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

focus of this talk

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
    - transmission (CPU ↔ GPU,server ↔ client)
  - 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

Hardly Visible Sets [Andujar00]

CC Shadow Volumes [Loyd03]

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

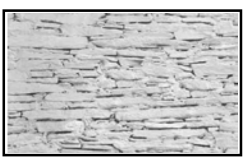
focus of this talk

Hardly Visible Sets [Andujar00]

CC Shadow Volumes [Loyd04]

Occlusion culling (1/4)

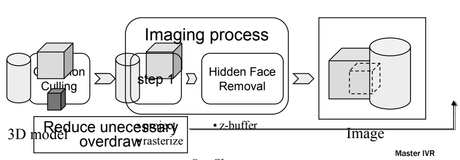
- Definition
  - Quickly reject what is not visible
- Goal
  - Reduce unecessary processing
  - Ex: "*How do you draw a white wall?*"



- draw the terrain behind
- draw a castle on the terrain
- draw trees around the castle and cattle in the field
- draw the white wall !

Hidden Face Removal

- Definition
  - For each ray/pixel, find visible surface
- Goal
  - Guarantee image is "correct"
- Hidden Face Removal *vs.* Occlusion Culling



Occlusion culling (2/4)

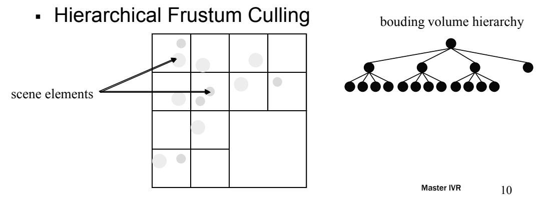
- Definition
  - Quickly reject what is not visible
- Goal
  - Reduce unecessary processing

early cheaply

- The problem of granularity
  - Cost *vs.* benefit
    - ⊙ OpenGL optimizations
    - ⊙ Bounding volumes
  - Hierarchical culling
- Meaning of "visible"?
  - no ray from eye to element
  - do not contribute to final image

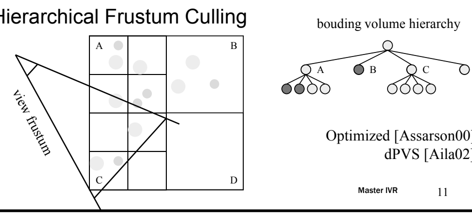
Occlusion culling (3/4)

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



Occlusion culling (3/4)

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



Optimized [Assarson00]
dPVS [Aila02]

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 $VS \subseteq PVS$
  - Aggressive $VS \not\subset PVS$
    $\forall e \in VS/PVS$ e is hardly visible

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 PVS(viewpoint) by PVS(viewcell $\supset$ viewpoint)
- From region visibility also useful for
  - database pre-fetching
  - viewcell placement
- Analogy with area *vs.* point light sources

The Erosion Theorem

- From point → from region
- Reduce occluders <u>and</u> occludees
- Different of "Extended Projections" [Durand00]
  - erode occluders, enlarge occludees
  - use projections on plane

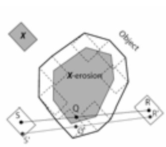
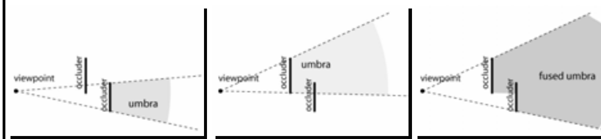


Figure 1: *If a ray [SR] is blocked by the* **X**-*erosion of an object, then any ray [S'R'] joining two points located in the convex* **X**-*shaped regions around S and R is blocked by the object.*

[Wonka00]
[Decoret03]

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

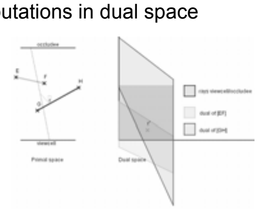


What causes occlusion

- 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 <u>and</u> penumbra

Occlusion in ray-space (1/2)

- Set of rays S from viewpoint/cell to occludee
- Each occluder blocks a set of rays B_i
- Hidden iff the union of B_i is dense in S
  - we ignore "single" unblocked rays
  - computations in dual space

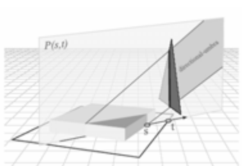


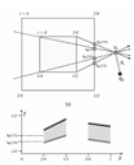
[Bittner01]

Occlusion in ray-space (2/2)

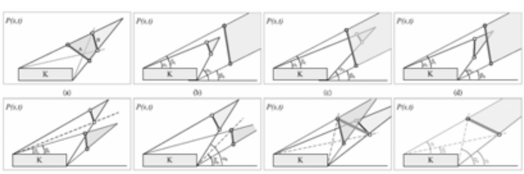
- Feasible in 2½D [Bittner01]
- Harder in 3D
  - Ray-space is 4D embedded in 5D (Plücker)
  - Feasible exactly [Nirenstein00] but slow
  - Conservative factorization [Leyvand03]
- Robustness issues
  - Epsilon visibility [Duguet02]

Ray-space factorization [Leyvand03]





- from region
- separate horizontal/vertical
  - horizontal: exact
  - vertical: conservative
- hardware accelerated
- very fast

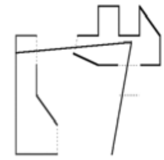


Various algorithms

- 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?

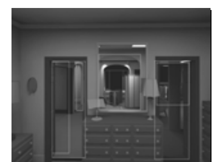
Cell and portals

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

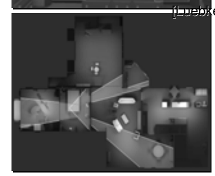


Cell and portals

- Architectural environments
  - Cells connected by portals
- Cells are visible through sequence of portals
  - pre-process [Teller91]
  - dynamic [Luebke95]



[Luebke95]



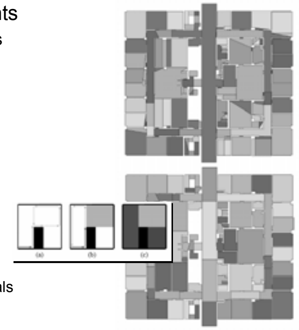
Cell and portals

- 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

Figure 4: *Different steps of the watershed algorithm. Two catchment basins getting in contact during an iteration reveal the presence of an opening: a portal is built to separate them.*

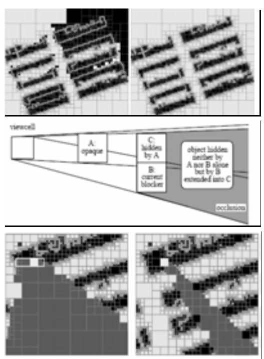
Cell and portals

- 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
  - Two pass [Lerner03]
    - initial partition
    - optimization of cells/portals



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



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

Occlusion map based algorithms

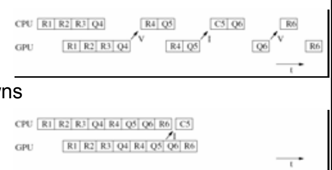
- Occluder selection
  - Big occluders
  - Front-to-back traversal
    - BSP
    - Kd-trees
  - Temporal coherency
- Occlusion map testing
  - Hierarchical Occlusion Map [Zhang97]
  - Hierarchical Z-buffer [Green93]
    - used by hardware [HyperZ]
  - Occlusion queries [Bittner04]

Hardware based occlusion culling

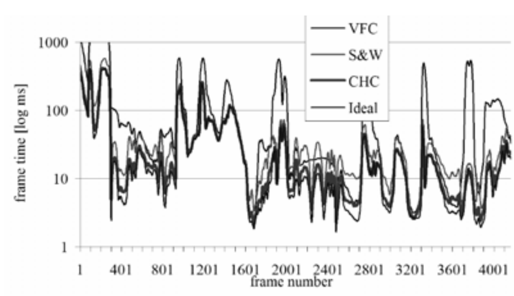
- Use Z-buffer power to test ocludee
  - start query
  - render occludee’s bounding volume
  - read back number of “visible” pixels

} OpenGL API
ARB_occlusion_query

- Interleave with rendering
  - Goal is to avoid
    - CPU stalls
    - GPU starvation
  - Needs pulls up/downs

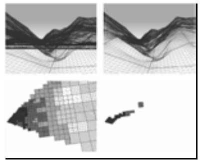


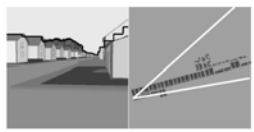
Hardware based occlusion culling



Horizon culling

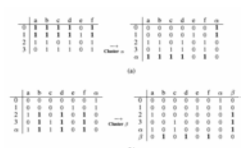
- Overall algorithm
  - Render front-to-back
  - Maintain conservative horizon
  - Test occludee against horizon
- Suitable to 2D scenes
  - Terrain rendering [Loyd02]
  - Urban scenery [Downs01]





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
  - http://artis.imag.fr/~Xavier.Decoret/bib/visibility/
  - 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