VISIBILITY CULLING AND RENDERING MASSIVE MODELS

VLASTIMIL HAVRAN

BASED ON PRESENTATION BY
JIŘÍ BITTNER

DCGI

Rendering Massive Models

- Real-time rendering > 60 FPS
- Fast GPUs vs.
  - Larger scenes
  - More details
  - Complex shaders
  - Multiple render passes

Massive Models – Optimizations

- Manual model optimization
  - Textures, bump maps, normal maps
- Optimal GPU utility
  - Triangle strips, vertex arrays, vertex buffer objects, optimized vertex and pixel shaders, minimize state changes
- Automatic model optimization
  - LODs, bilboards, depth impostors, point sampling, …
- Data management
  - Data prefetching, data layout, using coherence
- Visibility culling
  - Online culling, preprocessing

Visibility Culling – Motivation

- Q: Why visibility culling, when:
  - Object outside screen culled by HW clipping
  - Occluded objects culled by z-buffer in O(n) time
- A: Linear complexity not sufficient!
  - Processing too many invisible polygons
- Goal
  - Render only what can be seen!
  - Make z-buffer output sensitive

Visibility Culling Methods

- Online
  - Applied for every view point at runtime
- Offline
  - Partition view space into view cells
  - Compute Potentially Visible Sets (PVS)

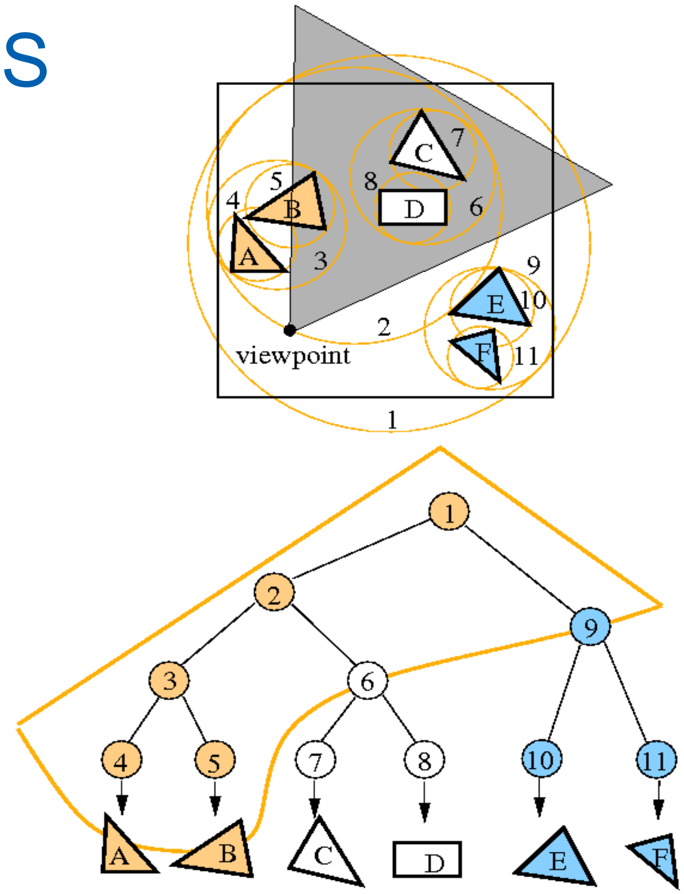
Online Visibility Culling

- For every frame cull whole groups of invisible polygons
- Conservative solution
  - Conservative determines a superset of visible polygons = guarantee
  - Precise visibility solved by z-buffer
- Basic Techniques
  - Backface culling
  - View-frustum culling

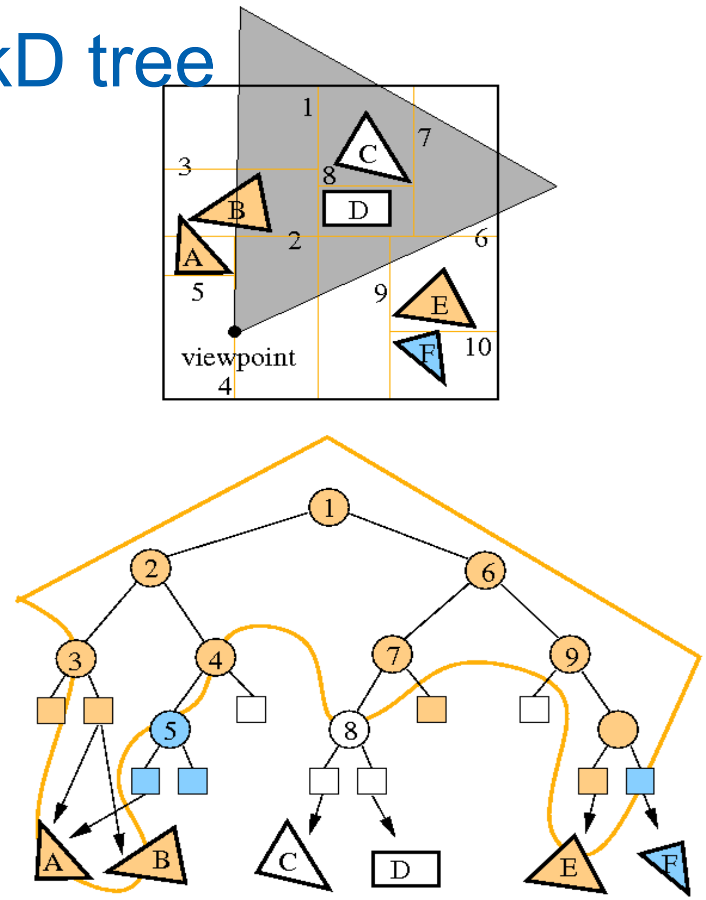
Note: remember computation exact, conservative, approximate and aggressive

View Frustum Culling

HBS



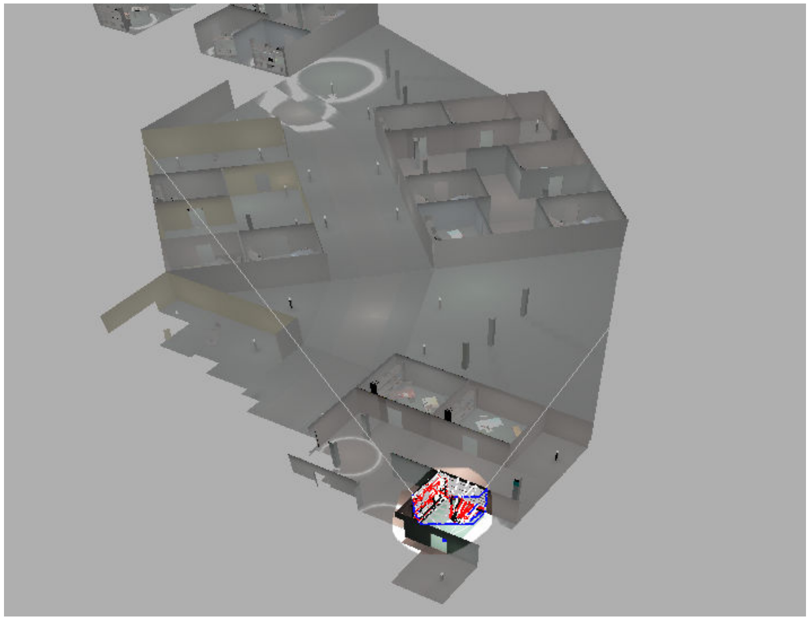
kD tree



Optimizations: Temporal coherence, efficient intersection test [Assarson00]

Occlusion Culling

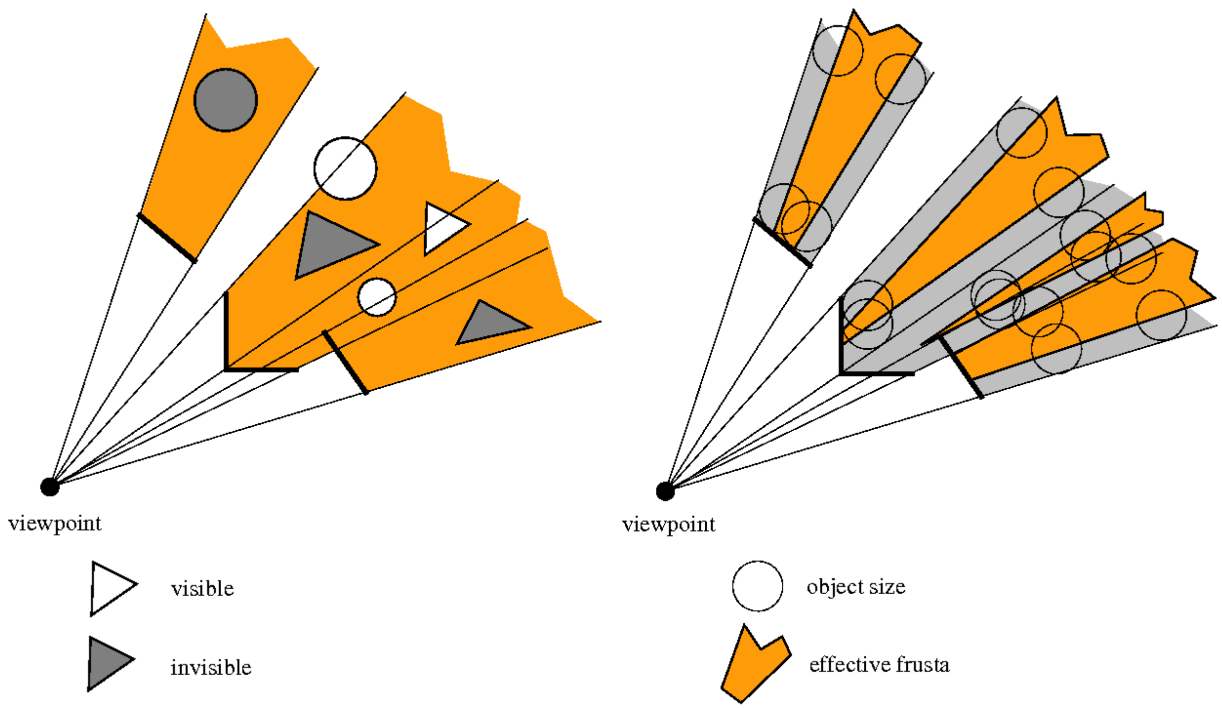
- VFC disregards occlusion
- 99% of scene can be occluded!



- Solution: Detect and cull also occluded objects

Shadow Frusta

- Construct shadow frusta for several occluders [Hudson97]



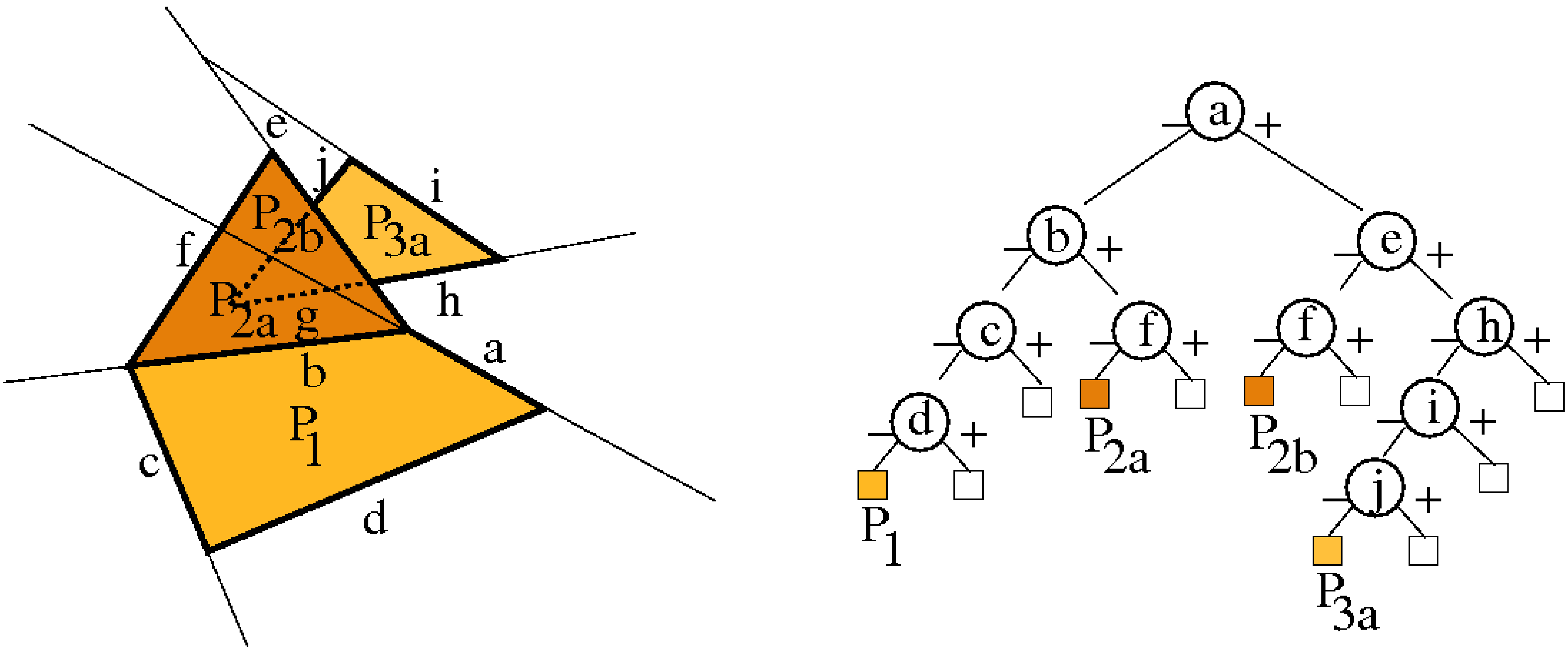
- Object is invisible when it is inside a shadow frustum
- Queries on the spatial hierarchy

Shadow Frusta - Properties

- Properties
  - + Easy implementation
  - No occluder sorting
    - No occluder fusion!
    - O(n) query time
  - Small number of occluders

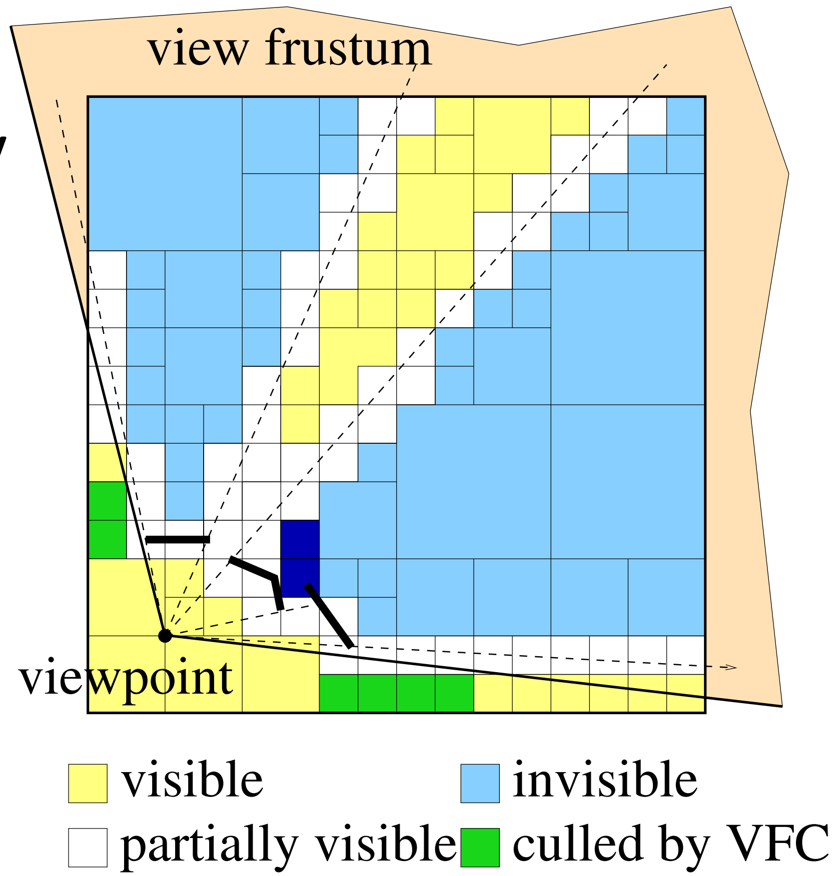
Occlusion Trees

- Occluders sorted into a 2D BSP tree (1998)
- Occlusion tree represents fused occlusion
- Example: occlusion tree for 3 occluders



Occlusion Tree - Traversal

- Visibility test of a node
  - Depth-first-search
  - Found empty leaf → tested object is visible
  - Depth test in filled leaves
- Example of final visibility classification of kD-tree



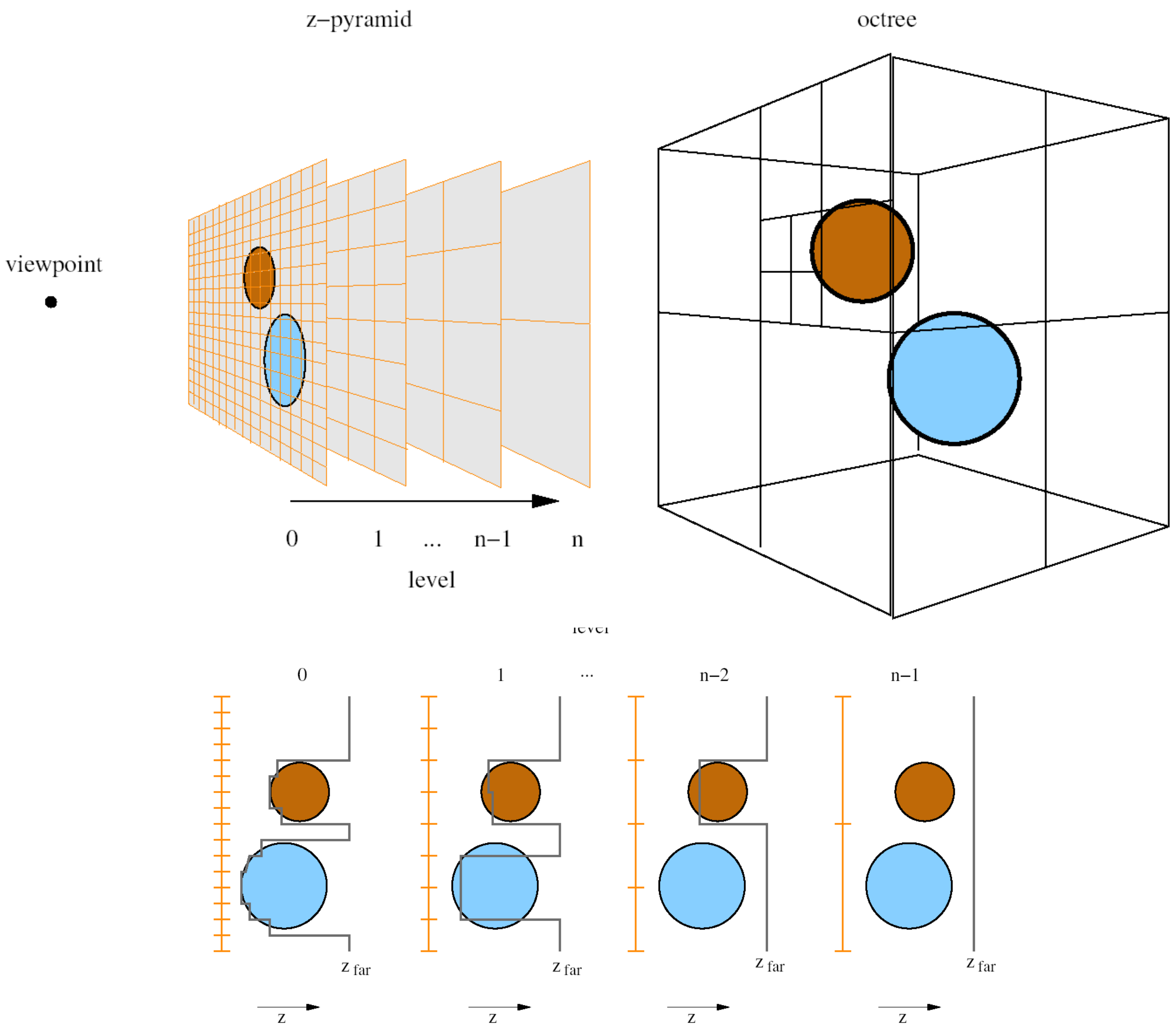
Occlusion Tree - Properties

- Presorting occluders
  - Tree size: worst case $O(n^2)$, n = #occluders
  - O(log n) visibility test
- + Allows to use more occluders (~100)
- Not usable for scenes with small polygons

Hierarchical Z-buffer

- Extension of z-buffer to quickly cull larger objects [Greene 96]
- Ideas
  - octree for spatial scene sorting
  - z-pyramid for accelerated depth test

Hierarchical Z-buffer - Example

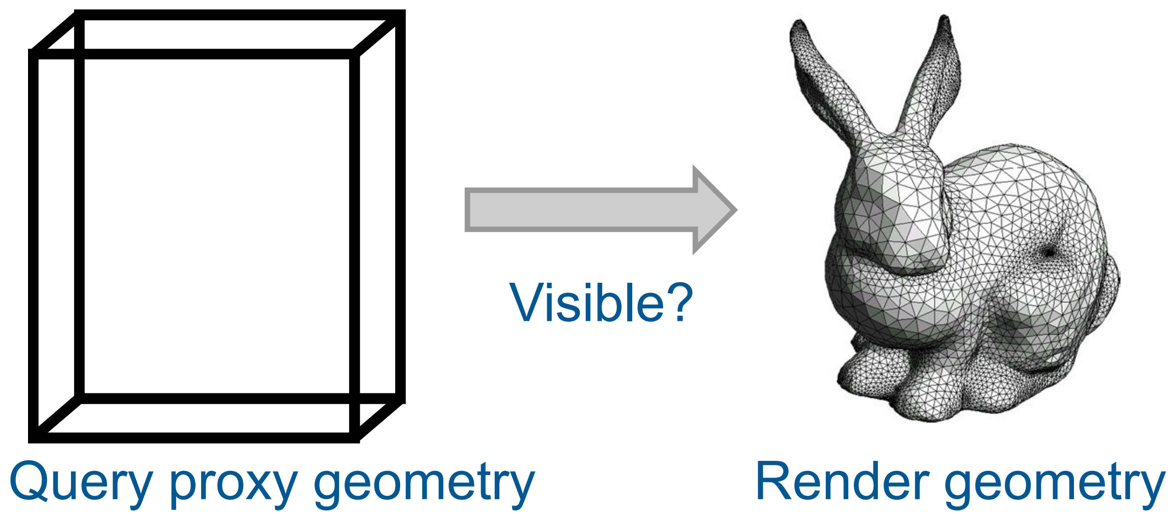


Hierarchical Z-buffer - Usage

- Hierarchical test for octree nodes that represent axis-aligned boxes
- Test on hierarchy node
  - Find smallest node of z-pyramid, which contains the tested box
  - Box depth > node depth $\rightarrow$ cull
  - Otherwise: recurse to lower z-pyramid level
- Optimization: use temporal coherence
  - z-pyramid constructed from polygons visible in the last frame

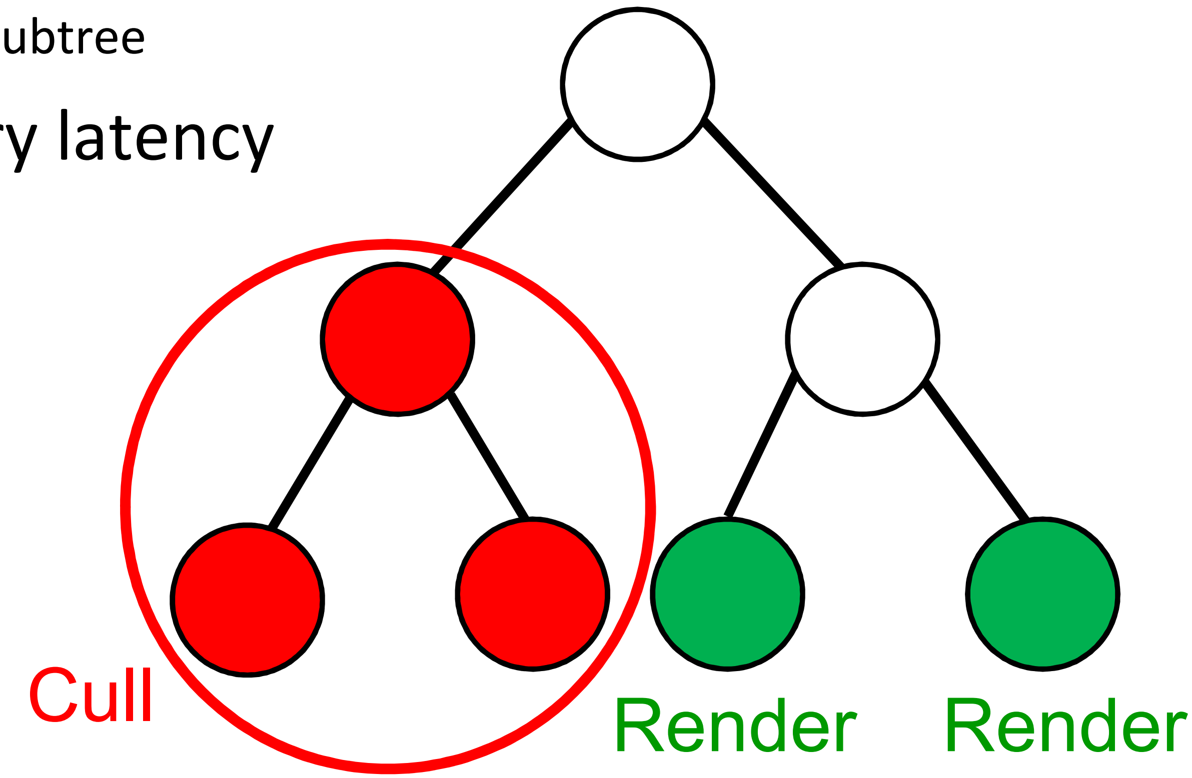
HW Occlusion Queries

- ARB_occlusion_query, NV_occlusion_query
- Return #pixels passing the depth test
- Feature which has been missing in old OpenGL!
- + No preprocessing, dynamic scenes
- - Latency, the query costs time



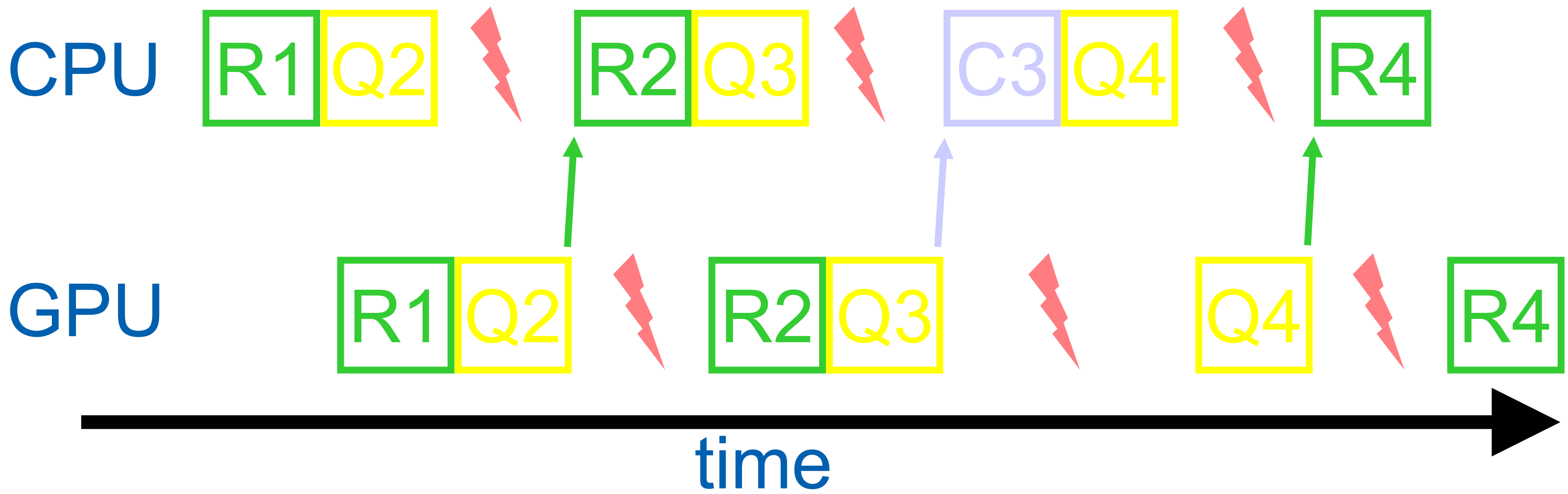
Naive Method: Hierarchical Stop & Wait

- For each node: Issue query
  - Visible → traverse subtree
  - Invisible → cull subtree
- Problem: Query latency
  - CPU stalls
  - GPU starvation



CPU Stalls

GPU Starvation



Rx Render object x

Qx Query object x

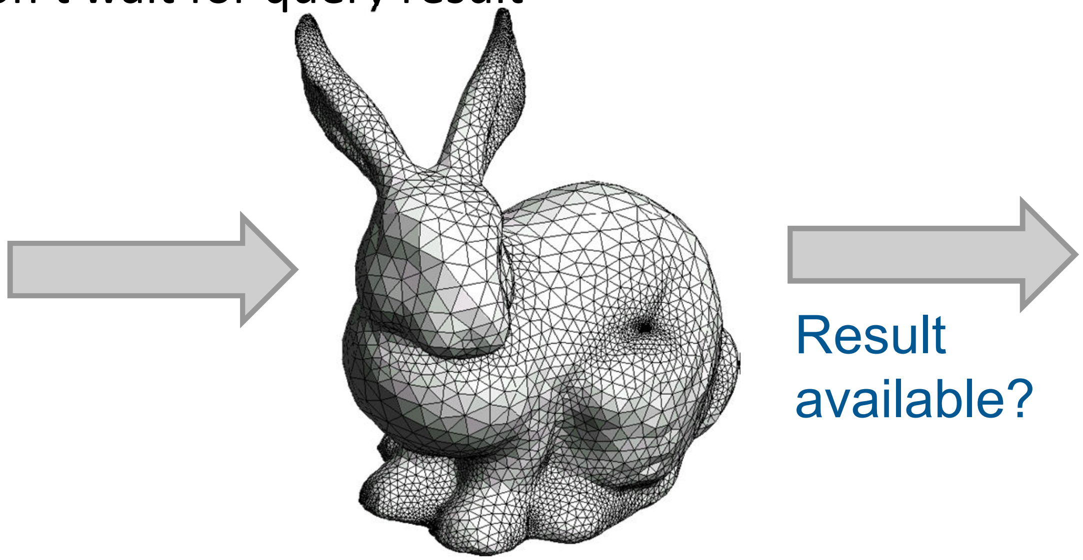
Cx Cull object x

Waiting time

m1

Coherent Hierarchical Culling (CHC)

- While waiting for query result → traverse / render
  - Keep query queue
- Use coherence, assume node stays (in)visible
- For previously visible nodes
  - Don‘t wait for query result

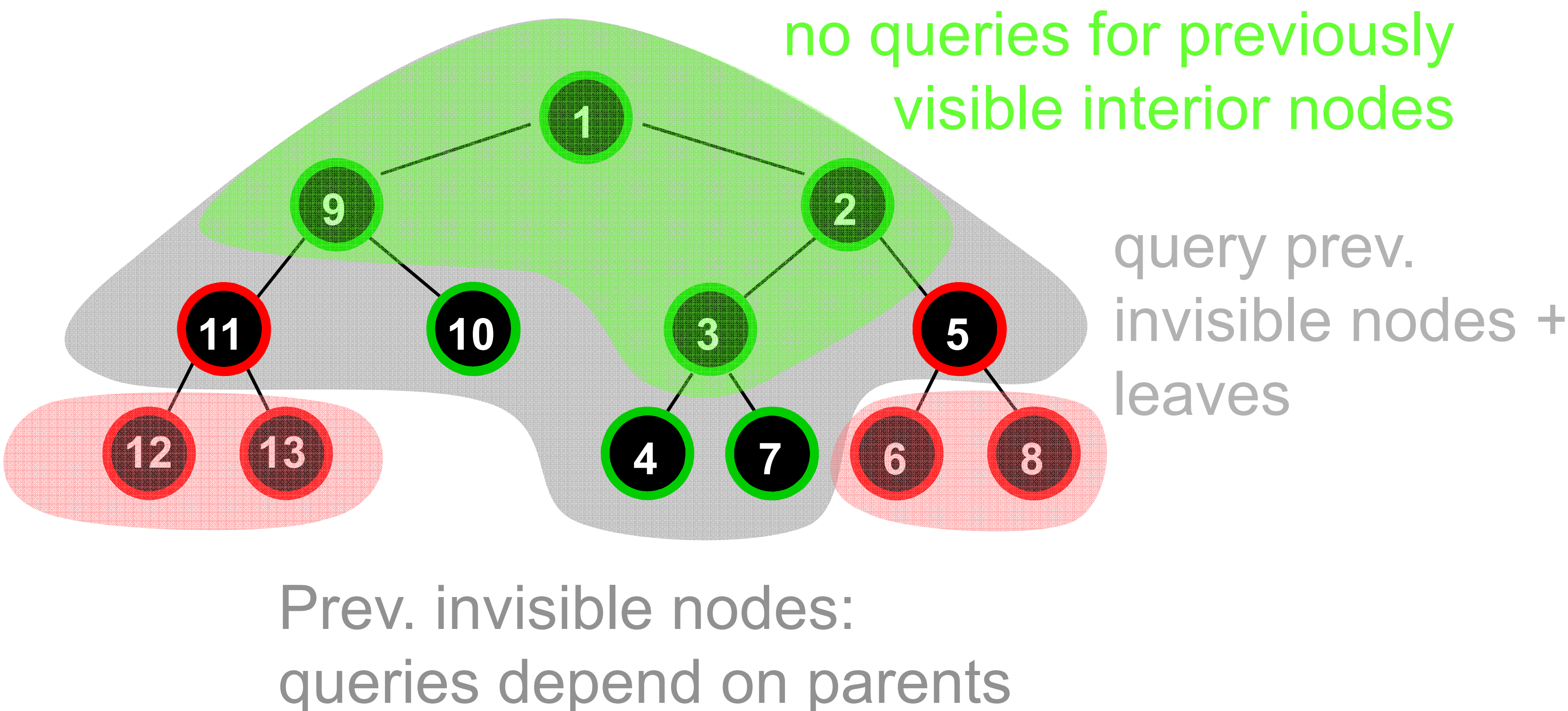


Snímek 20

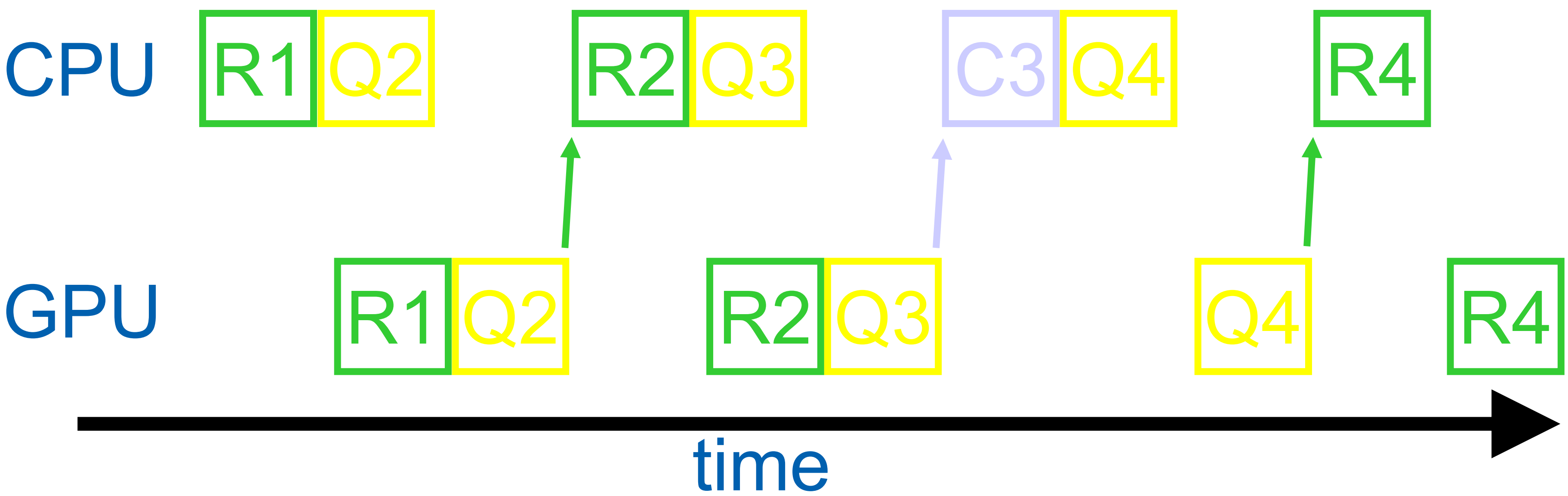
m1 too much text: merge
figure: show visible / invsible queries
matt, 5/13/2007

Coherent Hierarchical Culling

- Interleave queries and rendering
- Schedule queries based on temporal coherence



CHC

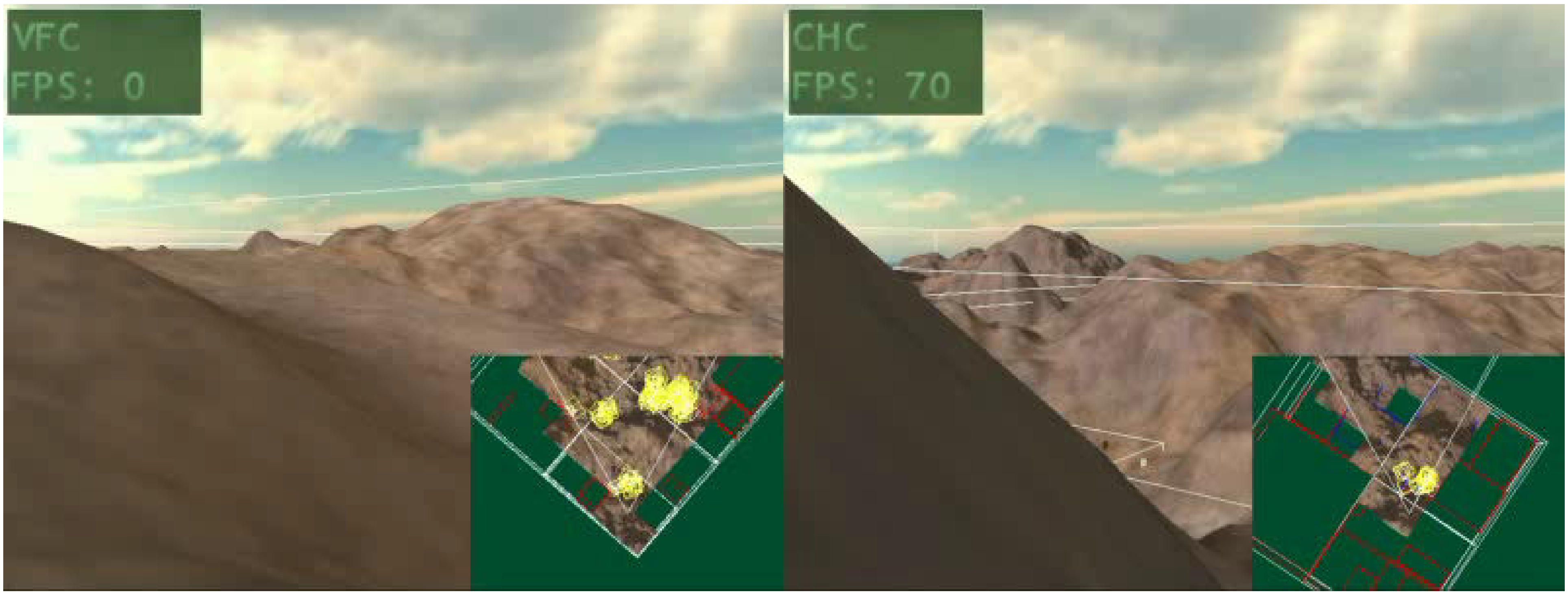


Rx Render object x

Qx Query object x

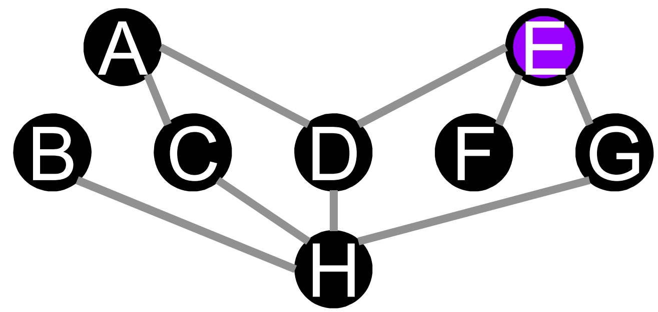
Cx Cull object x

Video: VFC vs. CHC



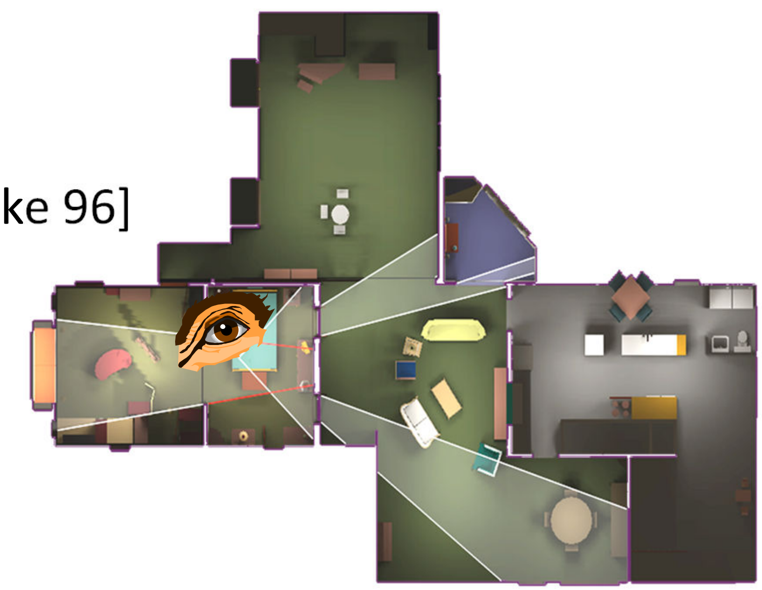
Interiors: Cells and Portals

- Partition the scene in cells and portals
  - Cells ~rooms
  - Portals ~ doors & windows



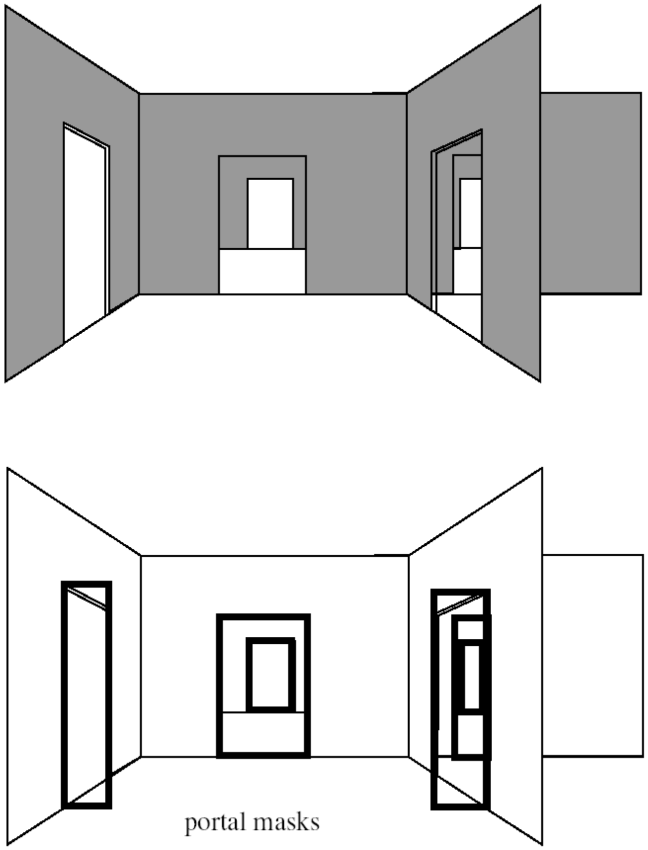
- Cell adjacency graph

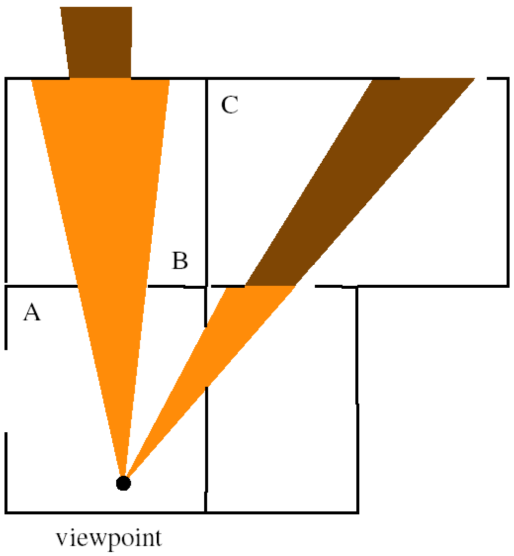
- Constrained search
  - Portal visibility test [Luebke 96]



Portal Visibility Test

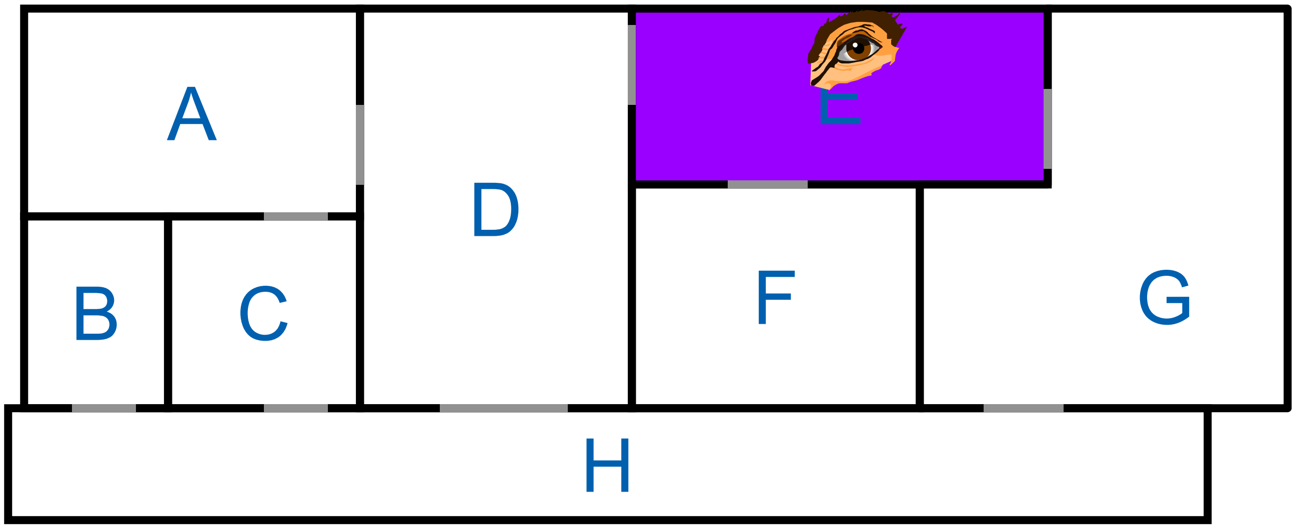
- Intersection of bounding rectangles of portals

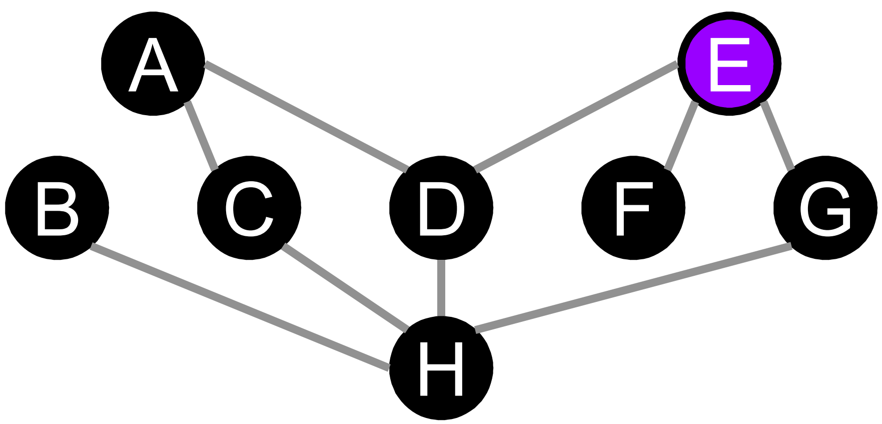




Cells and Portals Example

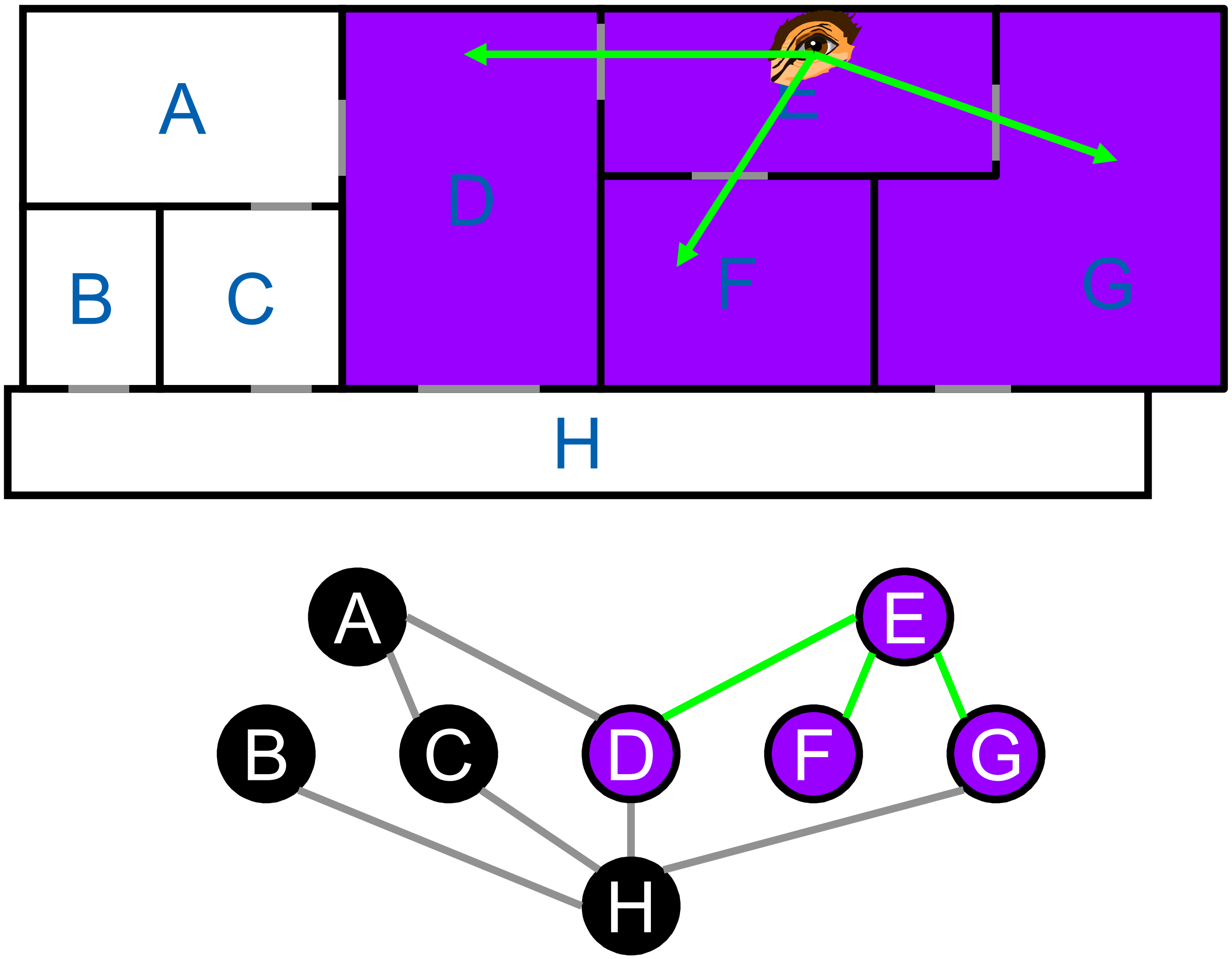
- Viewpoint in cell E





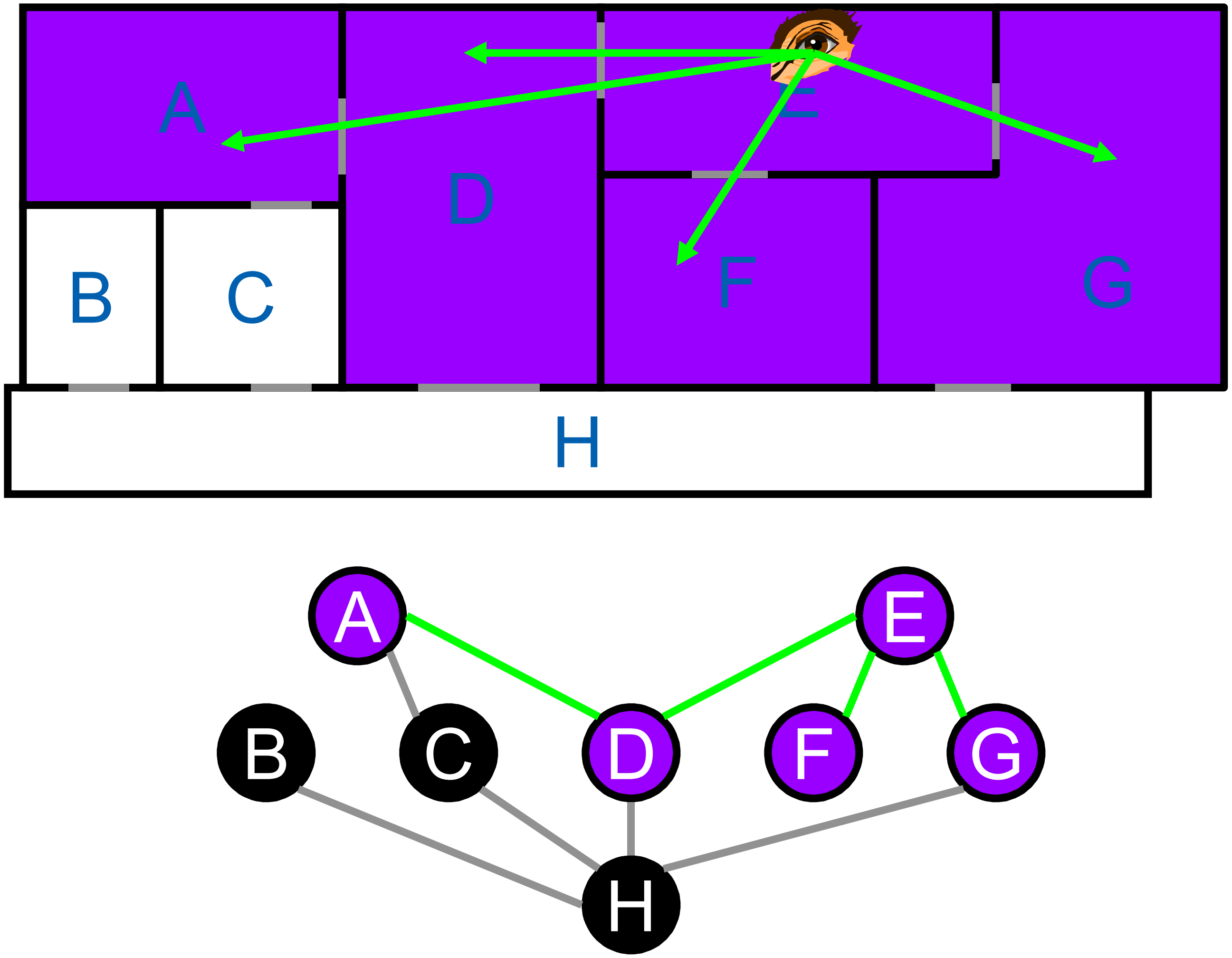
Cells and Portals - Example

- Adjacent cells DFG



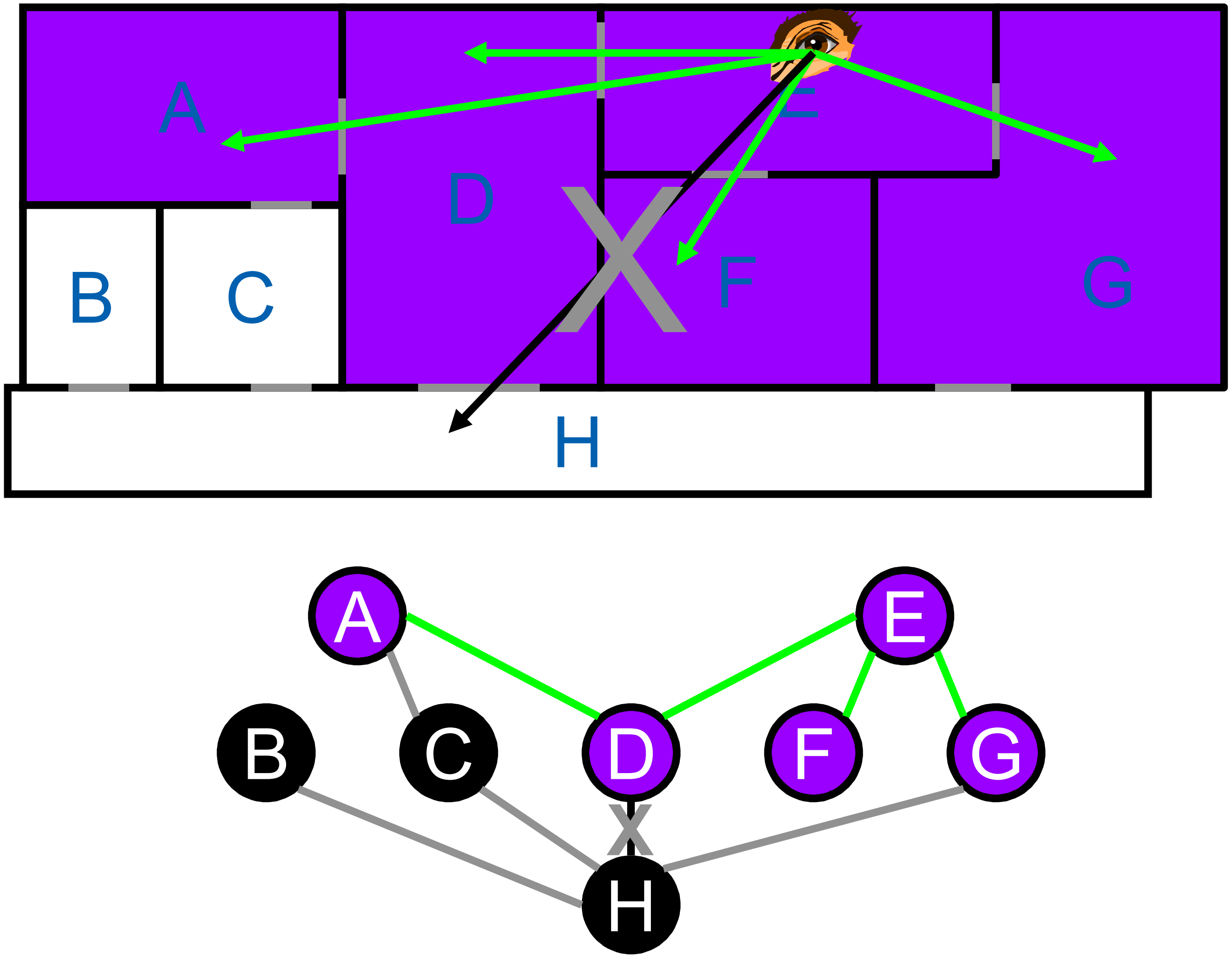
Cells and Portals - Example

- Cell A visible through portals E/D+D/A



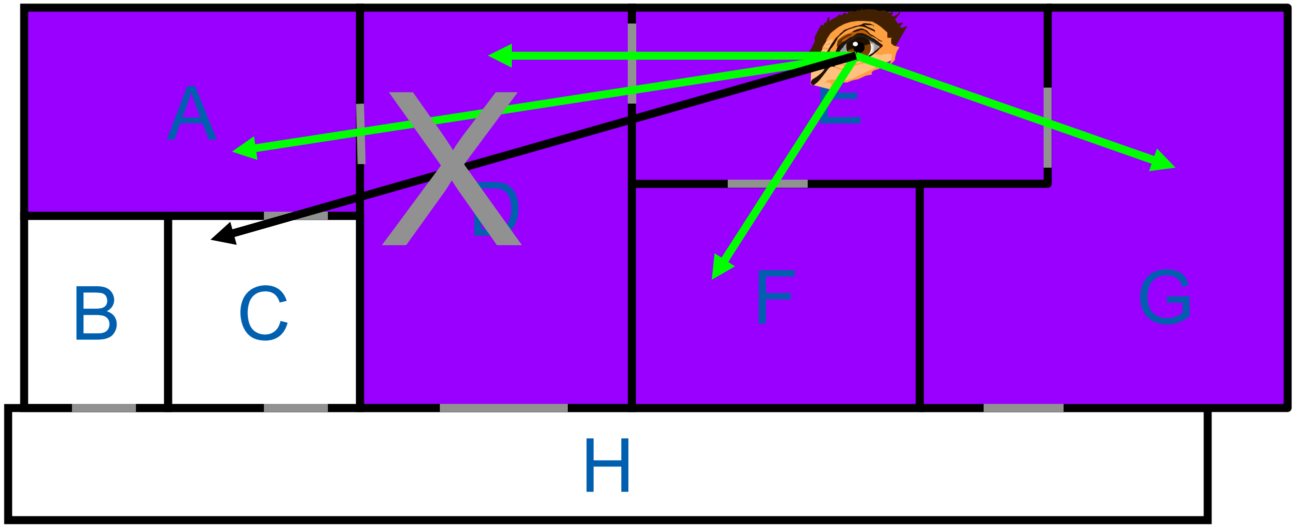
Cells and Portals - Example

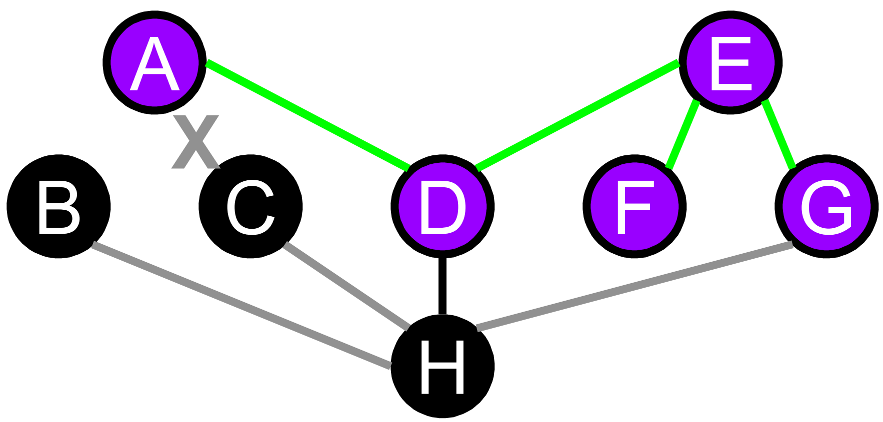
- Cell H not visible through portals E/D+D/H



Cells and Portals - Example

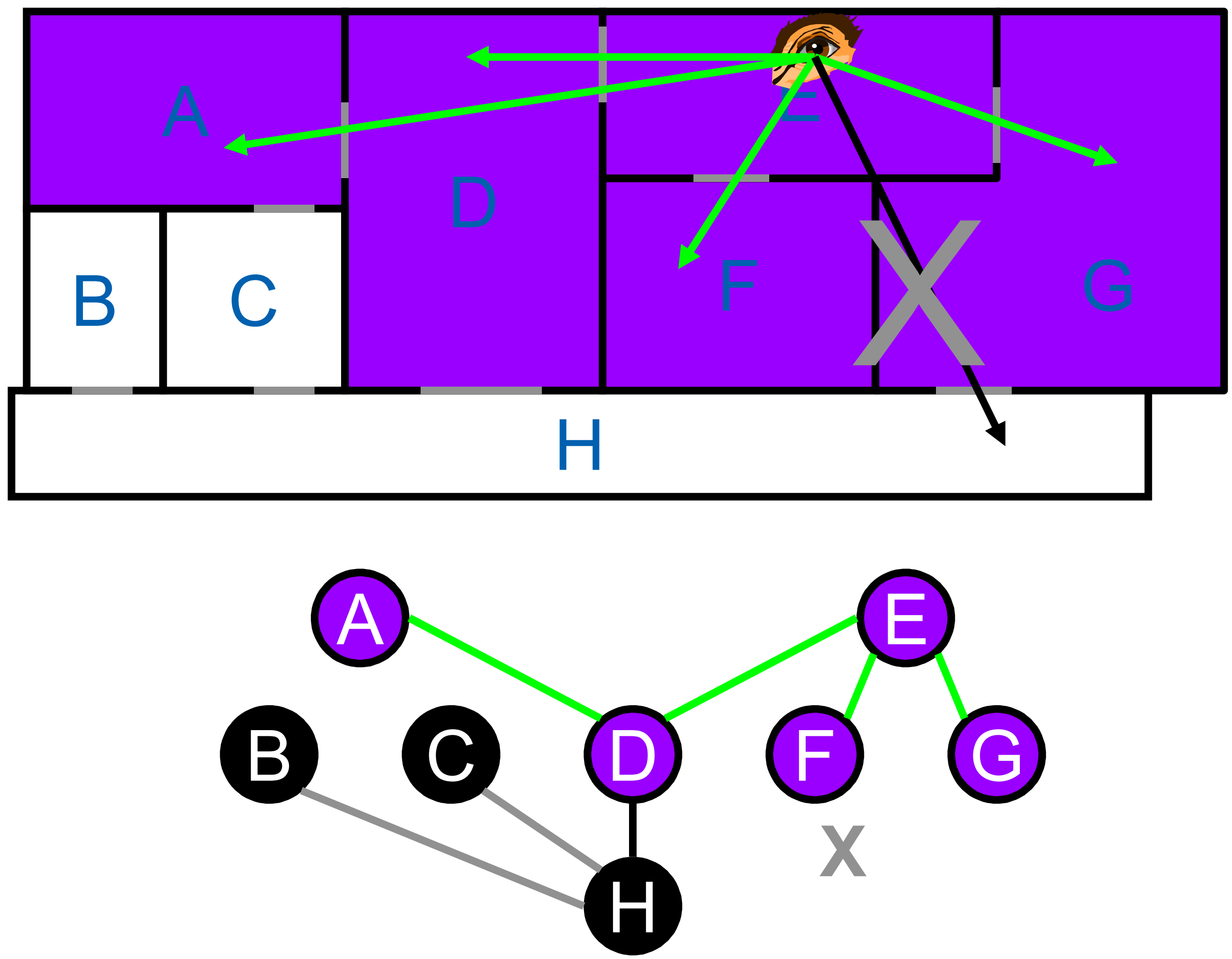
- C not visible through portals E/D+D/A+A/C





Cells and Portals - Example

- H not visible through portals E/G+G/H



Visibility Preprocessing

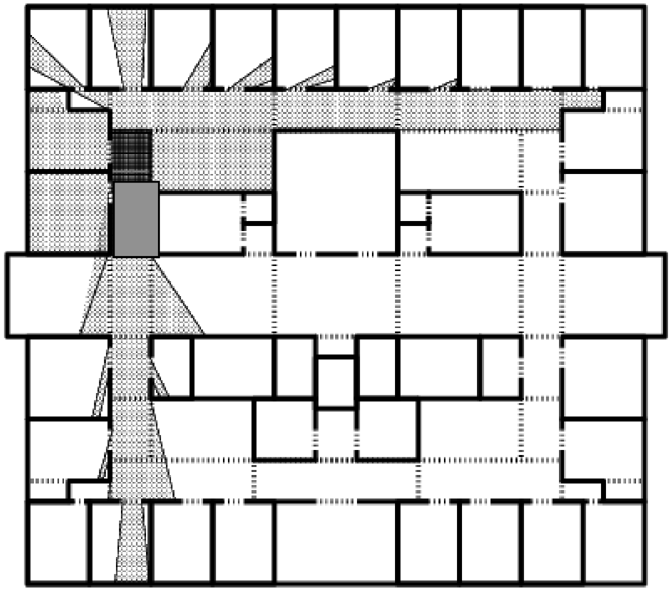
- Preprocessing
  - Subdivide view space into view cells
  - Compute Potentially Visible Sets (PVS)
  - Solves visibility “offline” for all possible view points
- Usage
  - Find the view cell (point location)
  - Render the associated PVS

Visibility Preprocessing

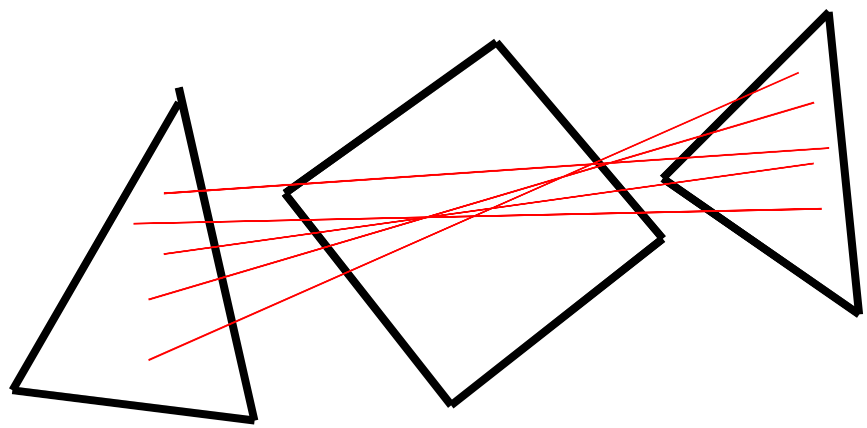
- Other benefits
  - Prefetching for out-of-core/network walkthroughs
  - Communication in multi-user environments
- Problems
  - Costly computation (treats all view points and view directions)
  - PVS storage

Interiors – Cells and Portals

- Subdivide the scene into cells and portals
- Constrained DFS on the adjacency graph
  - Portal visibility test
- More complex than the online algorithm
  - We do not have a view point!

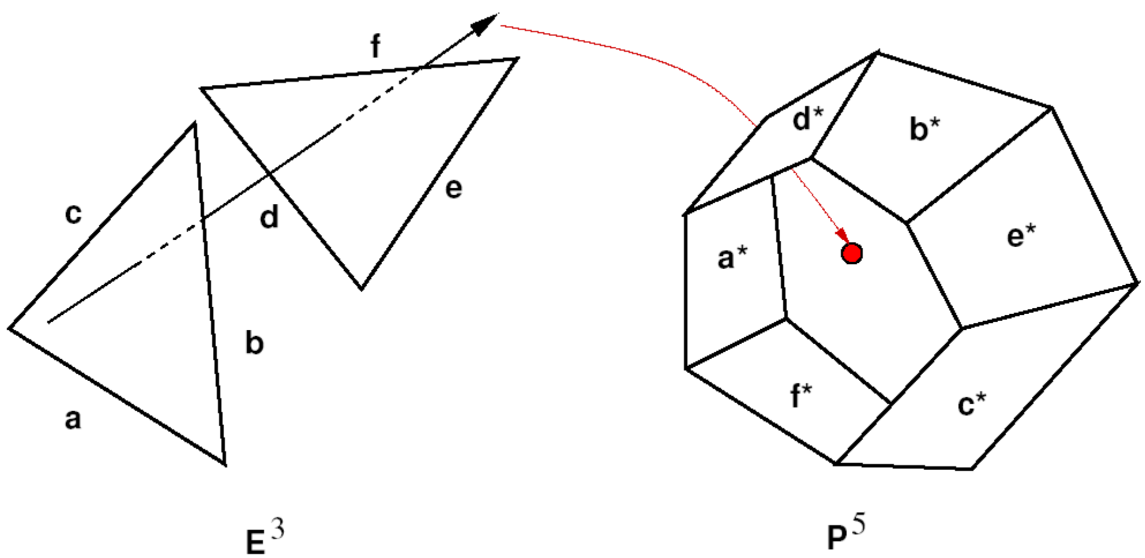


Interiors – Cells and Portals

- Sampling [Airey90]
  - Random rays
  - Non-occluded ray → terminate
- 
- + Simple implementation
- - Approximate solution

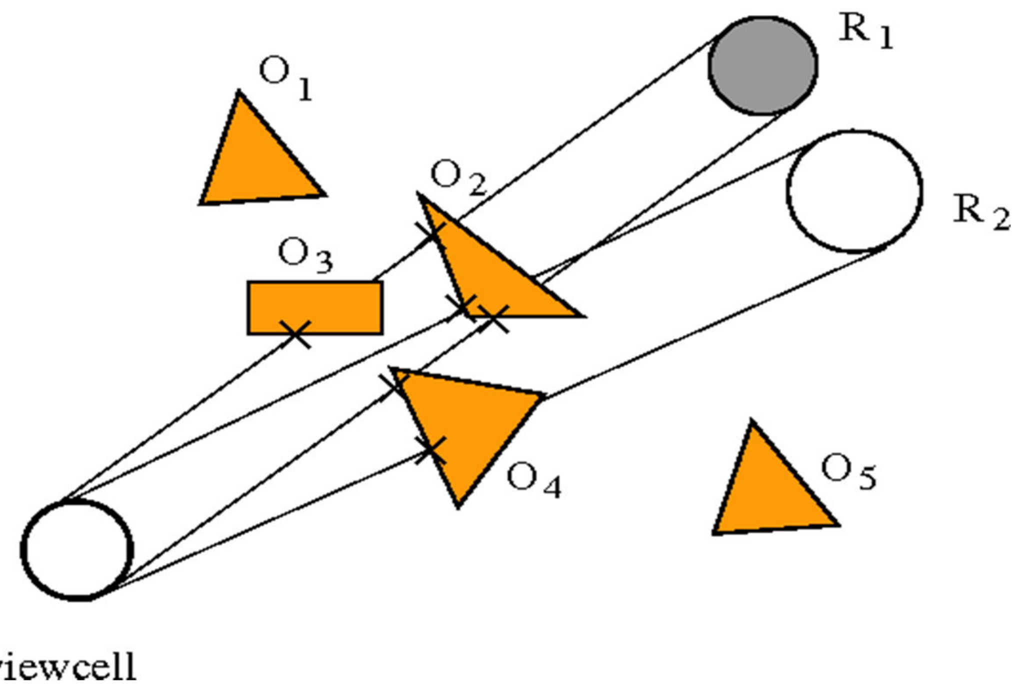
Interiors – Cells and Portals

- Exact computation [Teller 92]
  - Mapping to 5D (Plücker coordinates of lines)
- Portal edges → hyperplanes Hi in 5D
- Halfspace intersection in 5D



General Scenes - Strong Occlusion

- Occlusion by single object [CohenOr98]
- For each cell and object
  - Cast rays defining convex hull of the cell and object
  - If a convex occluder intersects all rays → invisible



General Scenes - Strong Occlusion

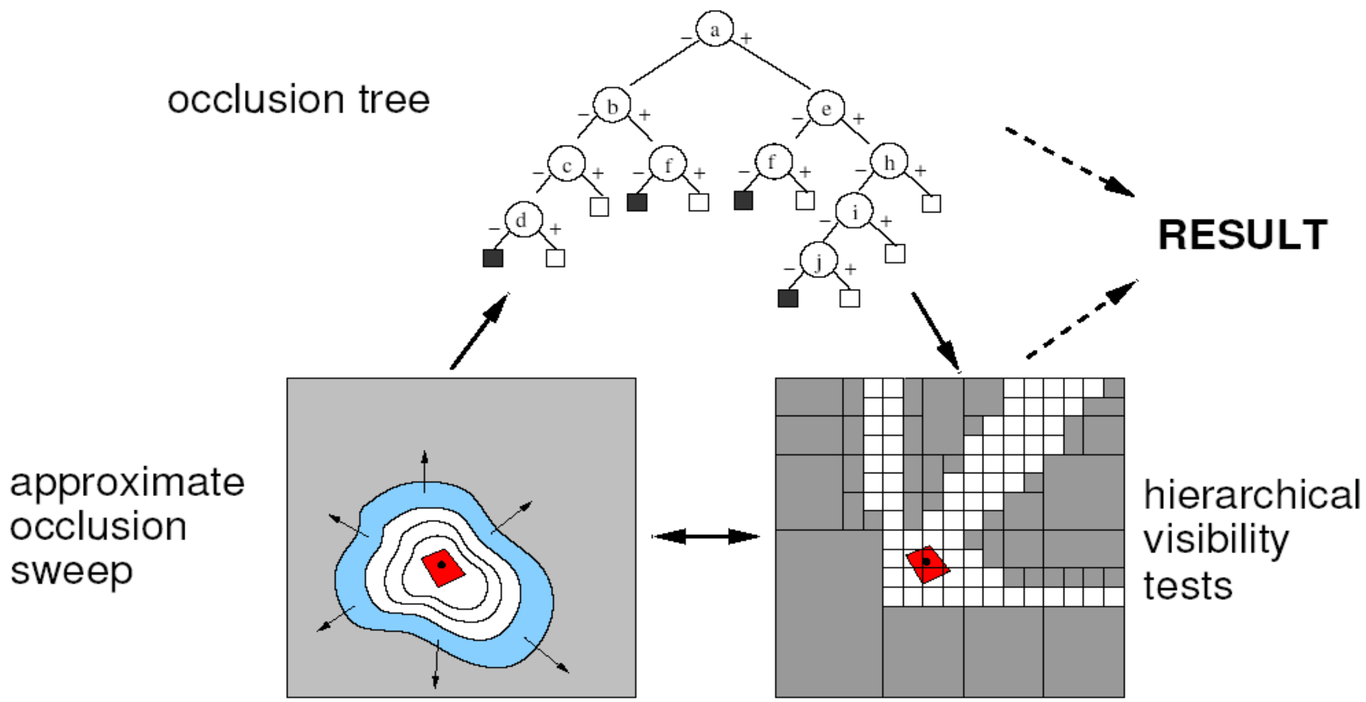
- Properties
  - + Simple
  - - No occluder fusion (no occluder sorting)
  - - Too conservative for larger view cells and small objects

General Scenes: Occlusion Tree

- Extension of the 2D occlusion tree
- 5D BSP tree
  - Plücker coordinates of lines
- The tree represents union of occluded rays

General Scenes: Occlusion Tree

- Process polygons in front-to-back order
- Polygon visible → enlarge tree by visible rays
- Polygon invisible → tree not modified

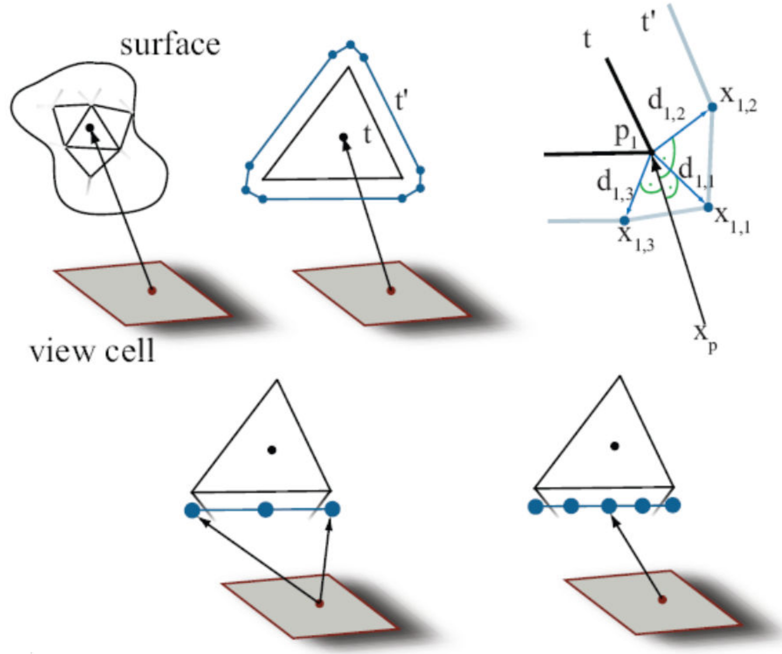


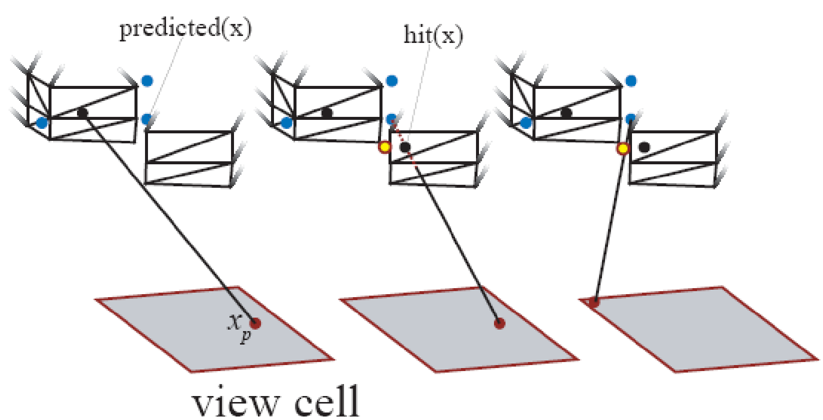
General Scenes: Occlusion Tree

- Properties
  - + Exact solution
  - + Uses visibility coherence
  - Difficult implementation
-

Guided Visibility Sampling (GVS)

- Stochastic + deterministic sampling
- Precision comparable with exact methods
- Useful for very large scenes
- Ideas
  - Random “seeding” ray
  - Adaptive border sampling (mutating termination point)
  - Reverse sampling (mutating ray origin)





Rendering Massive Models – Optimizations

- Manual model optimization
  - Textures, bump maps, normal maps
- Optimal GPU utility
  - Triangle strips, vertex arrays, vertex buffer objects, optimized vertex and pixel shaders, minimize state changes
- Automatic model optimization
  - LODs, bilboards, depth impostors, point sampling, …
- Data management
  - Data prefetching, data layout (out-of-core), using coherence
- Visibility culling
  - Online culling, preprocessing

Hierarchical Levels of Detail (HLOD)

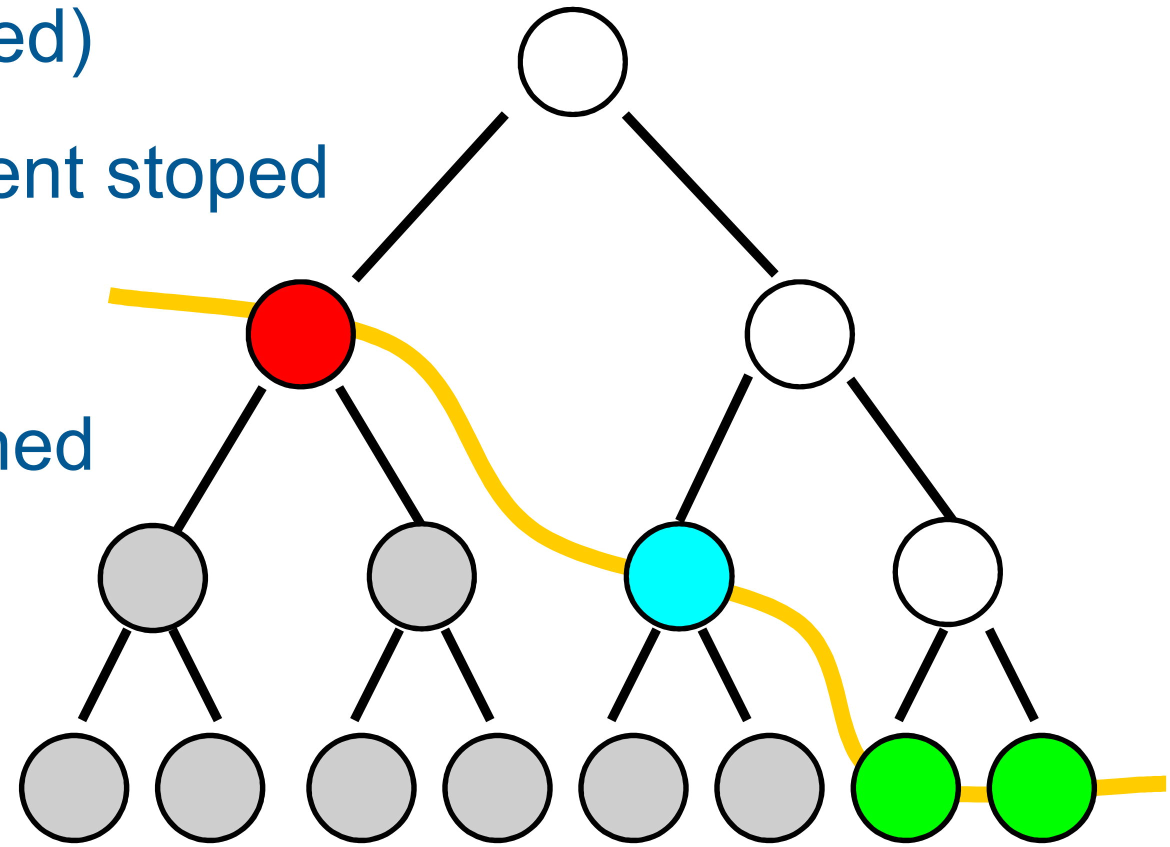
- LOD with each node of the hierarchy (2001)

Invisible (culled)

LOD refinement stoped

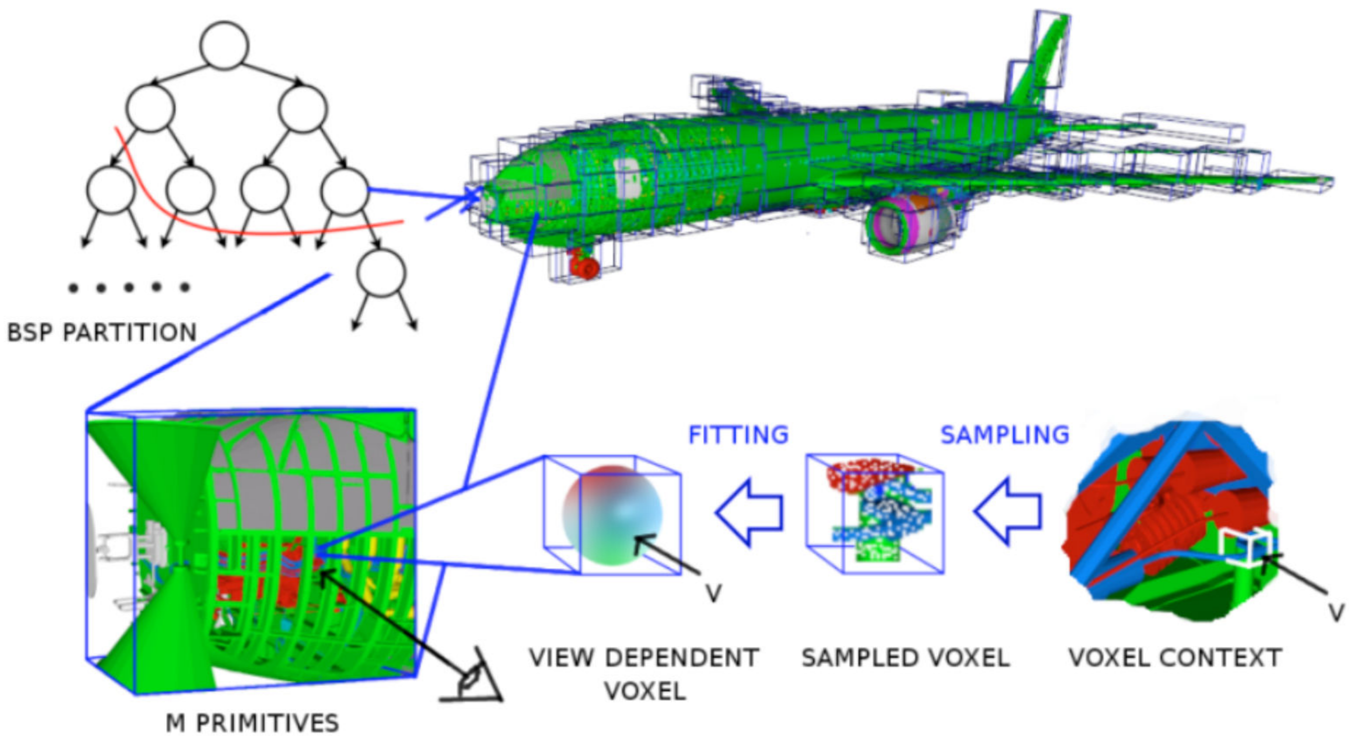
Visible leaf

Visible & refined



Far Voxels

- Gobetti & Marton 2005
- Approximate "far" geometry with view dependent voxel, original dataset 300 million triangles.



Courtesy of E. Gobetti (SIGGRAPH '06 course – Massive Model Visualization)

Visibility Culling - Summary

- Find visible objects for a view point or view cell
- Online Visibility Culling
  - +Dynamic scenes
  - +Simple implementation
  - -Every frame
  - -No global information
- Visibility preprocessing
  - -Static scenes
  - -Complicated implementation
  - +No overhead at runtime
  - +Global information

Surveys on Visibility

- F. Durand. 3D Visibility: Analytical Study and Applications, 1999.
- D. Cohen-Or et al.: A survey of visibility for walkthrough applications, 2003.
- J. Bittner and P. Wonka: Visibility in computer graphics, 2003.

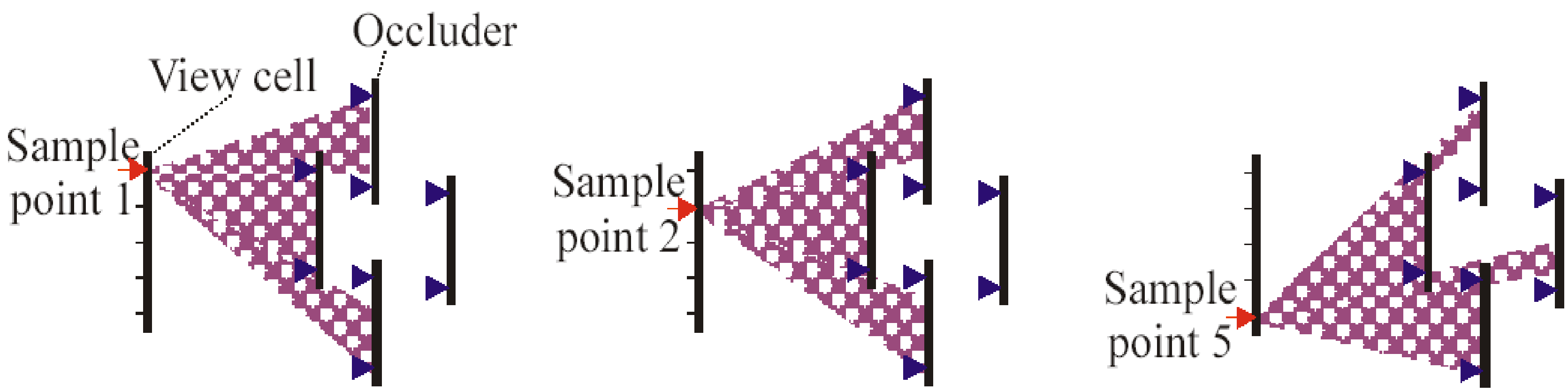
… end of this part

bittner@fel.cvut.cz

DCGI

2.5D Scenes Occluder Shadows

- Footprint of occluded volume [Wonka00]
  - Agrregates the shadow polygons using z-buffer
  - Represents intersection of all 'shadows'

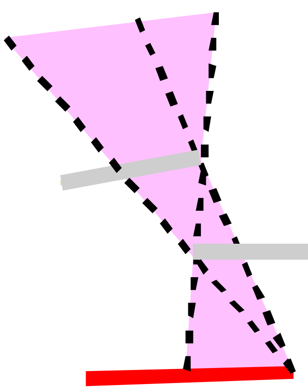


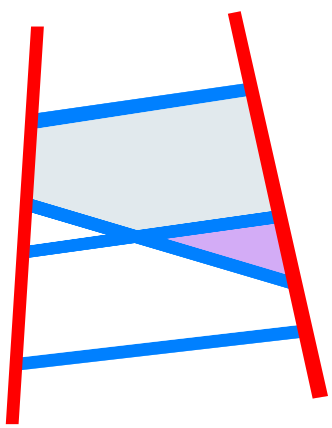
2.5D Scenes Occluder Shadows

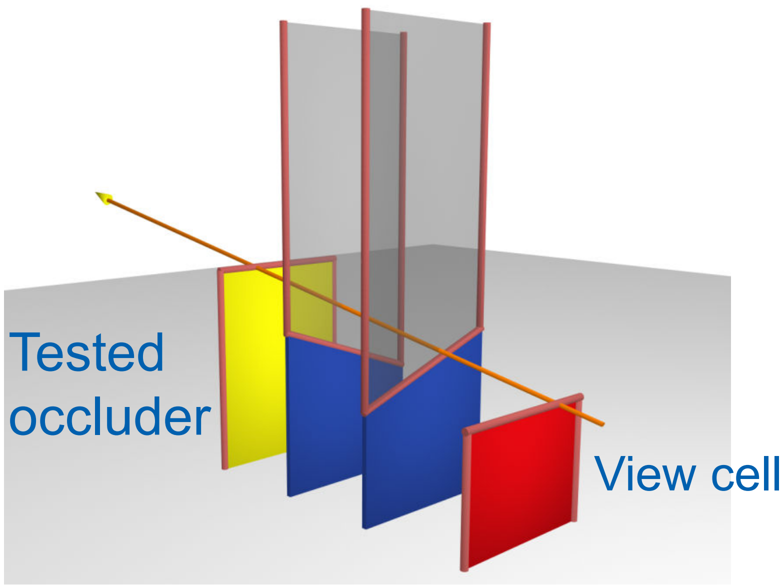
- Conservative solution
  - Shrinking occluder polygons
- Properties
  - + Relatively easy implementation
  - + Uses GPU
  - - Large view cells $\rightarrow$ more conservative solution
  - - Needs high resolution cull map

2.5D Scenes Occlusion Tree + Virtual Portals

- Occlusion tree for visibility in 2D footprint
- Identifies sequences of occluders
- Construct virtual portals over the occluders
- Portal visibility test in 5D [Teller 92]







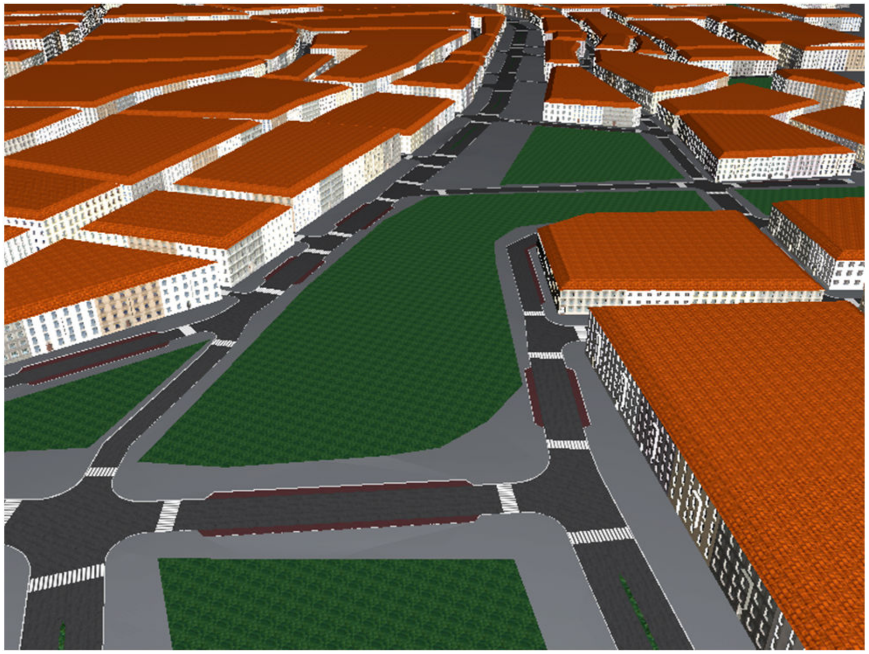
2.5D Scenes: Occlusion Tree + Virtual Portals

- Properties
  - + exact solution for 2.5D scenes
  - + computation time comparable with conservative methods
  - - difficult implementation

Problems of CHC

- Too many queries
- Not really GPU friendly
  - Many state changes
  - Bounding box query (8 vertices per draw call)
- Can be slower (!) than view frustum culling (VFC)

Most houses visible →
Bad view point for CHC



Near Optimal Hiearchical Culling

- Guthe et al. 2006
  - Query only if cheaper than rendering
  - Mostly better than view frustum culling
  - Close to self-defined optimum
  - Hardware calibration step
  - Complex set of rules

- Possible to beat the defined “optimum”
  - Can reduce cost of queries
  - Can further reduce # queries

CHC ++

- Mattausch et al. 2008
- Keep simplicity of CHC
- Reduction of
  - State changes
  - Queries
  - Wait time
  - Rendered geometry
- Game engine friendly

Building Blocks of CHC ++

- Query batching
  - Reduction of state changes
  - Reduction of CPU stalls
- Multiqueries
  - Reduction of queries
- Randomization
  - Better distribution of queries
- Tight bounding volumes
  - Reduction of queries
  - Reduction of rendered geometry
- Render queue
  - Interface to the game engine

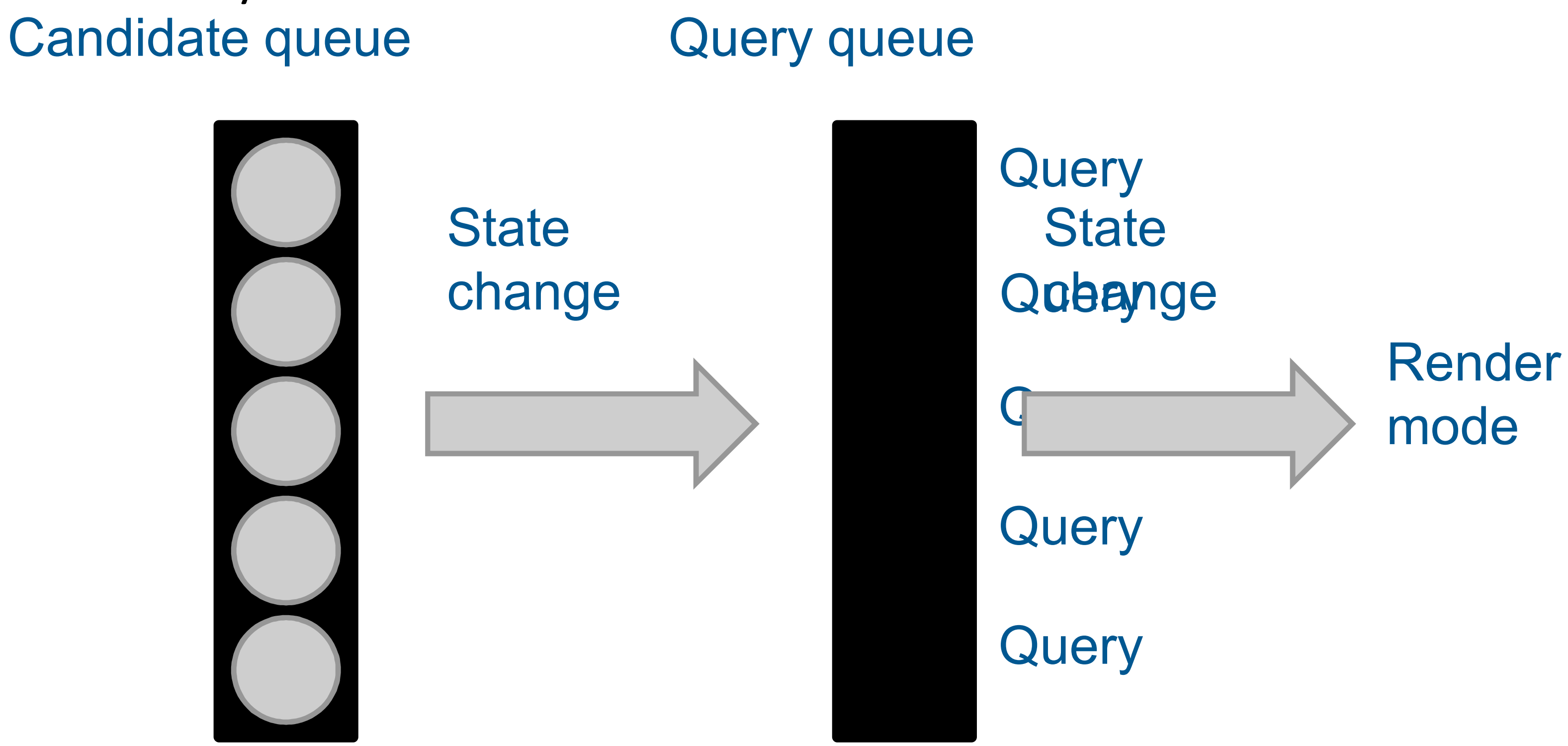
Query Batching: State Changes

- Switch between render / query mode
  → Need state change (depth write on / off)
- CHC induces one state change per query
- Big overhead on modern GPUs!

m2 show image of engine block
matt, 5/13/2007

Query Batching: Previously invisible nodes

- Idea: Store query candidates in separate queue
  - Collect n nodes
  - Switch to query mode
  - Query all nodes

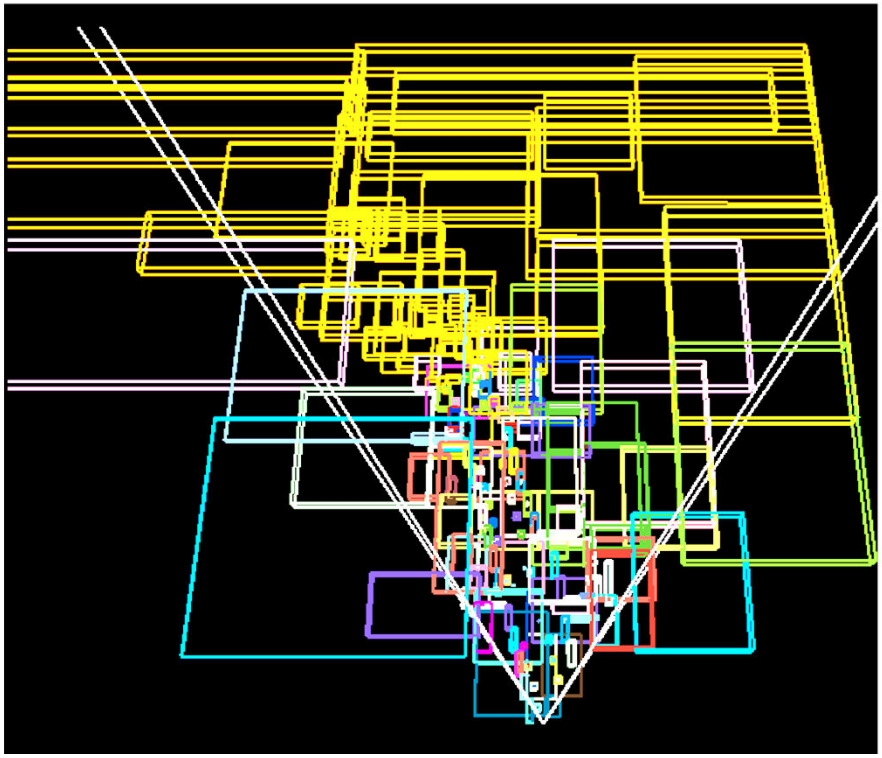


Query Batching: Previously visible nodes

- Previously visible nodes
  - No dependencies (geometry rendered anyway)
  - Can issue query at any time
  - Handle them in separate queue
- Issue queries to fill up wait time
  - Very likely no new state change
- Issue the rest of queries in the end of the frame

Query Batching: Visualization

Each color represents a state change

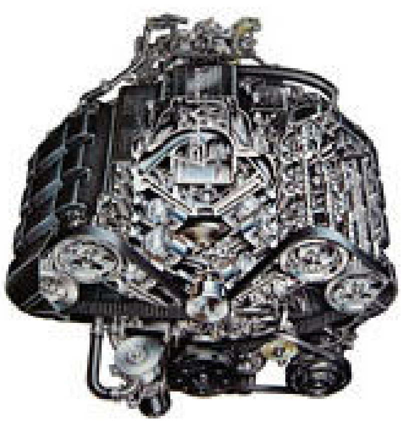


CHC: ~100 state changes

CHC++: 2 state changes
(Max. batch size: 50)

Multiqueries: Idea

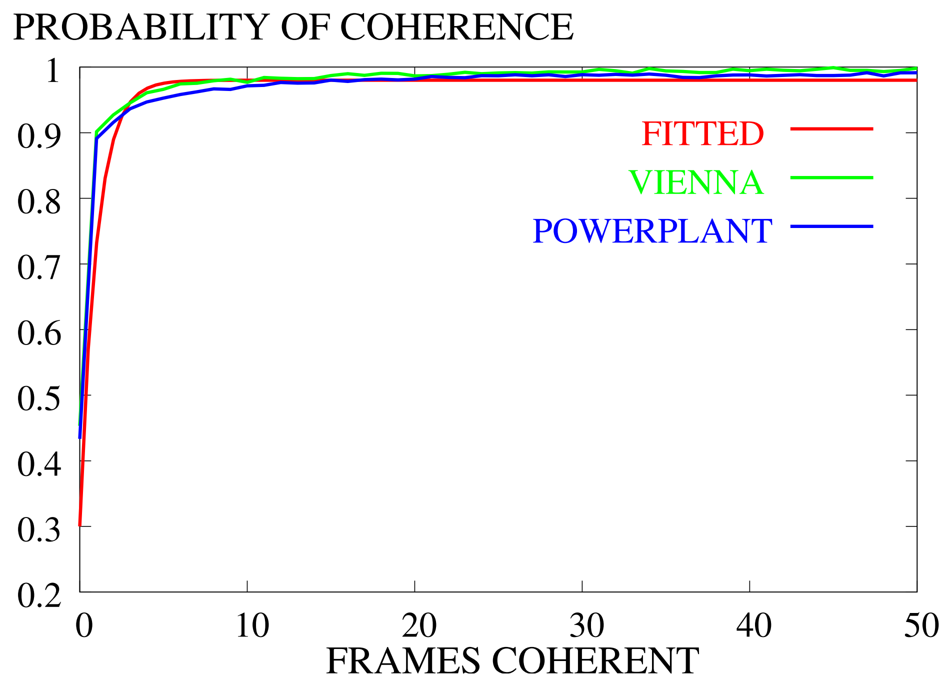
- Node invisible for long time
  - Likely to stay invisible (e.g., car engine block)
  - Cover many nodes with single query
- Test q invisible nodes by single multiquery
  - Invisible → saved (q – 1) queries
  - Visible → must test individually, wasted one query



Multiqueries : Minimize #queries

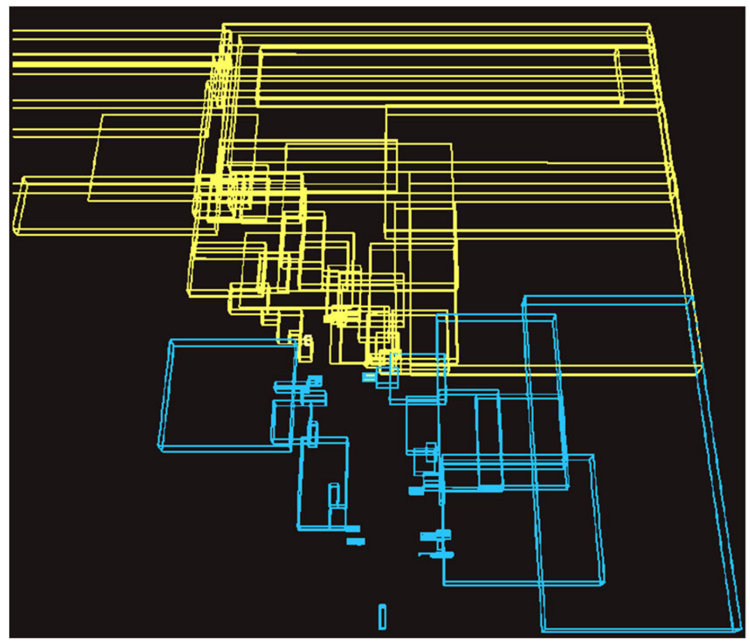
- Use history of nodes
- Estimate probability that node will still be invisible in frame n+1 if it was invisible in frames ≤ n
- Measurements behave like 1-k*e-x function → sufficient in practice

Fitted and measured functions



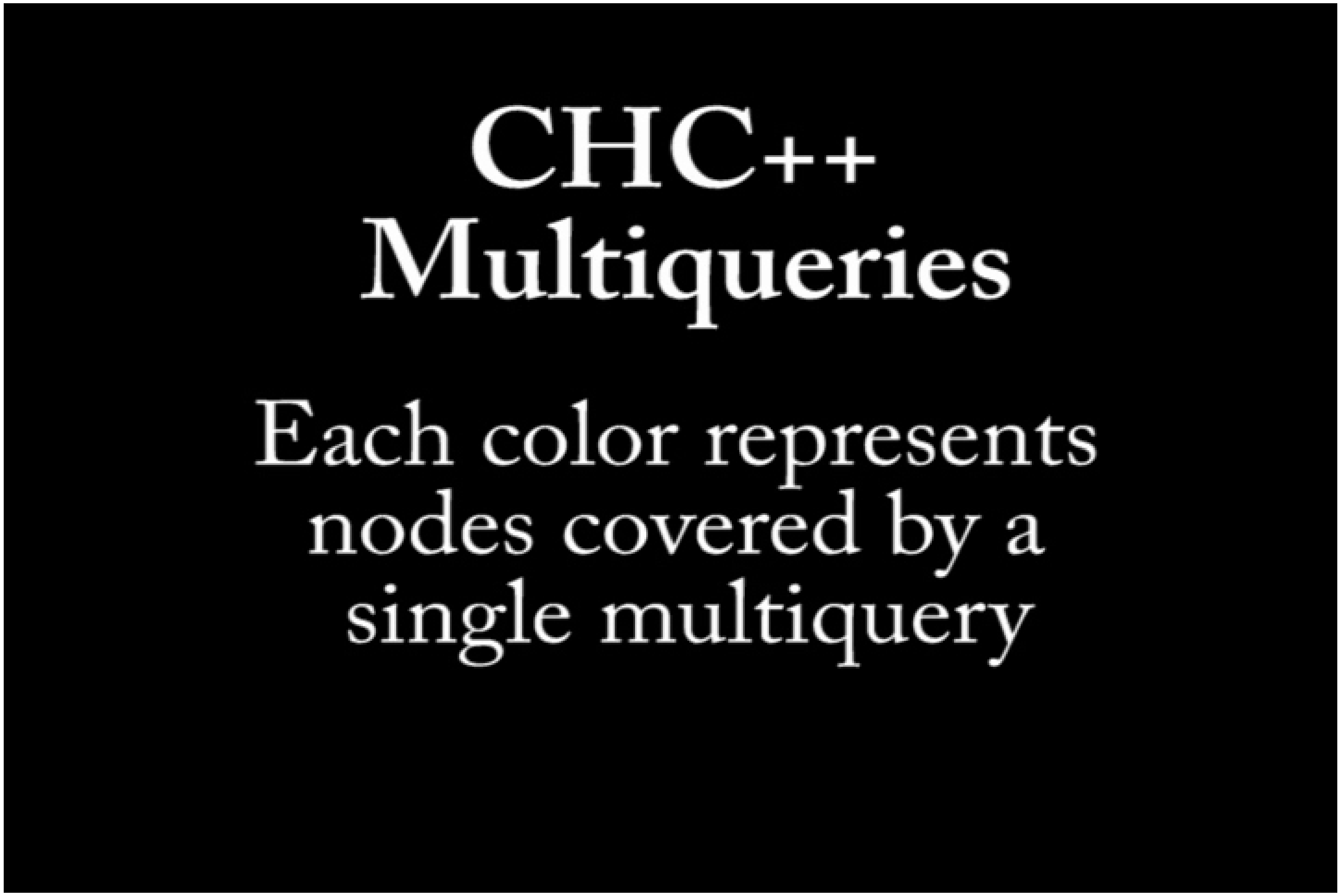
Multiqueries: Greedy Optimization

- While node batch not empty
  - Add node to multiquery
  - Use cost / benefit model
    - Query size optimal → issue multiquery



Visualization: Each color represents a multiquery

Multiqueries: Vienna (1M triangles)

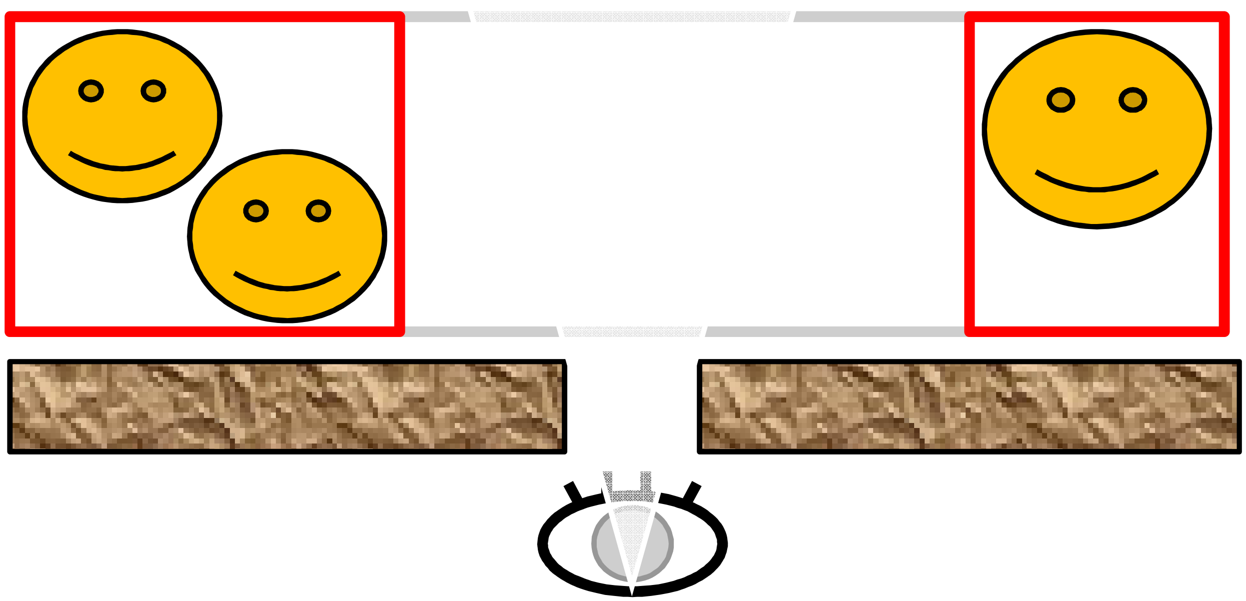


Tight Bounding Volumes: Idea

- Optimization for bounding volume hierarchy (BVH)
- For each node → query bounding boxes of children (using single query)

Child boxes invisible → Cull node, saved 2 queries

Box visible → Traverse node

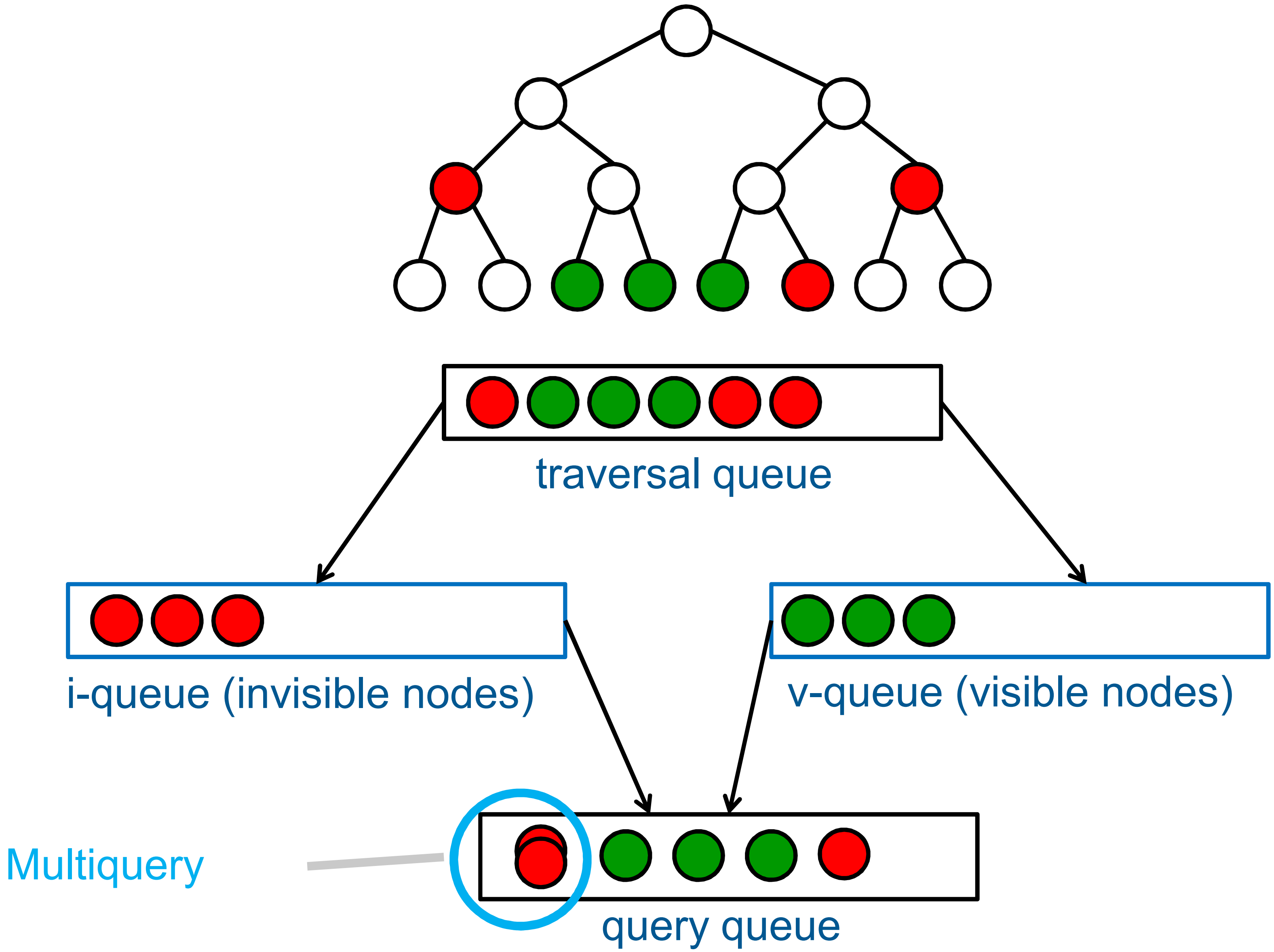


Game Engine Integration

- Modern game/rendering engines
  - Collect visible objects in render queue
  - Sort by materials
  - Render everything at once
- Rendering single nodes is inefficient (CHC)
- With batching:
  - Traverse render queue just before processing a query batch

m3

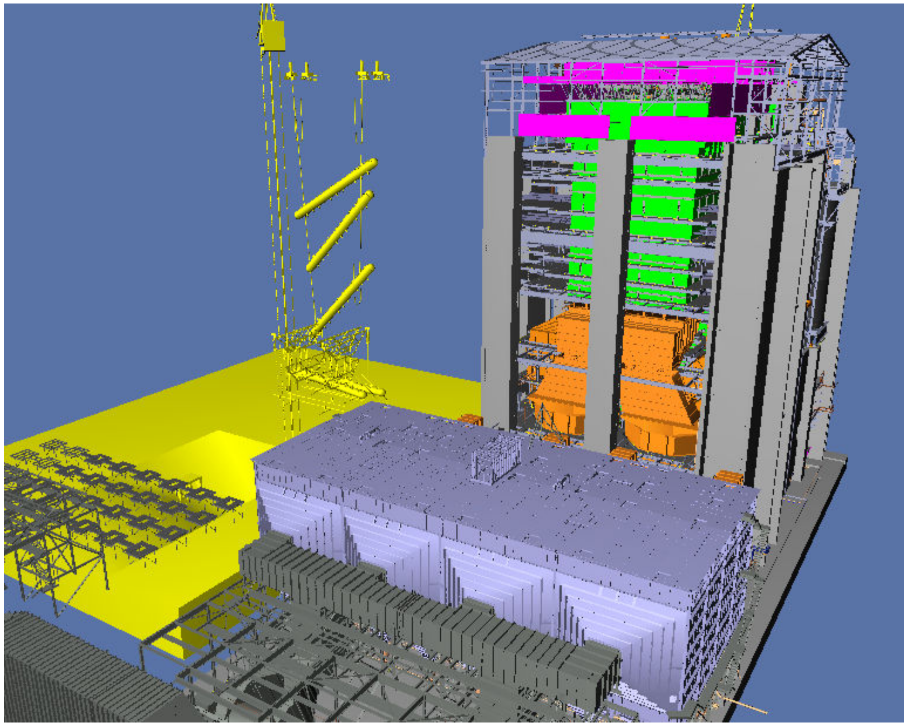
Queues in CHC++



m3 update picture
matt, 3/30/2008

m4

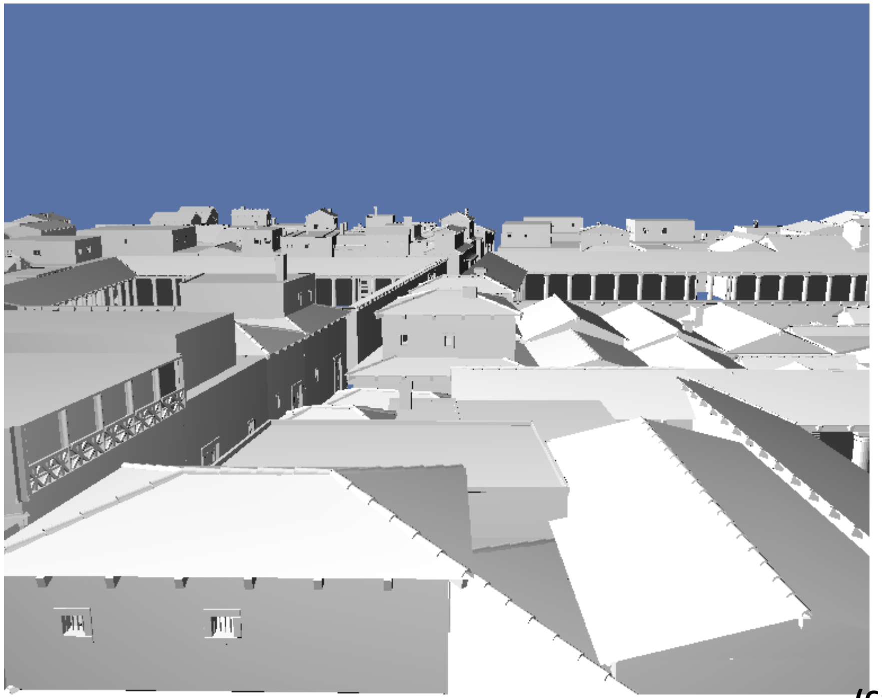
Results



Powerplant
(12M triangles)

m5

Pompeii
(6M triangles)



Snímek 68

---

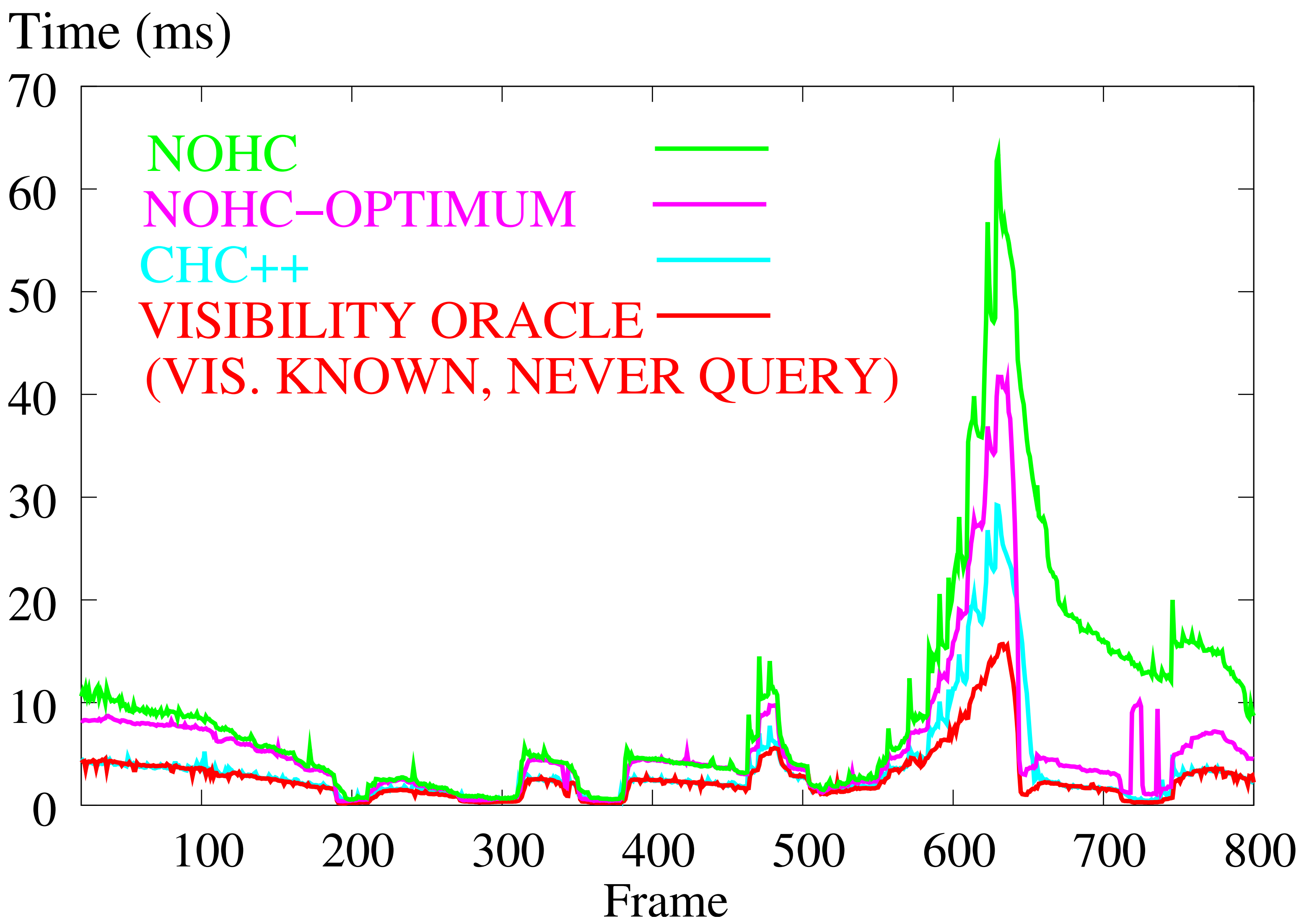
m4 show teaser: state change reduction or just graph!
only show vfc, chc and chc ++
matt, 3/6/2008

m5 show videos of state changes/ multiqueries
also show videos of walkthroughs?
matt, 3/30/2008

m6

Results: Pompeii (6M triangles)

m7



Snímek 69

m6 show teaser: state change reduction or just graph!
only show vfc, chc and chc ++
matt, 3/6/2008

m7 change yellow color, change caption
matt, 3/28/2008