Visibility for Computer Graphics

Xavier Décoret
Master IVR 2004

Foreword

Visibility in other domains
- Robotics
  - path planning
- Vision

Visibility in CG
- Real-time rendering
- Lighting computations

What you will learn

- Stakes & issues
- Definitions & terminology
- Algorithms Toolkit

Context (1/3)

- Models are costly to display
  - Geometric complexity
    - intersections in ray-tracing
    - projection & rasterization in OpenGL/DX9
  - Appearance complexity
- We must treat only what’s necessary
  - Is it visible?
  - How much is it visible?
    - LOD selection

“Ce que l’on ne voit pas, on peut l’ignorer.”
Graham Greene

Context (2/3)

- Realism requires light simulation
  - Shadow casting
    - hard & soft shadows
  - Light transport
    - radiosity
- We must find amounts of light received
  - Do I “see” a light source?
  - How much do I “see” it?
    - Umbra intensity
    - Form factors

“Le soleil ne sait rien de l’ombre.”
Eugène Guillevic

Context (3/3)

- Two domains of application
  - Occlusion Culling
    - more about “is it visible?”
  - Lighting Computations
    - more about “how much is visible?”
- Common problematic
  - “What is seen from here in that direction?”
  - Dual but equivalent terminology
Context (3/3)

- Two domains of application
  - Occlusion Culling
    - more about “is it visible?”
  - Lighting Computations
    - more about “how much is visible?”
  - Common problematic
    - “What is seen from here in that direction?”
    - Dual but equivalent terminology

Occlusion culling (1/4)

- Definition
  - Quickly reject what is not visible
- Goal
  - Reduce unnecessary processing
    - Ex: “How do you draw a white wall?”
      - draw the terrain behind
      - draw a castle on the terrain
      - draw trees around the castle and cattle in the field
      - draw the white wall!

Occlusion culling (2/4)

- Definition
  - quickly reject what is not visible
- Goal
  - Reduce unnecessary processing
    - Meaning of “visible”? 
      - no ray from eye to element 
      - do not contribute to final image

Occlusion culling (3/4)

- Definition
  - Quickly reject what is not visible
- Goal
  - Reduce unnecessary processing
    - Example
      - Hierarchical Frustum Culling

Hidden Face Removal

- Definition
  - For each ray/pixel, find visible surface
- Goal
  - Guarantee image is “correct”

Occlusion culling (4/4)

- Definition
  - Quickly reject what is not visible
- Goal
  - Reduce unnecessary processing
    - Example
      - Hierarchical Frustum Culling
Occlusion culling (4/4)

- **Terminology**
  - Viewpoint/viewcell: a point/region from where we compute visibility
  - Visible Set: the set of elements exactly visible from a viewpoint/viewcell
  - Potentially Visible Set: the set an algorithm thinks is visible from a viewpoint/viewcell

- **Classification**
  - Conservative: \( \text{VS} \subseteq \text{PVS} \)
  - Aggressive: \( \text{VS} \not\subseteq \text{PVS} \)
  - \( \forall e \in \text{VS}; \text{PVS} e \) is hardly visible

What causes occlusion

- An occludee is hidden by several occluders
- Occluder fusion is important
- Occluder fusion is difficult to account for
  - from point: fused umbra

From point vs. From region

- Two approaches for culling
  - Compute PVS online for current viewpoint
  - Compute PVS offline for finite # of viewcells
    - partition the navigable space in viewcells
    - pre-compute PVS offline for every viewcells
    - approximate \( \text{PVS(viewpoint)} \) by \( \text{PVS(viewcell \subset viewpoint)} \)
- From region visibility also useful for
  - database pre-fetching
  - viewcell placement
  - Analogy with area vs. point light sources

The Erosion Theorem

- From point \( \rightarrow \) from region
- Reduce occluders and occludees
- Different of “Extended Projections” [Durand00]
  - erode occluders, enlarge occludees
  - use projections on plane

Occlusion in ray-space (1/2)

- Set of rays \( S \) from viewpoint/cell to occludee
- Each occluder blocks a set of rays \( B_j \)
- Hidden iff the union of \( B_j \) is dense in \( S \)
  - we ignore “single” unblocked rays
  - computations in dual space

What causes occlusion

- An occludee is hidden by several occluders
- Occluder fusion is important
- Occluder fusion is difficult to account for
  - from point: fused umbra
  - from region: fused umbra and penumbra
Occlusion in ray-space (2/2)
- Feasible in 2½D [Bittner01]
- Harder in 3D
  - Ray-space is 4D embedded in 5D (Plücker)
  - Feasible exactly [Nirenstein00] but slow
  - Conservative factorization [Leyvand03]
- Robustness issues
  - Epsilon visibility [Duguet02]

Ray-space factorization [Leyvand03]
- From region
- Separate horizontal/vertical
- Horizontal: exact
- Vertical: conservative
- Hardware accelerated
- Very fast

Cell and portals
- Architectural environments
  - Cells connected by portals
- Cells are visible through sequence of portals
  - Pre-process [Teller91]

Various algorithms
- Is it conservative?
- What kind of occlusion can it detect?
- What kind of scene can it handle?
- Is it offline or online?
- What is the complexity?
  - Theoretical complexity (cpu/memory)
  - Implementation complexity
- Does it work with moving objects?

Cell and portals
- Architectural environments
  - Cells connected by portals
- Cells are visible through sequence of portals
  - Pre-process [Teller91]
  - Dynamic [Luebke95]
- Finding cells and portals
  - Floodfill [Haumont03]
    - Robustness to input
Cell and portals
- Architectural environments
  - Cells connected by portals
- Cells are visible through sequence of portals
  - pre-process [Teller91]
  - dynamic [Luebke99]
- Finding cells and portals
  - Floodfill [Haumont03]
  - robustness to input
  - Two pass [Lerner03]
    - initial partition
    - optimization of cells/portals

Occlusion map based algorithms
- Occluder selection
  - Big occluders
  - Front-to-back traversal
    - BSP
    - Kd-trees
  - Temporal coherency
- Occlusion map testing
  - Hierarchical Occlusion Map [Zhang97]
  - Hierarchical Z-buffer [Green93]
    - used by hardware [HyperZ]
  - Occlusion queries [Bittner04]

Voxelisation
- Voxelize scene
  - rasterize polygons in an octree
  - find interior/exterior by floodfill
- Visibility of pairs of cells
  - Occlude by opaque voxels
    - interior voxels
    - previously occluded voxels
  - Use simple shaft culling
  - Perform blocker extension
  - Use hierarchy to speed-up

Hardware based occlusion culling
- Use Z-buffer power to test occludee
  - start query
  - render occludee’s bounding volume
  - read back number of “visible” pixels
- Interleave with rendering
  - Goal is to avoid
    - CPU stalls
    - GPU starvation
  - Needs pulls up/downs

Occlusion map based algorithms
- Online from point method
- Overall algorithm
  - Select “good” occluders
  - Render them in an occlusion map
    - disable everything but depth
  - Test occludee’s against occlusion map
    - use bounding volume
    - use hierarchy
  - Proceed in several steps
- Image space accuracy

Hardware based occlusion culling
- OpenGL API
  - ARB_occlusion_query

```c
ARB_occlusion_query Parameters
```

```c
ARB_occlusion_query
```
Horizon culling

- Overall algorithm
  - Render front-to-back
  - Maintain conservative horizon
  - Test occludee against horizon

- Suitable to 2D scenes
  - Terrain rendering [Loyd02]
  - Urban scenery [Downs01]

PVS compression

- From region visibility
  - How do you place viewcells?
  - How do you represent the PVS efficiently?

- Overall approach
  - Use small viewcells
  - Compare adjacent viewcells
  - Merge if PVS are "closed"

- Visibility matrix [DePanne99]
  - lossless/lossy
  - RLE + clusterization

- Other work by [Zach03]

Conclusion

- A very rich field
  - Just an overview!

- Keep in mind
  - What’s difficult
    - occluder fusion
  - How to evaluate/choose an algorithm
    - from region/from point
    - online vs. offline
    - exact/conservative/aggressive [Nirenstein04]
    - image space vs. object space