Real-Time Rendering (Echtzeitgraphik)

Dr. Michael Wimmer
wimmer@cg.tuwien.ac.at
Visibility
Overview

- Basics about visibility
- Basics about occlusion culling
- View-frustum culling / backface culling
- Occlusion culling
  - From a point
    - Object space / image space
  - From a region
    - Cells – portals / extended projections
    - Point sampling / line space
What Can You Learn from this Lecture

- **Terminology** and **problems** of visibility computation
- **Principles** of existing algorithms
- Goal: judge existing algorithms, design your own visibility algorithms
Visibility is Researched in …

- Computer graphics
- Computational geometry
- Computer vision
- Robotics
- Architecture
- GIS
- …
Applications in Computer Graphics

- Occlusion culling
- Shadows
- Global illumination
- Hidden-surface removal
- Viewpoint selection
- Image-based rendering
- ...

...
Basics of Visibility
Visibility from a Point

Terms: occluder, occludee, shadow volume = umbra
Visibility from a Point

Complete point umbra for occluders $\text{occ}_1, \ldots, \text{occ}_n = \text{union}$ of all individual umbrae
Occluder Fusion

- **Occluder fusion**: exploit combined effect of multiple occluders

- No occluder fusion:
  - Test against individual umbrae → visible

- Occluder fusion:
  - Test against **complete umbra** → invisible
Simple Algorithm for Point Visibility

- Umbra data structure (UDS) = empty
- For each occluder occ<sub>i</sub>
  - Calculate umbra U<sub>i</sub>
  - Add U<sub>i</sub> to UDS
- Test the scene against the UDS to see what is visible / occluded

- Examples for UDS: BSP-tree, z-buffer, …
Visibility from a Region

(Example in 2D)

occluder

(viewing region)

(region) umbra
Visibility from a Region

- Goal: find complete (region) umbra!

- Try: union of (region) umbrae…
Visibility from a Region

- Test: from-point visibility for some viewpoints…

- Viewpoint 1: XXX invisible
Test: from-point visibility for some viewpoints…

- Viewpoint 5: XXX invisible
Visibility from a Region

XXX is always occluded $\rightarrow$ suggests: complete region umbra is more than union of individual region umbra
Solution: complete region umbra for occluders $\text{occ}_1, \ldots, \text{occ}_n = \text{intersection of complete point umbrae for all viewpoints in region!}$
Important Terms 1: Umbra / Penumbra

- The area (volume) in full shadow is the **umbra**, the grey area the **penumbra**.
**Umbra** is a simple in/out classification.

**Penumbra** additionally encodes which parts of the viewing region are visible.
Important Terms 2: Supporting / Separating Planes

- Supporting planes
- Separating planes
- Occluder
- Viewing region
Supporting / Separating Planes

- Planes between two polyhedra defined by:
  - Edge of one polyhedron (view cell/occluder)
  - Vertex of other polyhedron (view cell/occluder)

- Supporting planes
  - Example: bound umbra of one occluder
  - Polyhedra on same side of plane

- Separating planes
  - Example: bound penumbra of one occluder
  - Polyhedra on opposite sides of plane
Important Terms 3: Visual Events

- Surfaces where visibility changes when a point crosses it
- Interpretation 1: point is viewpoint
  - Visual events bound regions of constant visibility
- Interpretation 2: point is “viewed point”
  - Visual events are the shadow boundaries

![Diagram of visual events](image)
Visual Events

Visual event types:

- Vertex-Edge (VE): supporting/separating planes
- Edge-Edge-Edge (EEE): curved surfaces!
Visual Events / Shadow Boundaries
Shadow Boundaries

- Visual events, interpretation 2
  - View cell always participates

Diagram:
- View cell
- Height
- Ground plane
- Points a and b
Shadow Boundaries

- Vertex/edge

View cell

OCC₁
Shadow Boundaries

- Vertex/edge

\[ \text{view cell} \]

\[ \text{OCC}_2 \]

\[ \text{OCC}_1 \]

a

b

e

c

d

f
Shadow Boundaries

- **Edge/edge/edge**

  - **VE plane:**
    - **Vertex:** a
    - **Edge:** cd

  - **VE plane:**
    - **Vertex:** b
    - **Edge:** ef

  - **EEE surface:**
    - **Edge:** cd
    - **Edge:** ef
    - **Edge:** ab

  - Shadow boundary
curved!
Recall: complete region umbra = intersection of complete point umbrae

But: impossible to calculate!

Approach: look at ways to merge penumbrae

Complete region umbra = union of individual region umbrae + all regions where penumbrae merge to umbra

Problem: How to store Penumbra?
3 ways how penumbrae merge to umbra
Occlusion Culling from a Region I

- Idea I: ignore problem completely
- Umbra data structure (UDS) = empty
- for each occluder occ\textsubscript{i}
  - Calculate umbra U\textsubscript{i}
  - Add U\textsubscript{i} to UDS
- Test the scene against the UDS (union of U\textsubscript{i})
Idea II: detect overlapping umbrae (case b)
- UDS = empty
- front-to-back: for each occluder occ\(_i\)
Occlusion Culling from a Region II

- Idea II: detect overlapping umbrae
- $\text{UDS} = \text{empty}$
- front-to back: for each occluder $\text{occ}_i$
  - Extend occluder into existing umbra
  - Calculate (extended) umbra $U_i$
  - Add $U_i$ to UDS
Idea II: detect overlapping umbrae

- UDS = empty
- front-to-back: for each occluder $occ_i$
  - Extend occluder into existing umbra
  - Calculate (extended) umbra $U_i$
  - Add $U_i$ to UDS

Test the scene against UDS (which is now more than union of original $U_i$!)
Occlusion Culling from a Region III

- Idea III: calculate everything (case c)
- Problem: complete region umbra bounded by planes and reguli (ruled, quadric surfaces with negative curvature) (recall visual events!)
- Possible solutions (see later):
  - Sample from viewpoints and shrink occluders
  - Solve problem in line space
  - Extended projections
  - Special case solutions (horizons, cells/portals)
Oriented 2D line maps to point in 2D oriented projective space (line space)

Conversely, 2D point maps to line

Parameter choice:

\[ y = k x + d \]

Plücker coordinates (in practice)
Visibility in Line Space (2D)

- All lines between the view region and an occluder map to a polygon in line space.
- “Occluder polygon”, represents all possible sight lines.
Visibility in Line Space (2D)

- Use a data structure that classifies line space as in / out to store the umbra
- Front-to-back rendering

$S =$ view area

$O_x =$ occluder
Overview of Occlusion Culling Algorithms
Visibility in Real-Time Rendering

- Interactively walk through a large model
- Large model $\rightarrow$ millions of polygons $\rightarrow$ acceleration necessary (e.g. visibility)
Why is the Z-Buffer Not Enough?

- Does not eliminate depth-complexity (overdraw) (but: early-z in newer cards)
- Does not eliminate application/vertex processing of occluded objects

Visibility should also happen here
Visibility Culling

- View-frustum culling
- Occlusion culling
- Backface culling
- View-frustum culling
- Occlusion culling
- Backface culling
Visibility Culling

- View-frustum culling
- Occlusion culling
- Backface culling
Visibility culling

- Result

- view frustum

- view point
View-Frustum Culling

- Eliminate polygons outside of the view frustum
- Hierarchical data structure
  - Bounding-volume hierarchy
  - or any spatial data structure
View-frustum culling

- Hierarchy based on bounding volume
Hierarchical view-frustum culling based on bounding volume
Hierarchical view-frustum culling using BSP (Binary Space Partitioning) trees
Hierarchical view-frustum culling using quadtree (octree)
Backface Culling

- Screen space
  - Cross product (only z is needed!)
  - Orientation of a polygon is determined by the vertex order
  - Calculated by hardware

- Eye space
  - Dot product
Occlusion Culling / Overview

- General Information
- Occlusion Culling from a point
  - Object Space
  - Image Space
- Occlusion Culling from a region
  - Cells Portals
  - Extended Projection
  - Point Sampling
  - Line Space Visibility
Occlusion Culling

- Possible results:
  - Visible
  - Partially visible
  - Occluded (invisible)
Occlusion Culling

- Calculate PVS = potentially visible set
- Exact hidden surface removal is done by the z-buffer
- PVS can be
  - Aggressive, PVS ⊆ EVS
  - Conservative, PVS ⊇ EVS (preferred)
  - Approximate, PVS ~ EVS
- EVS = exact solution (on a per-object basis)
Objects (not individual triangles) are organized in a hierarchical data structure (scene data structure SDS)

- bounding box tree
- octree, quadtree
- kd tree
- bsp tree
- …
Occlusion Culling (We need:)

- The scene organized in a hierarchical data structure (= SDS).
- A (hierarchical) data structure for the umbra (= UDS)
- A (selected) set of occluders (also stored in the SDS)
  - Sometimes all triangles in the scene can be occluders
  - If not, large polygons close to the viewpoint or viewing region are selected
Occlusion Culling (General Idea)

- Traverse the SDS top-down / front-to-back
- Test each node of the SDS against the UDS for visibility
  - If node invisible $\rightarrow$ skip node
  - If node visible $\rightarrow$
    - Traverse down or
    - mark objects in node visible and insert occluders into UDS (see earlier)

- Note: interleave creating UDS and checking SDS
Occlusion Culling Acceleration

- Ideas to accelerate occlusion culling / overcome implementation problems
  - 2.5D occlusion culling
  - Occluder selection
  - Lazy update of the UDS
  - Approximate front-to-back sorting
Idea: 2.5D Occlusion Culling

- Buildings are occluders, connected to the ground
- → 2.5D visibility algorithms
  - General 3D SDS, occluder is a function \( f(x,y) = z \)
  - → UDS only 2.5D
Idea: Occluder Selection

- Costly to use all scene polygons as occluders
  - Each occluder requires update to UDS
- Idea: Select only subset of polygons that
  - Are close to the view point (view region)
  - Have a large area
  - Are facing the view point (view region)
Idea: Lazy Update of UDS

- Normally interleave:
  - adding occluders to UDS
  - testing objects of SDS against UDS

- But: UDS can be costly to update or access
  - E.g., z-buffer

- Idea: Lazy update
  - Insert many occluders into UDS at once
  - Or: insert all occluders, then test (as in first part of lecture)
Idea: Approximate front-to-back sorting

- Exact front-to-back sorting is expensive
- Use approximate front-to-back sorting
  - Usually based on hierarchy
  - Need to be careful not to calculate incorrect occlusion, especially for visibility from a region
Occlusion Culling Algorithms: From Point

- Object space: Occlusion trees
- Image Space: Hierarchical z-Buffer
- Image Space, hardware: Occlusion Queries
Occlusion Trees

- [Bittner98]
  - SDS = kd tree
  - UDS = BSP tree
- Works fine, all sorts of occluder fusion
- Adding thousands of occluders to the UDS is slow
Hierarchical z-Buffer

- [Greene93]
  - SDS = octree
  - UDS = z-pyramid
Z-Pyramid

- Lowest level: full-resolution Z-buffer
- Higher levels: each pixel represents the maximum depth of the four pixels “underneath” it
Hardware Implementation

- Only 2-3 levels on current hardware
- Only per-fragment culling
  - Works automatically
  - Saves rasterization time
- Per-object culling: occlusion queries
  - Ask whether an object would have been rendered
  - Uses hardware pyramid
  - Problem: latency of query
Hardware Occlusion Queries

- Extension name: ARB_occlusion_query
- Returns no. of pixels that pass
  - For aggressive occlusion culling
- Provides an interface to issue multiple queries at once before asking for the result of any one
  - Allows hiding latency
  - Do other work in parallel
- Coherent Hierarchical Culling [Bittner04]
  - Exploit temporal coherence to eliminate latency and reduce queries
Occlusion Culling Algorithms: From Region

- Special case: Cells and portals
- Image space: Extended Projections
- Point Sampling
- Line Space
Visibility Preprocessing

- Subdivide view space into view cells
- Calculate PVS for each view cell
- Store all PVS on disk
Cells and Portals

- Architectural walkthroughs
- Structure scene into
  - Cells (mainly rooms)
  - Portals (mainly doors)
Cells and Portals

- Build adjacency graph
  - Cells = nodes, portals = edges
- Portal sequences
- Preprocess algorithm:
  - Test sightlines through an oriented portal sequence
  - Use depth search in adjacency graph (e.g. linear-programming)
- Online algorithm:
  - Project portals to screen space
  - Intersect with previous projected portals
Extended Projections

- [Durand2000]
  - SDS = anything
  - UDS = z-pyramid / z-buffer
- Image space algorithm
- Modifies projection of
  - Occluder (smaller)
  - Occludee (larger)
- Depending on viewing region
Point Sampling

- [Wonka2000]
- Make point sampling possible for conservative occlusion culling for a region
Idea: Shrink Occluders

Sample point

Occluder

Conservative umbra for $\varepsilon$-neighborhood
Algorithm Overview

- Shrink all occluders
- For each view cell
  - For each sample point calculate PVS
- Calculate union of all PVS
[Bittner02]

- SDS = kd tree
- UDS = Line Space BSP tree
- 3D primary space → 5D line space

J. Bittner, Havran, Slavik. Hierarchical visibility culling with occlusion trees. CGI'98.


