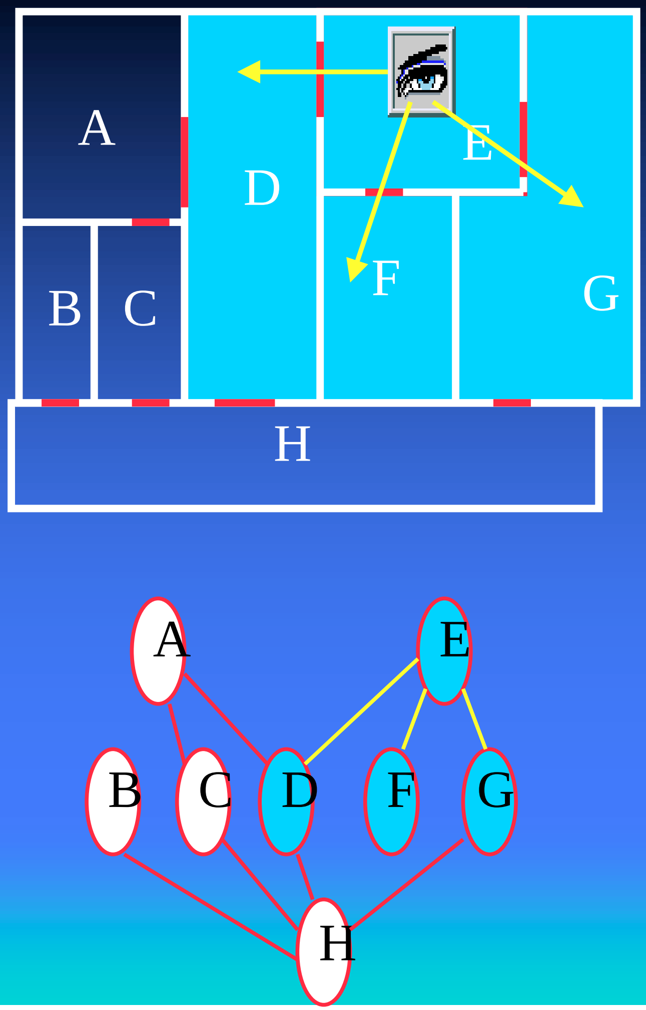
Cells and Portals - Example



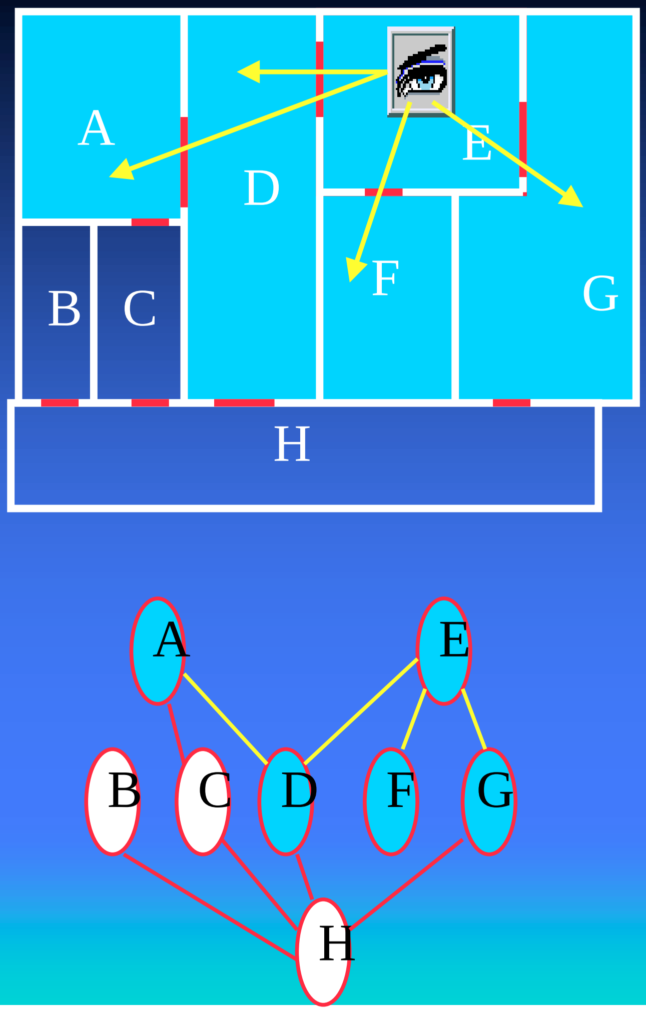
Cells and Portals - Example



Cells and Portals - Example



Cells and Portals - Example



Temporally Coherent Conservative Visibility

S. Coorg and S. Teller
MIT Laboratory of Computer Science
1996

Overview

- Polygon is visible if it is not occluded by any *single* convex object
- Visual events - changes in the visibility status of a polygon occur only when the viewpoint crosses specific planes
- From a particular viewpoint, only a small subset of such planes are relevant. As the viewpoint changes, it is sufficient to consider only these planes to detect a visual event
- Dynamic, hierarchical data

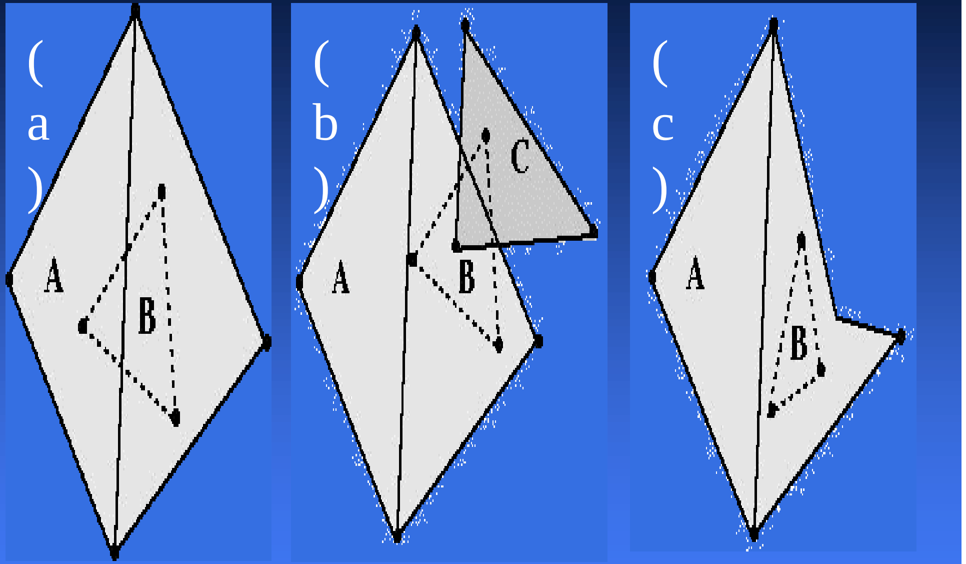
Conservative Visibility

- **Generally:**
  May classify invisible object as visible but
  may never classify visible object as invisible

- **Here:**
  A polygon is invisible iff all its vertices are
  occluded by a single convex polyhedron

Conservative Visibility

Under this definition:



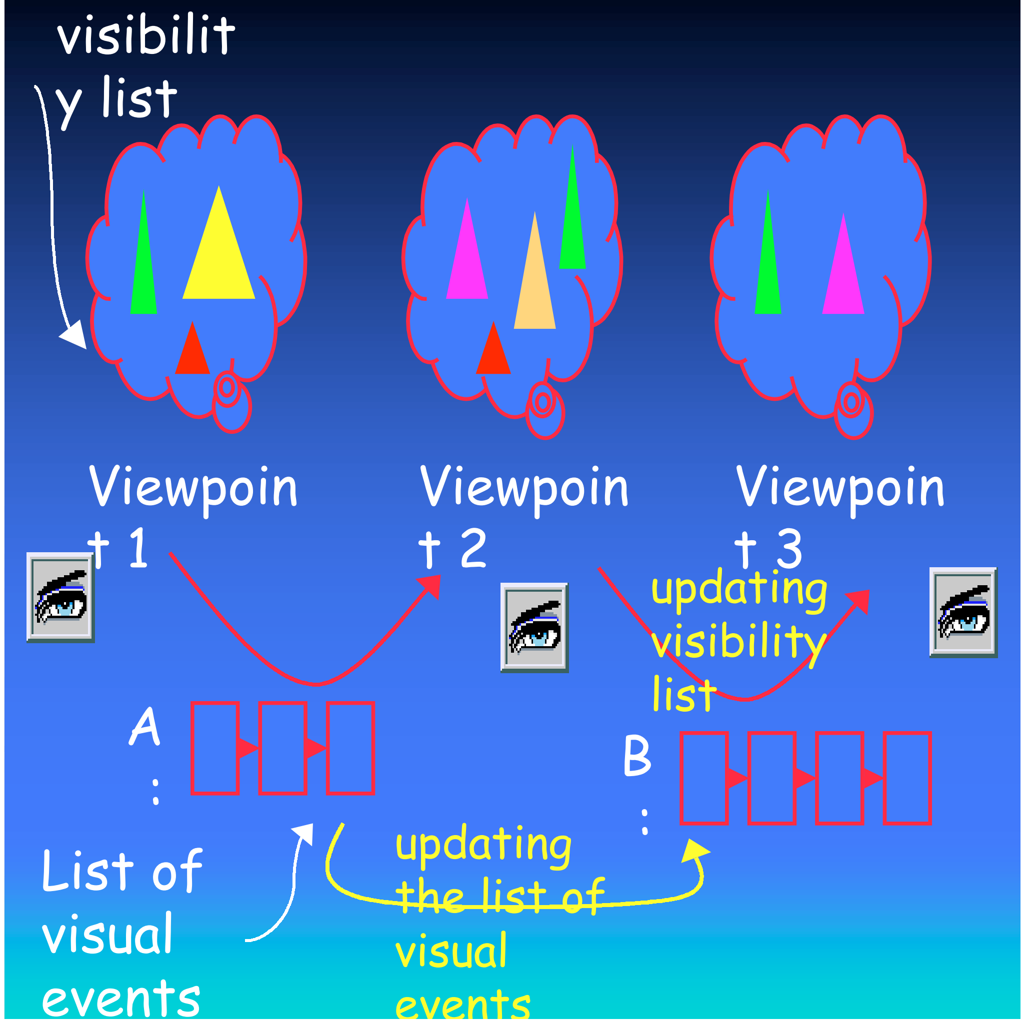
(a) an invisible polygon.
(b) B is visible although it is occluded by collusions among convex
polyhedra
(c) B is visible since the occlusion caused by non-convex polyhedra

Visual Events

- When a viewpoint moves we may track changes in the visibility
- **Visual events** - changes in the visibility status of a polygon occur only when a viewpoint crosses specific planes

---

Conservative Visibility and Visual Events



visibility list

Viewpoint 1

Viewpoint 2

Viewpoint 3

updating visibility list

A:

B:

List of visual events

updating the list of visual events

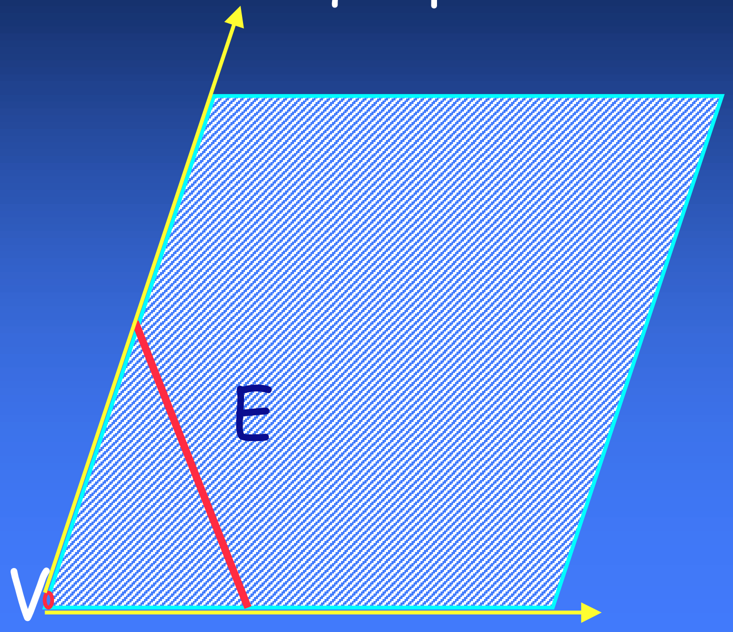
Visual Events

- The space of viewpoints can be partitioned into regions such that, within each region, the visibility remains constant. The boundaries separating these regions are called visual events
- Under the definition of visibility, there is only one kind of visual event -- vertex-edge or VE event in which, the projection of a vertex of the scene lies in the projection of an edge

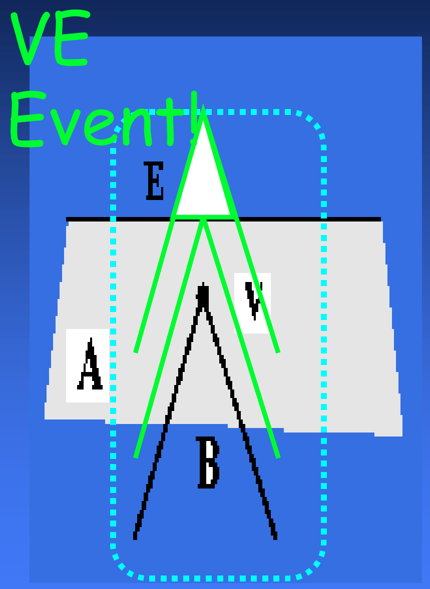
---

Visual Events

- An edge E and a vertex V formed a unique plane



Visual Events



A Naive Visibility Algorithm

The Data Structure:

- The algorithm generates planes formed by all pairs of scene vertices and edges
- Using these planes, it divides 3-dimensional space into cells
- Associates with each cell the set of polygons visible from the cell, and associates cell boundaries with changes in the visibility set.

A Naive Visibility Algorithm

At Runtime:

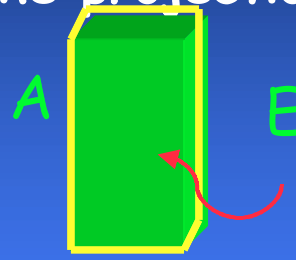
- For initial viewpoint: locates the cell containing it and reports the visible polygons associated with that cell.
- Given eye motion, any cell boundary crossing by the eye cause the visibility to be updated.

- The algorithm Reports visibility changes rather than recomputing visibility for each new viewpoint
- However major drawback of the algorithm is the excessive time and storage cost of the preprocessing step

---

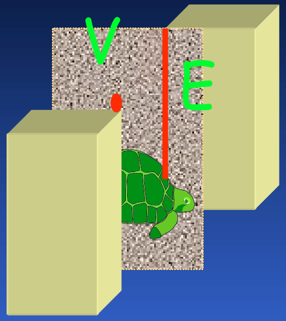
Terminology

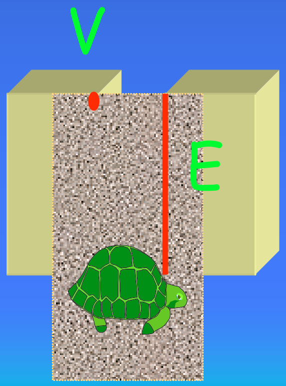
- **Silhouette** edge of a convex polyhedron A from a viewpoint is an edge E of A such that the projection of A lies completely on one side of the projection of E



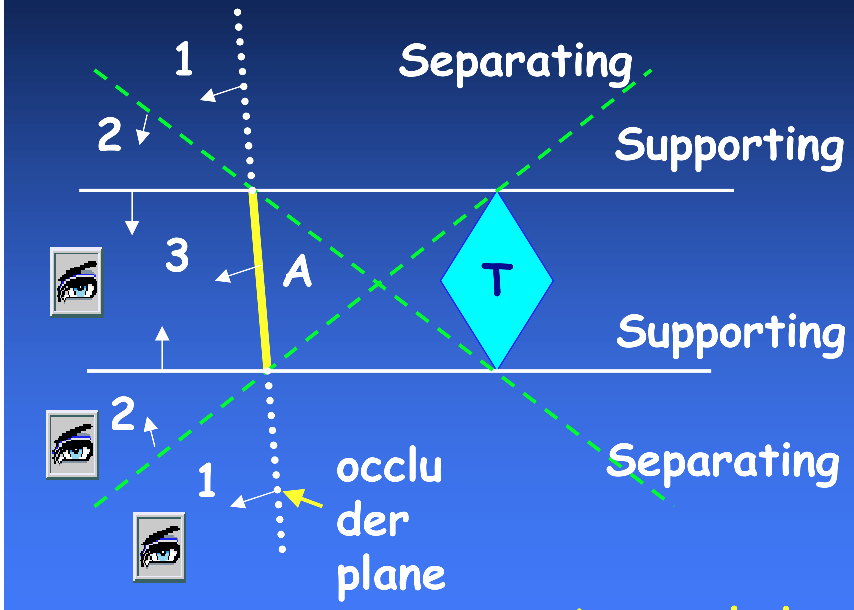
Relevant Planes - Terminology

- **Separating planes** of two convex polyhedra are planes formed by an edge of one polyhedron and a vertex of the other such that the polyhedra lie on opposite sides of the plane
- **Supporting planes** are similar, except that both polyhedra lie on the same side of





Relevant Planes - Terminology



1 - T is not occluded by A

2 - T is partially occluded by A

3 - T is completely occluded by A

A - occluder

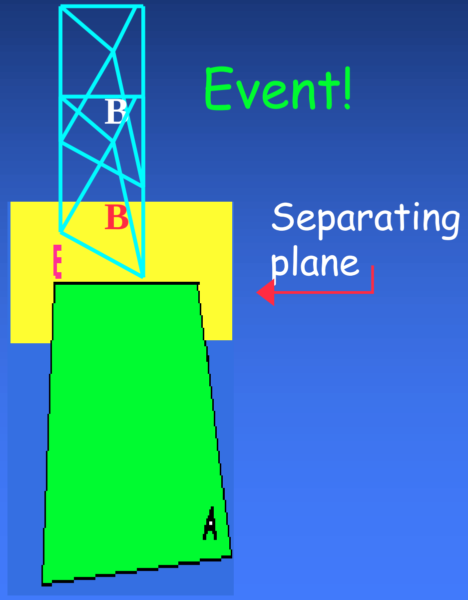
T - occludee

Relevant Planes

- Subset of all VE planes - called **relevant planes** - which is guaranteed to contain the planes which relevant to the current cell
- For occlusion relation between two convex polyhedra, it is sufficient to maintain only the **supporting** and **separating planes** (formed by a silhouette edges and vertexes of the occluder)

Separating Plane

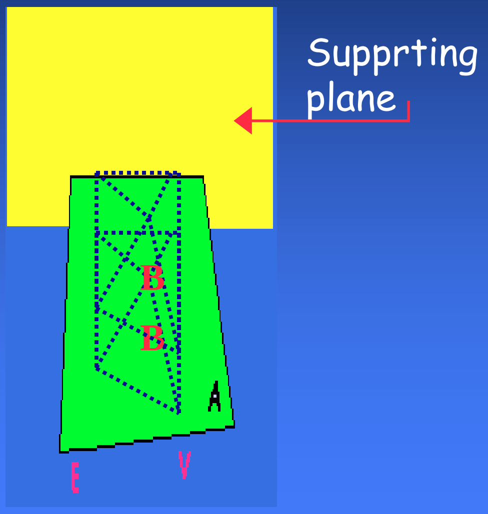
The interaction between polyhedra A and B, with A occluding B.



For this viewpoint, only the separating planes of polyhedra A and B are relevant. The first visual event that can happen is for these two polyhedra to (begin to) overlap in the image.

Supporting Plane

Event!



Supprting plane

For this viewpoint, only the separating planes of polyhedra A and B are relevant. The first visual event that can happen is for these two polyhedra to (begin to) overlap in the image.

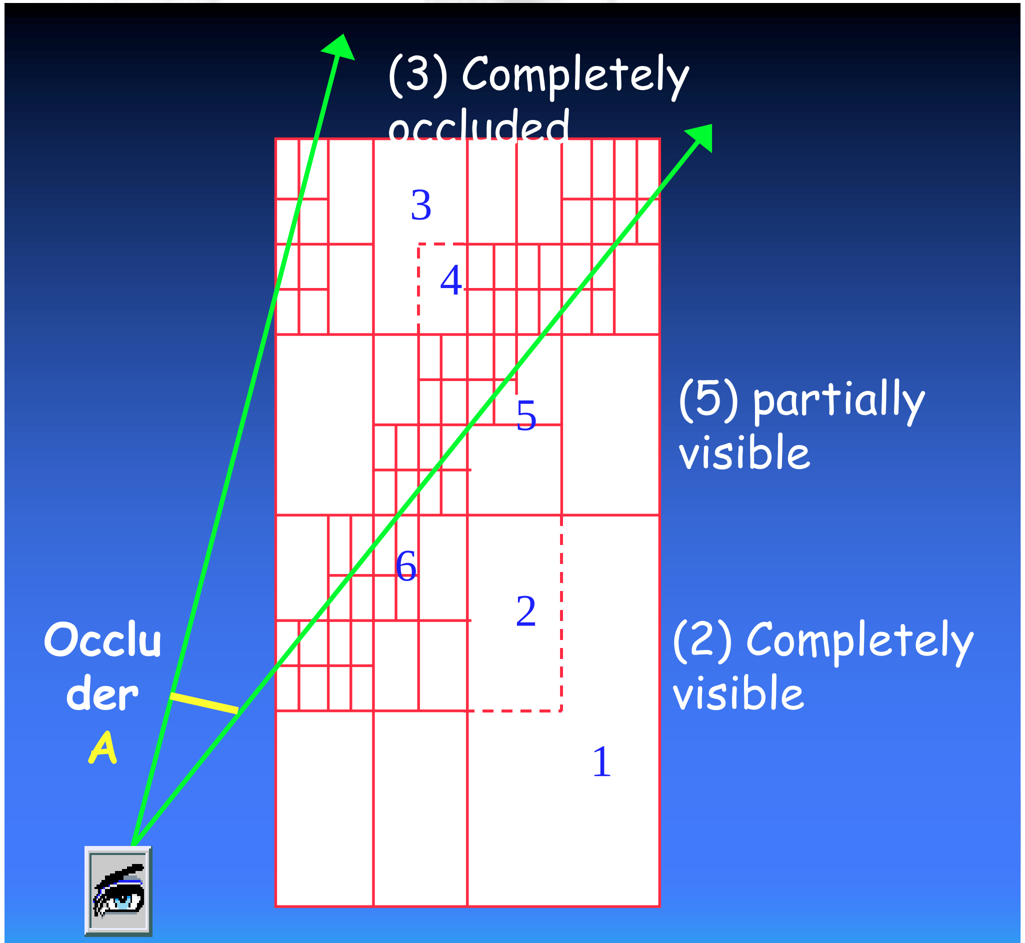
Object Hierarchies: Octree Data Structure

Build an octree:

- Root node - bounding box containing all objects
- Recursively, subdivide the box to 8 boxes until some termination criteria (e.g number of object in leaf is less than some K)

- Given an octree, we can determine the objects that are not occluded by a **single occluder A**:

Object Hierarchies: Octree Data Structure



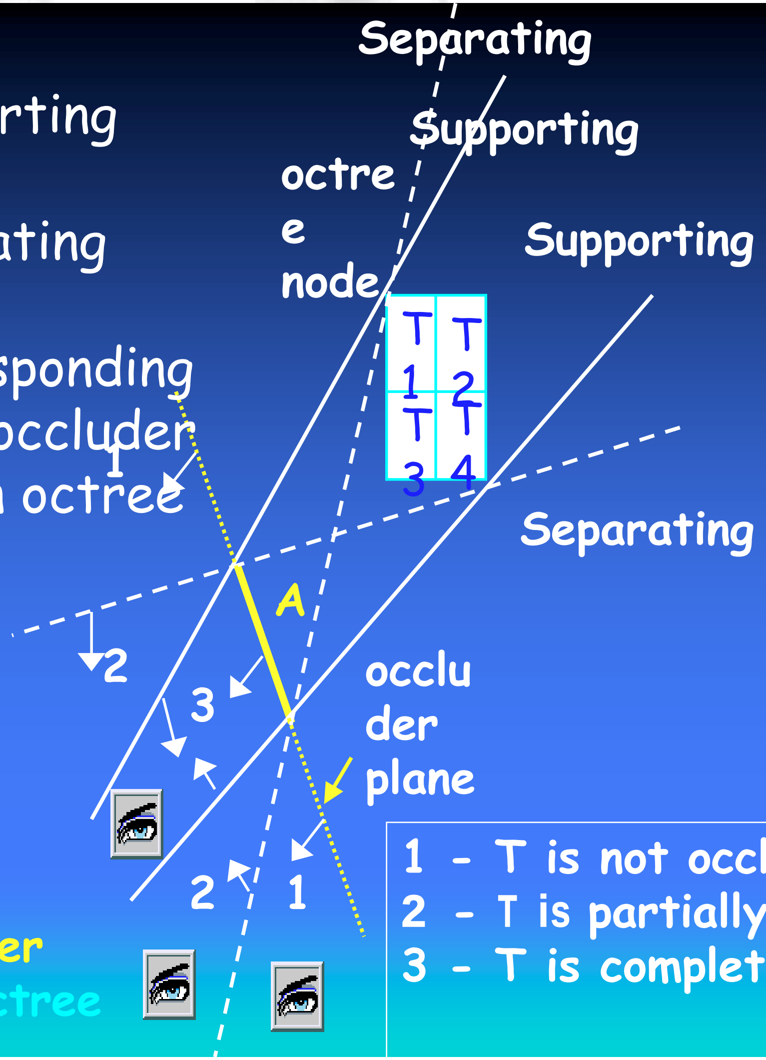
Identify Visible Objects

```
Visible (Octree Node T, Occluder A)
if A completely occludes T return;
if T is a leaf
        check visibility for each object in T and report
else        /* T is an internal node */
        if T is visible with respect to A
                report all objects within its children as visible
        else if T is partially visible
                for each child Ts of T
                        visible(Ts, A)
```

Visits only the nodes T whose visibility status is different from T's parent

Identify Visible Objects

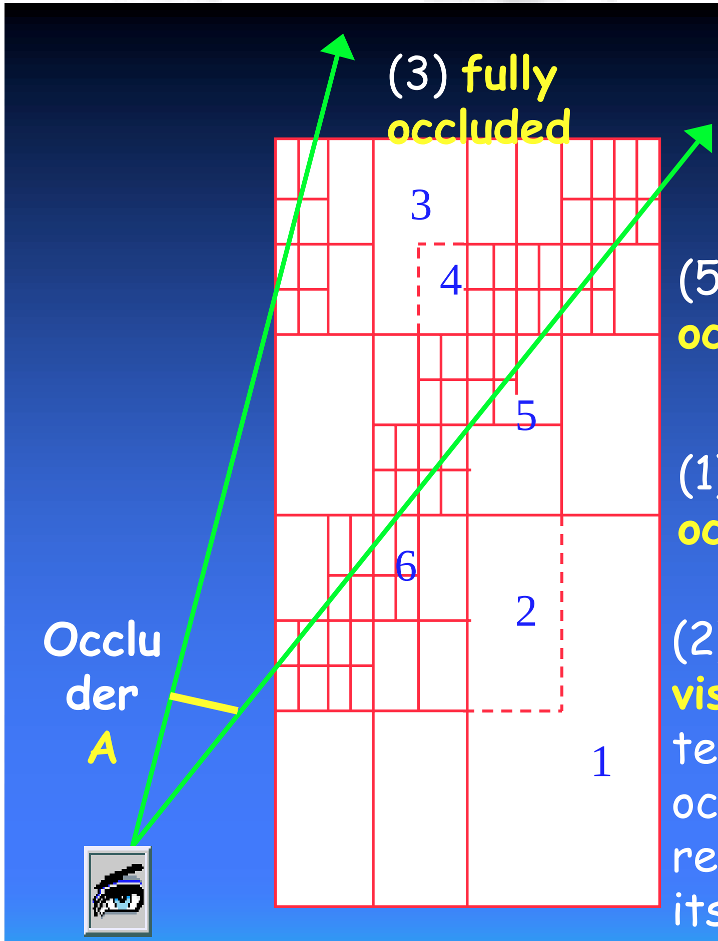
Supporting and separating planes corresponding to an occluder and an octree node



1 - T is not occlude
2 - T is partially occ
3 - T is completely

A - occluder
T - octree node

Octree Nodes Classification



(3) **fully occluded**

(5),(6) **partially occluded**

(1) **not occluded**

(2), (4) **Non visited** - The test for occlusion is resolved by its ancestor

Relevant planes

When the viewpoint moves:

- for non-visited nodes- all separating/supporting planes are irrelevant

For visited nodes the set of relevant planes is:

- for fully-occluded - supporting planes
- for not-occluded - separating planes
- for partially occluded - union of supporting and separating planes

---

The Algorithm: Updating Octree Status

- When the viewpoint crosses a relevant plane and enters a new region, the status of any affected octree node is updated, including its children, terminating whenever the status is found to be unchanged
- Summary: The algorithm processes only changes in octree status, rather than the entire octree, for each viewpoint
- Multiple occluders can be handled

Summary

- Polygon is visible if it is not occluded by any *single* convex object
- Visual events - changes in the visibility status of a polygon occur only when the viewpoint crosses specific planes
- From a particular viewpoint, only a small subset of such planes are relevant. As the viewpoint changes, it is sufficient to consider only these planes to detect a visual event
- Dynamic, hierarchical data

Accelerated Occlusion Culling using Shadow Frusta

T.Hudson D.Manocha J.Cohen
M.Lin H. Zhang
University of North Carolina
1997

Terminology

- **Occluder** - object from the model which occludes most of the others objects. convex or can be expressed as union of two convex objects
- **Shadow Frustum** - the space which is not visible from the viewpoint due to the occluder

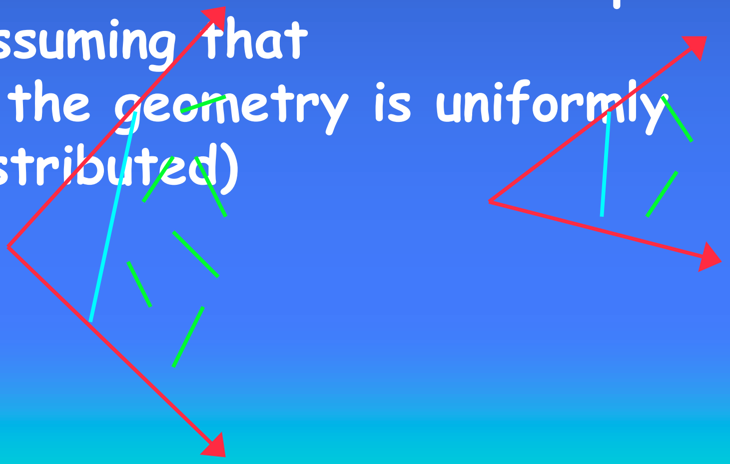
The Algorithm

- **Conservative:** May classify invisible object as visible but may never classify visible object as invisible
- Two stages:
  - **Step 1:** Select a small set of good occluders to use
  - **Step 2:** Given good occluders, use them to cull away occluded portions of the model
- **Empirical Observation:** A few occluders cause most of the occlusion from most viewpoints, and using other occluders contributes little

Step 1: Occluder Selection

Guiding Principles for the value of an occluder:

- **Solid Angle:** The viewed solid angle of a convex object measures the fraction of the visual field that is occupies (assuming that the geometry is uniformly distributed)



Step 1: Occluder Selection

Guiding Principles for the value of an occluder:

- **Depth Complexity:** select some random viewpoints in each region and determine the number of objects contained in the shadow frustum. The average of several samples is a direct estimate of the value of the occluder

Step 1: Occluder Selection

- **Preprocess:**
  Constructing a spatial partition which divides the model into **regions.**
  Each region will store a list of potentially good occluders for all viewpoints within it.

Step 1: Occluder Selection

Runtime Computation (At every frame):

- Find the region which contains the viewpoint.
  The region has list of potentially good occluders.
- The list is narrowed by view-frustum culling to determine which occluders lie within the field of view.
- These potential occluders are sorted based on the optimization function and the K first occluders are used as that frame's occluders.

Step 2: Visibility Culling Using Occluders

- Construct a hierarchy of **bounding volumes** that contain the entire model:
  - Each node of the tree contain one bounding box of the entire model
  - Each polygon lies in exactly one leaf
  - Each internal node contains the volumes of all its descendants

---

Step 2: Visibility Culling Using Occluders

- Traversing the tree:
  - View-frustum culling
  - Visibility culling using occluders by intersecting with their shadow-frusta
  - Rendering

Thank You for Listening