Algorithmen für die Echtzeitgrafik

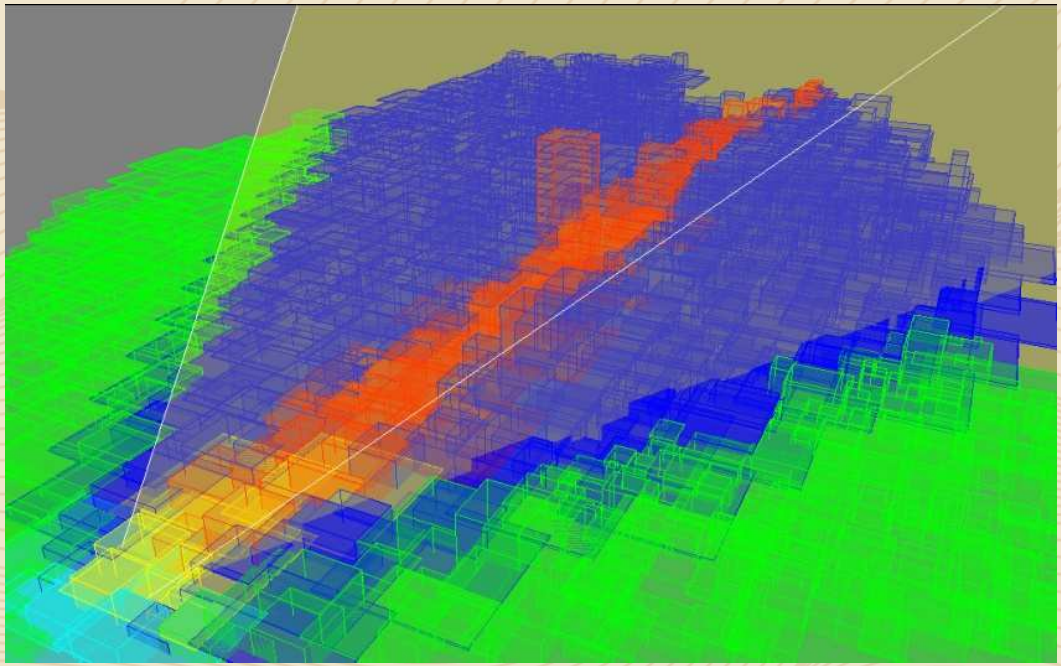
Algorithmen für die Echtzeitgrafik

Daniel Scherzer
scherzer@cg.tuwien.ac.at

LBI Virtual Archeology

Temporal Coherence

Object Space



Temporal Coherence in Object Space

- Incremental global illumination

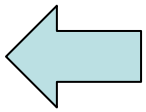
  less → we describe one

- Online occlusion culling

  naturally in object space → many!

Temporal Coherence in Object Space

- Incremental global illumination

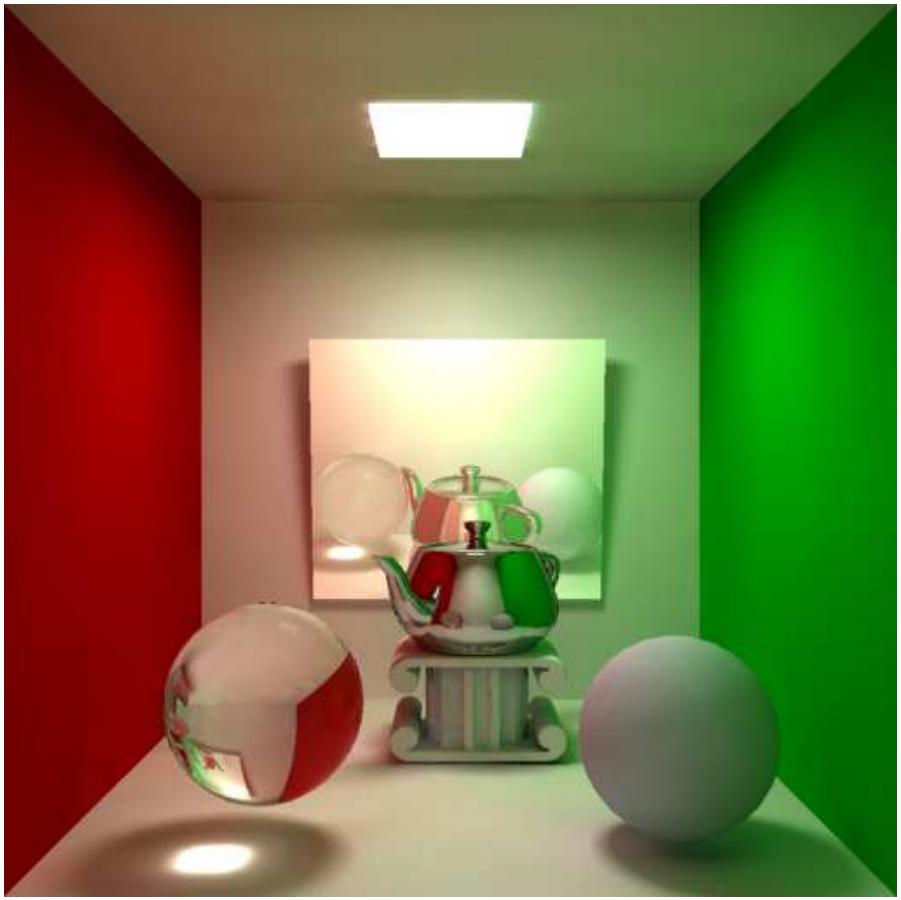


- Online occlusion culling

Global illumination: Motivation

direct illumination

direct + indirect illumination



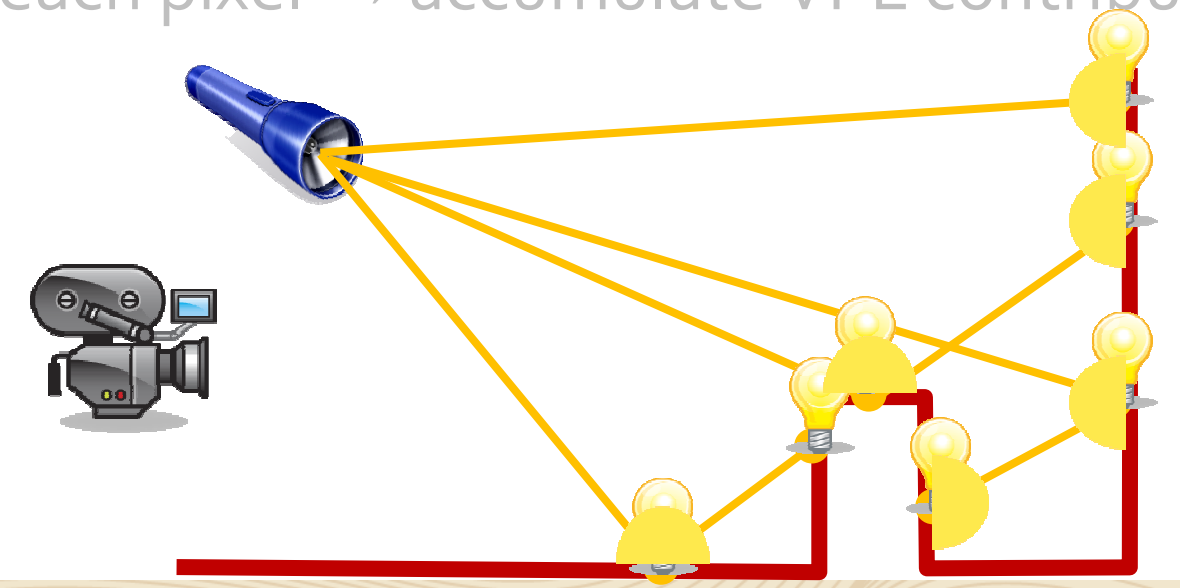
[image courtesy of Guerrero et al.]

Incremental Global illumination: Motivation

- Spectacular image quality with GI
- Real-time GI is cool, but hard to do …
- Preprocessing
  - Tedious
  - Dynamic local lights difficult
- → Reuse previous computations (with TC)!

Instant Radiosity [Keller97]

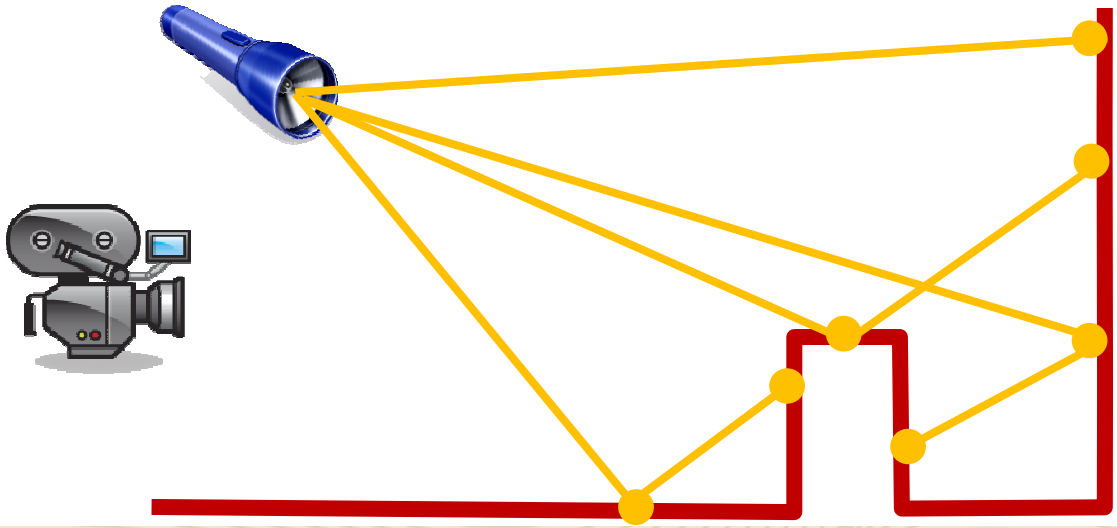
- Cast paths from light source
- At intersection with scene geometry
  - Assign virtual point lights (VPLs)
- Compute VPL visibility (→ shadow map) — costly!
- For each pixel → accumulate VPL contributions



Instant Radiosity

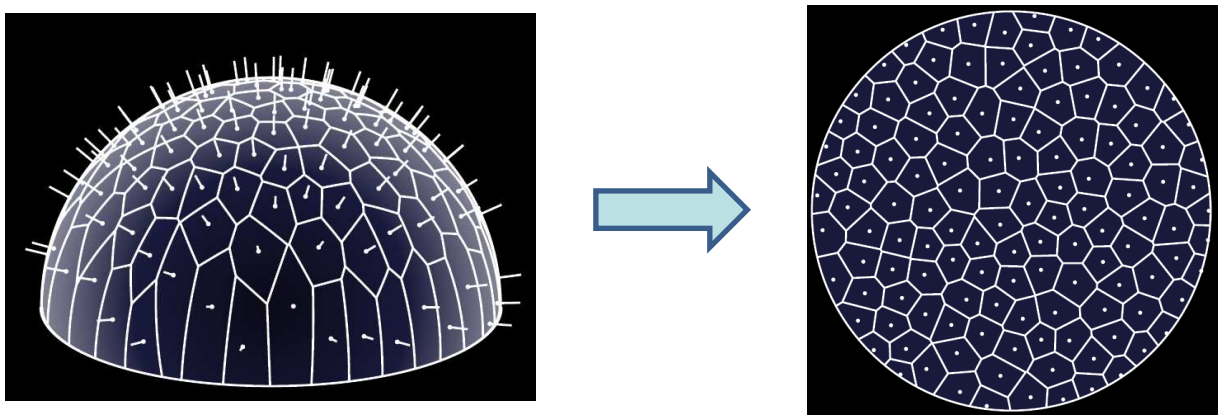
- Usually first bounce sufficient
- Still requires many shadow maps (≥ 256)
- Too much for real-time rendering!

→ reuse VPLs if possible!



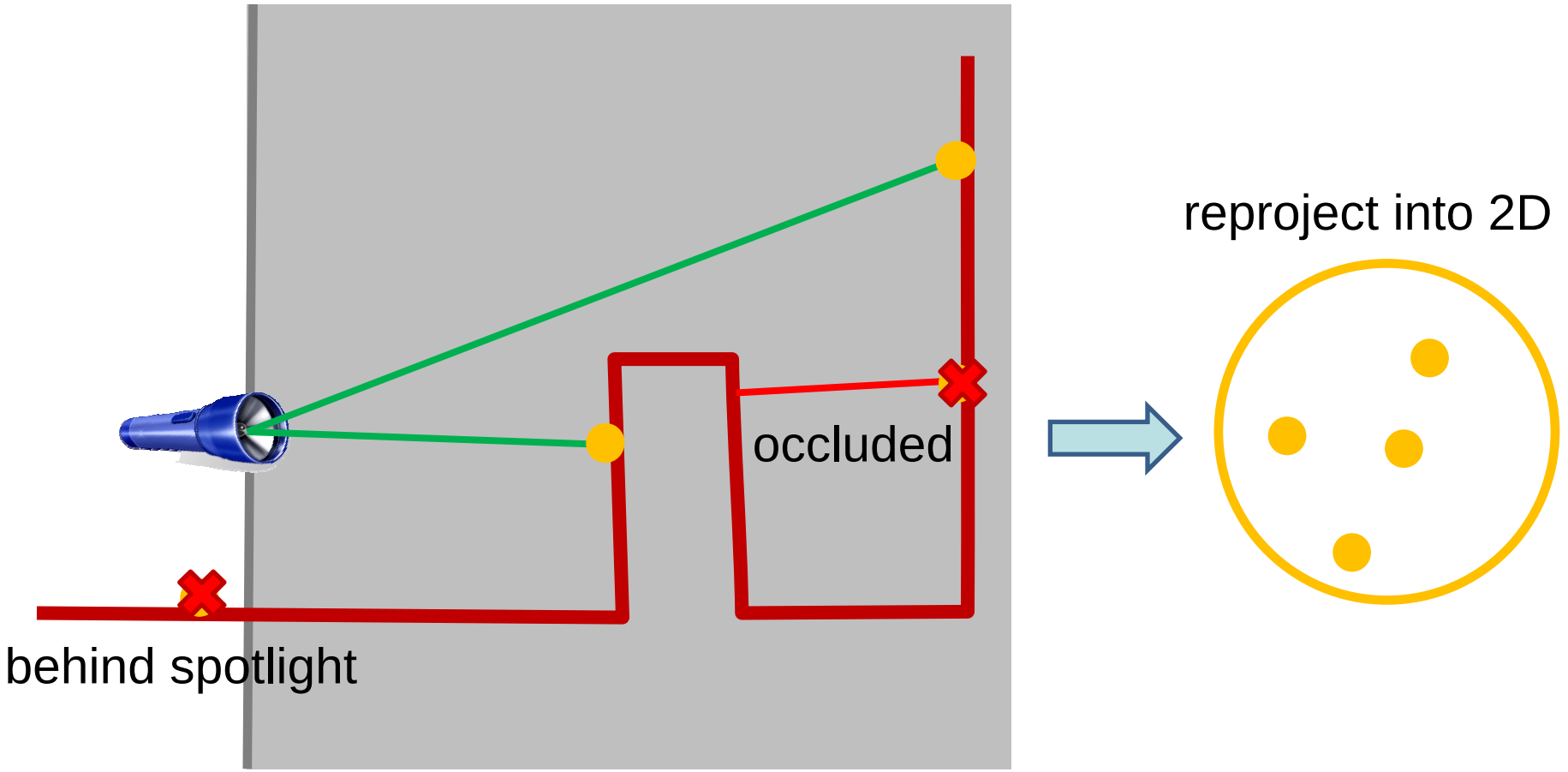
Incremental Instant Radiosity [Laine07]

- Recompute budget of VPLs (4 – 8)
- Reuse rest (if valid)
- Incremental VPL managment:
  - Delete invalid VPLs
  - Update VPL positions
    - Keep cosine-weighted VPL distribution (for 180° spotlight)
      → uniform distribution in 2D domain



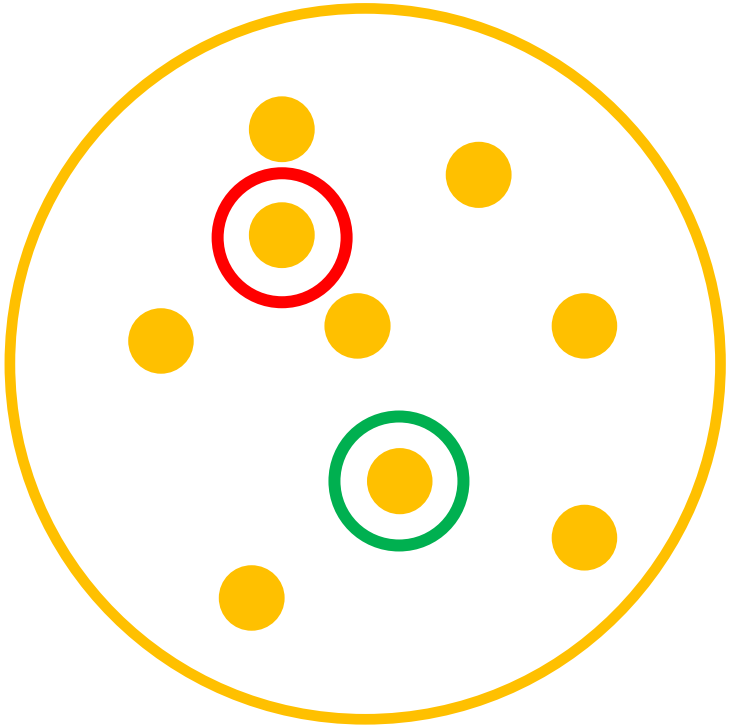
[image courtesy of Laine et al.]

Invalid VPLs



Update VPL positions

- Reproject VPLs to 2D domain
- Delete worst VPLs (use measure of uniformity)
- Find best position for new VPLs

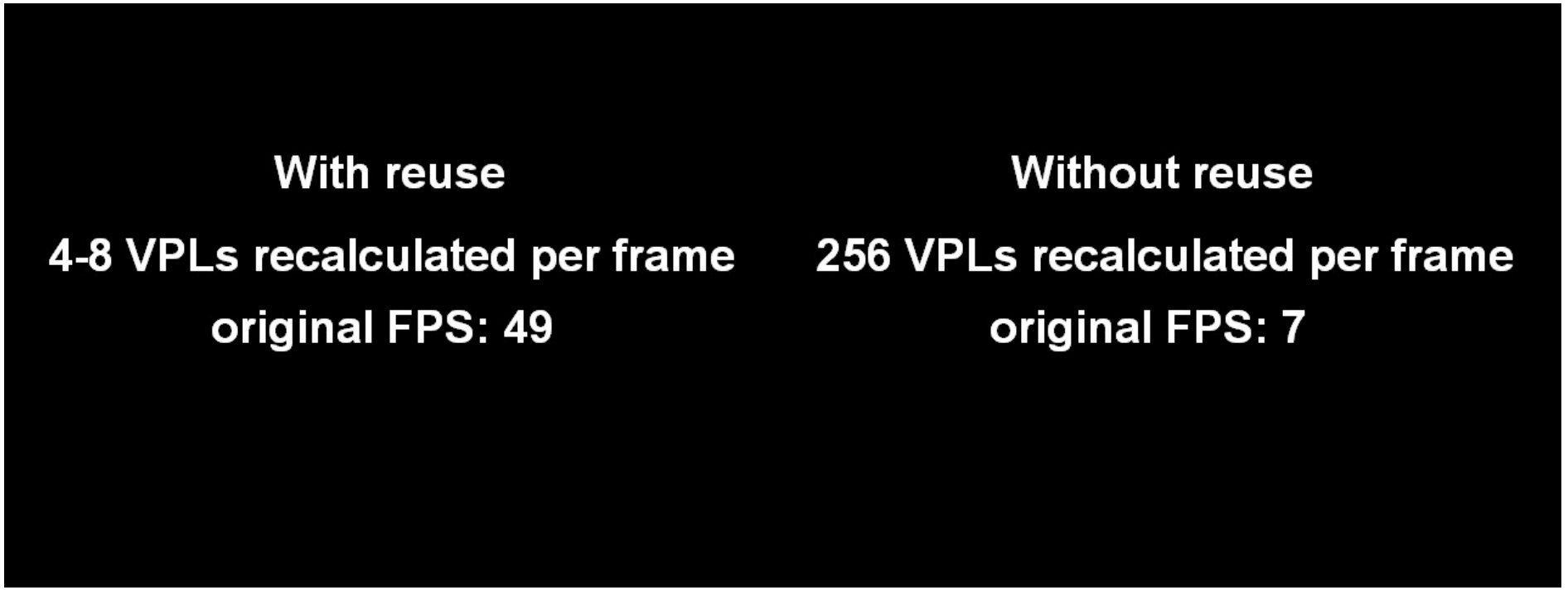


Incremental Instant Radiosity: Results

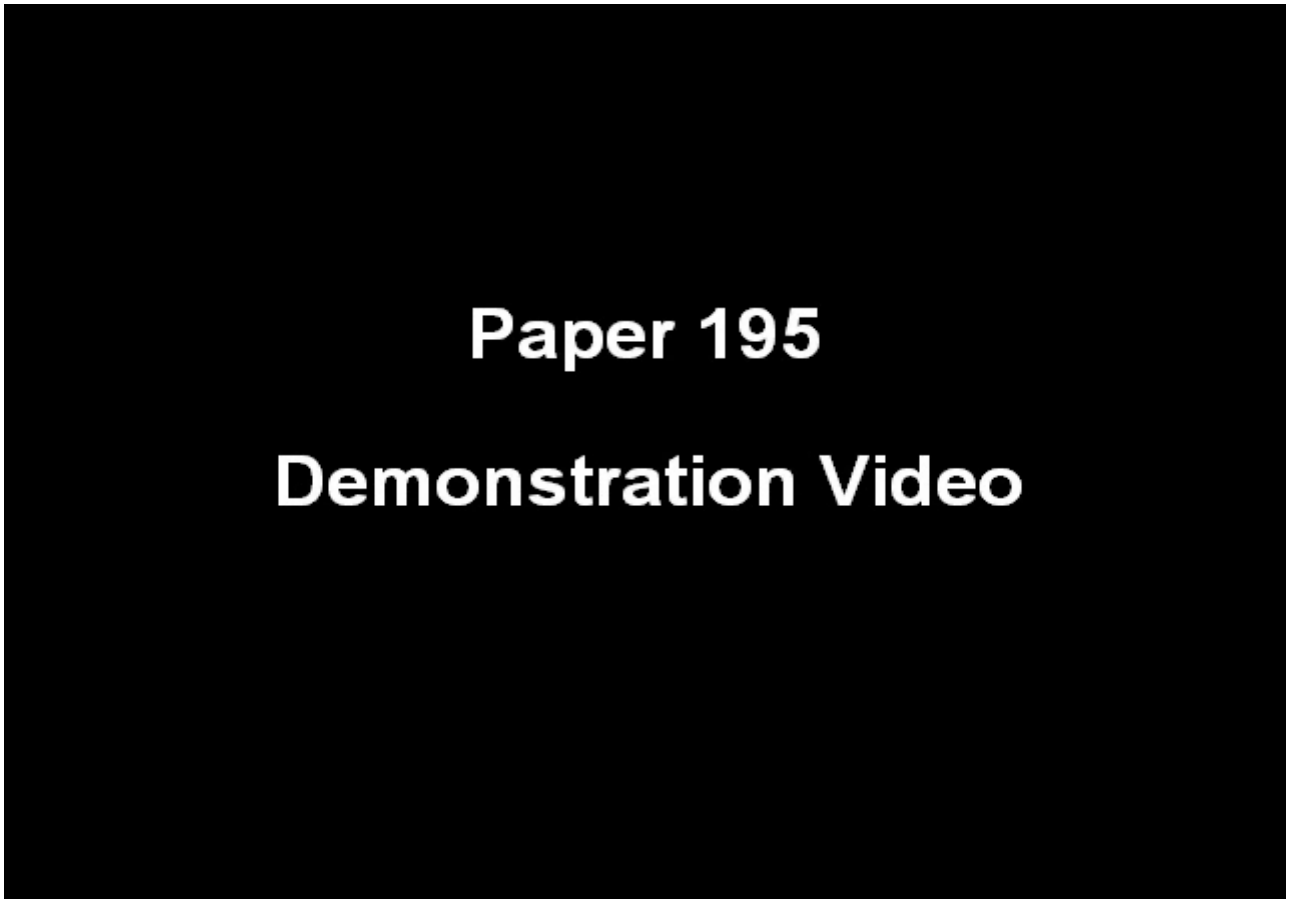
[image courtesy of Laine et al.]

Sibenik scene	80k triangles
Base method	7.1 FPS
Incremental	48.6 FPS

Results: Comparison [Laine et al.]



Demo [Laine et al.]



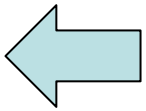
Incremental Instant Radiosity: Conclusions

+ Allows real-time global illumination
+ Dynamic local light source
+ No preprocessing required
+ General principle could be used for other methods
– Problems with dynamic scene changes
– Recent "Imperfect shadow maps" allow fully dynamic GI

Temporal Coherence in Object Space

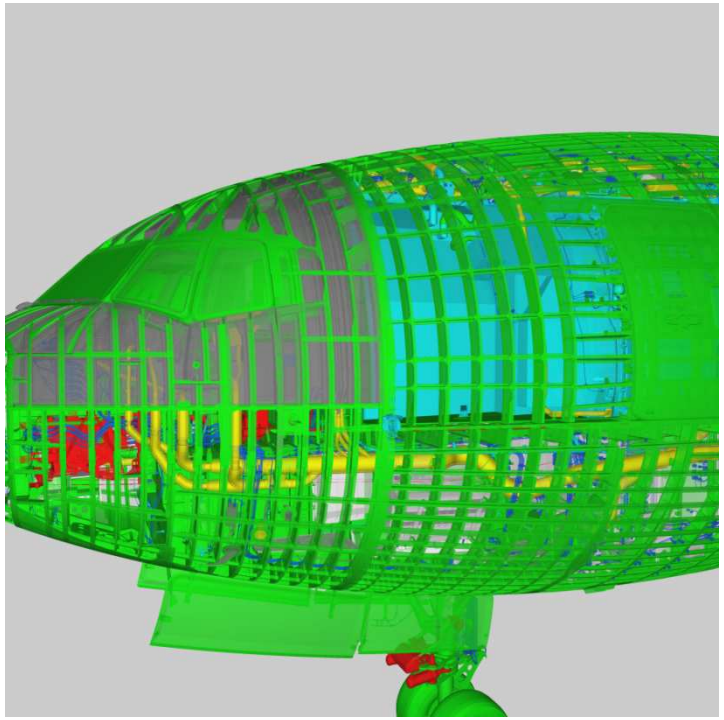
- Incremental global illumination

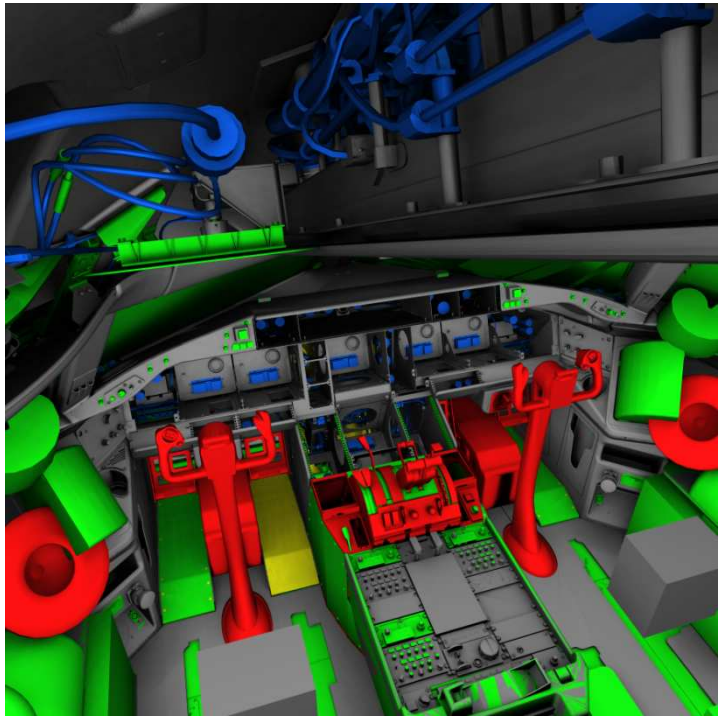
- Online occlusion culling



Visibility Culling: Motivation

- Massive models with complex interior parts
- Naive rendering impossible for most powerful GPUs

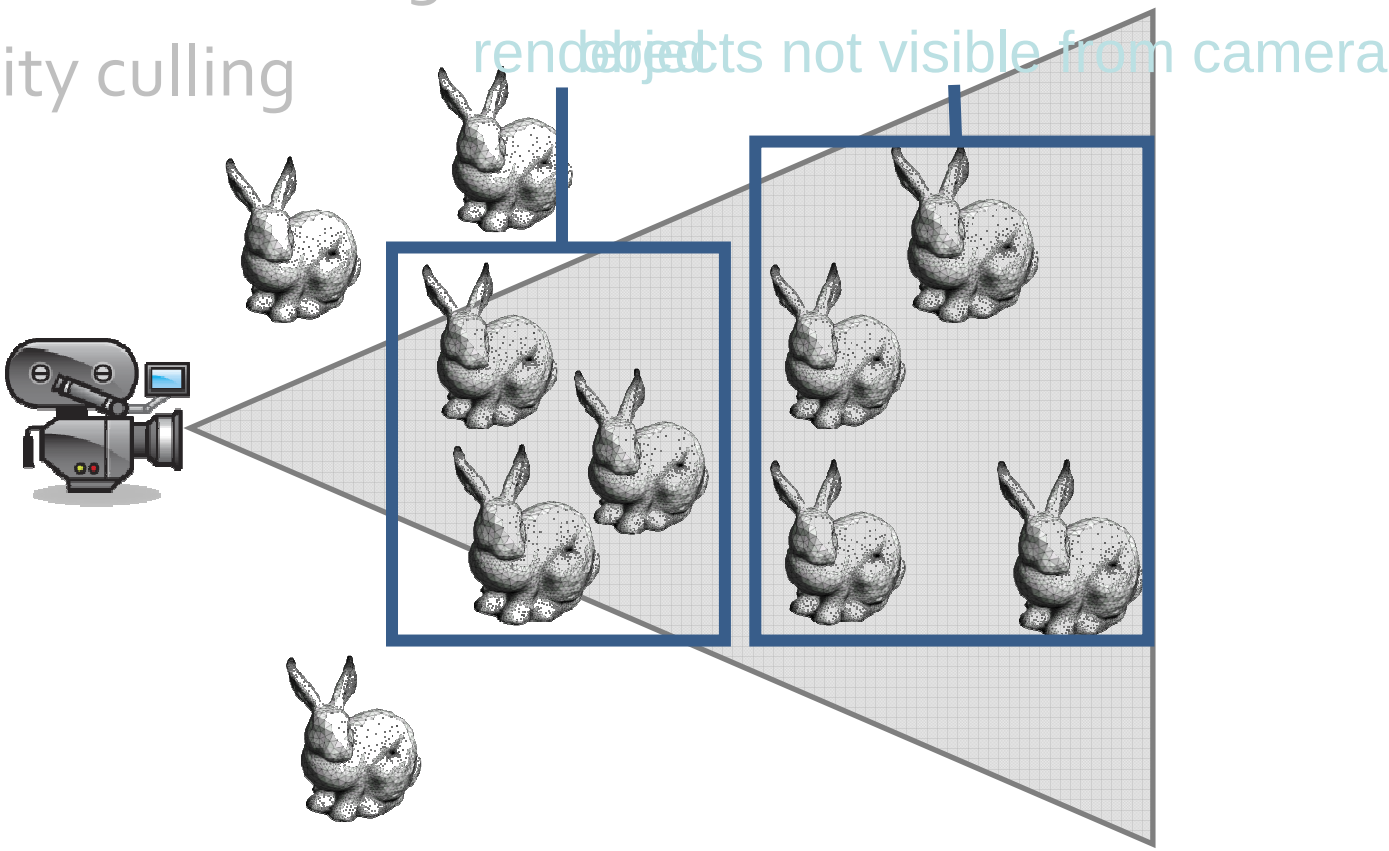




[image courtesy of Saarland university and Boeing]

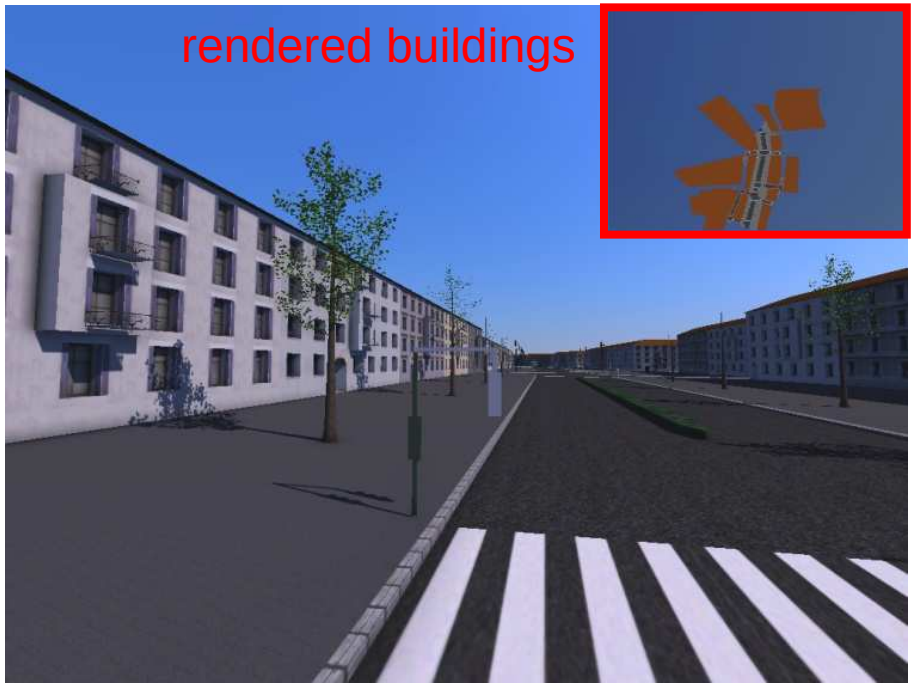
Culling Techniques

- Reduce number of rendered objects
- View frustum culling
- Visibility culling



Visibility Culling

- Only render visible primitives
- Resolve fragment-level visibility with depth buffer
- Goal: output sensitivity



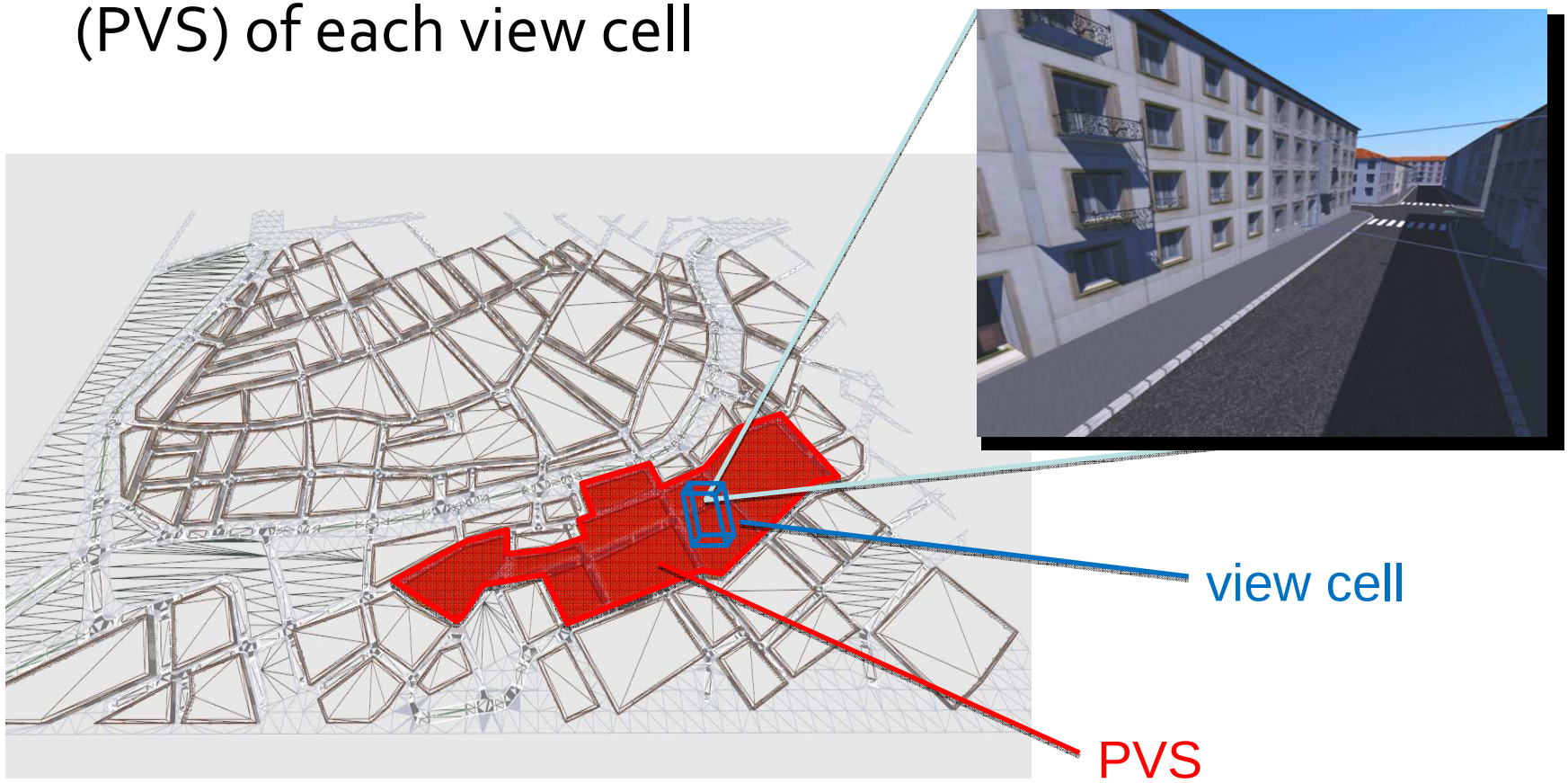
Visibility Culling: Paradigms

- Visibility preprocessing
- Online occlusion culling

Visibility Preprocessing

m1

- Partition view space into view cells
- Compute potentially visible set (PVS) of each view cell



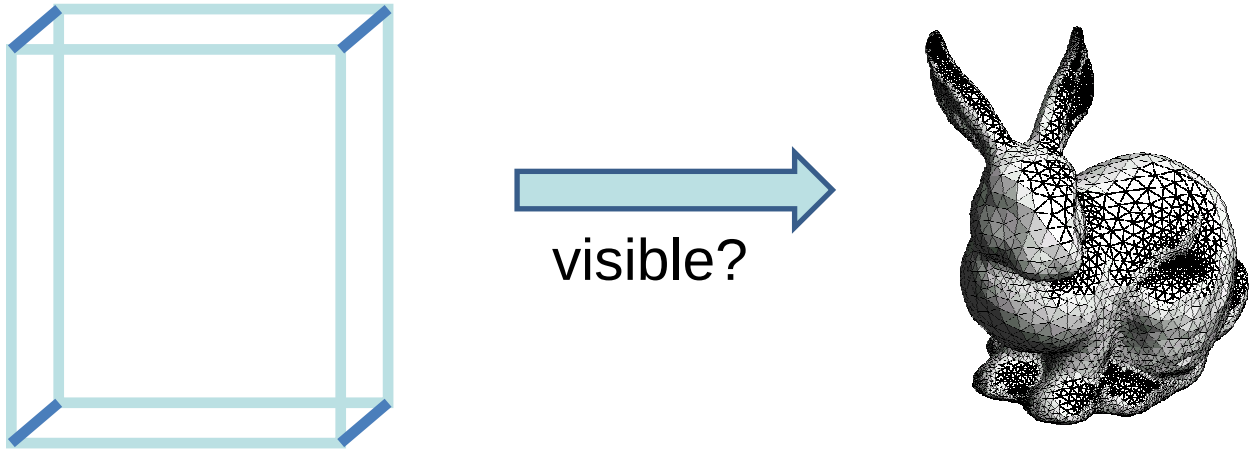
Folie 21

m1 teaser with town, 3d figure
matt; 20.06.2006

Online Occlusion Culling

- Query visibility from view point → occlusion query
- \+ No tedious preprocessing
- \+ Fully dynamic!

occlusion query: typical usage

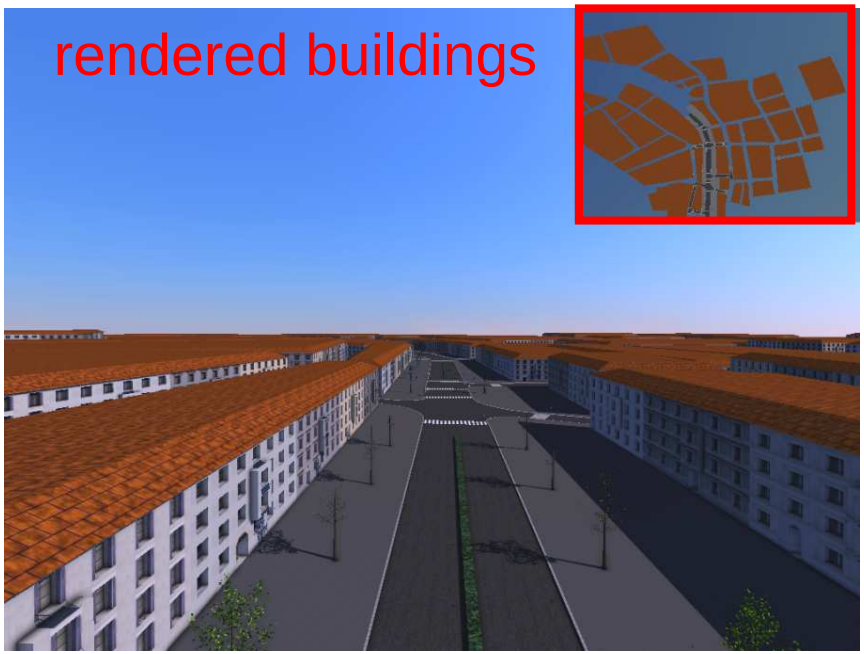


query bounding box

render complex geometry

Online Occlusion Culling: Problems

- Most objects are visible (overhead viewpoint) → many wasted queries!
- Naive method (query every object) infeasible!

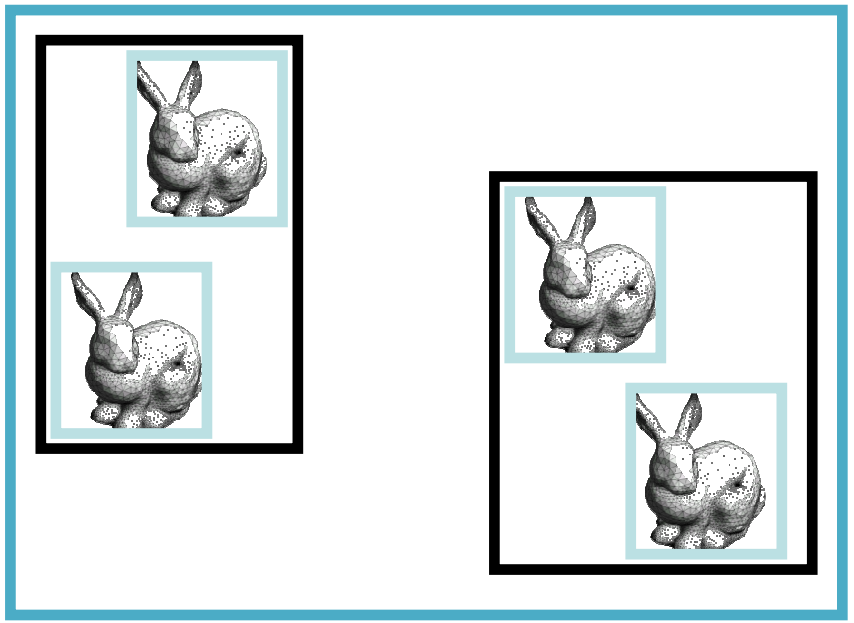


Occlusion Culling: Acceleration Methods

- Spatial coherence
  - Hierarchy over object / image space
- Temporal coherence
  - Assume objects to stay (in)visible

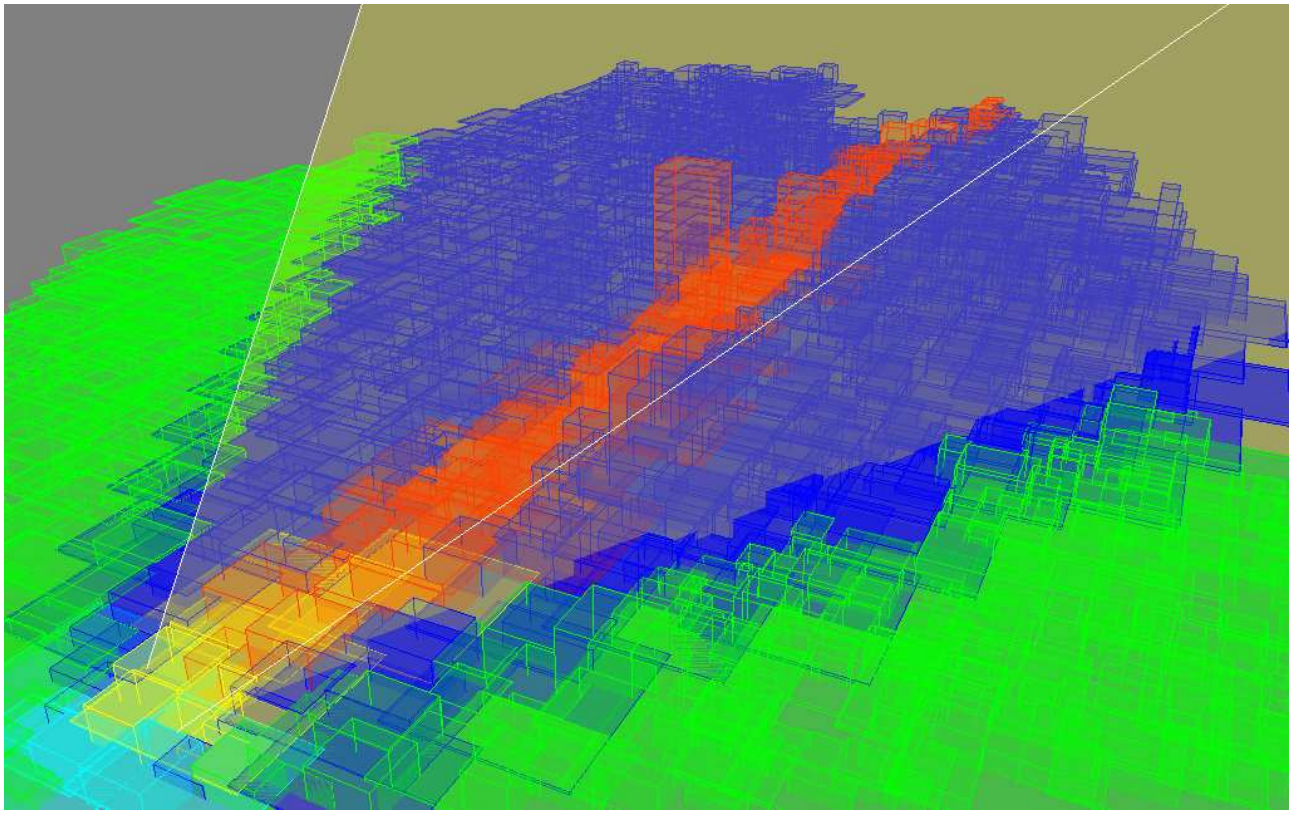
Occlusion Culling: Hierarchy Over Objects

- Bounding volume hierarchy (BVH)
  - Each object corresponds to single leaf
  - Easy to update (dynamic objects)
- Objects in the leaves
- Hierarchically query nodes
- Reduces complexity



BVH: 2D visualization

Occlusion Culling: Spatial Coherence

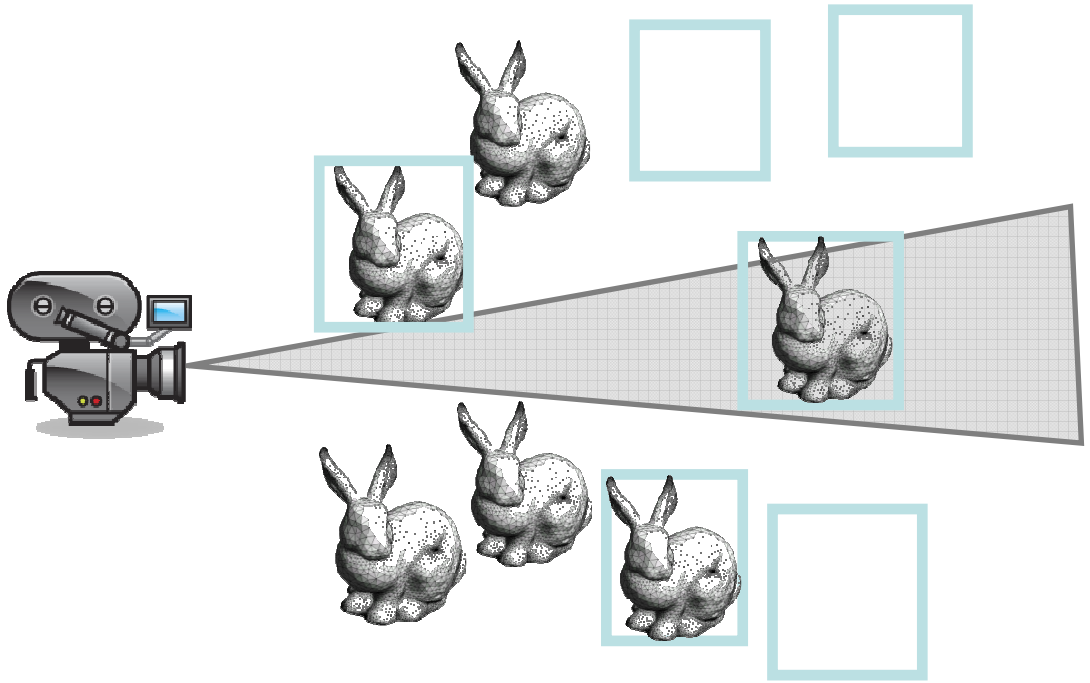


green	frustum culled nodes
blue	query culled nodes
red	rendered nodes

[image courtesy of Reinalter]

Temporal Coherence: General Strategy

- First render previously visible nodes ⟶ visible front
- Query previously invisible nodes
  - Render if visible
- Lazy update visible front (query sporadically)



Online Occlusion Culling: Previous Work

- Several methods exploited these principles
  - Hierarchical Z-Buffer Visibility [Greene94]
  - Hierarchical Occlusion Maps [Zhang01]
  - dPVS: An Occlusion Culling System for Massive Dynamic Environments [Aila01]
- No dedicated hardware support at the time
  → overhead!

Hardware Occlusion Queries

- Interface
  - Query (proxy) geometry
  - Fetch query result → returns number of pixels
    - Stalls if not ready!
  - Check query status

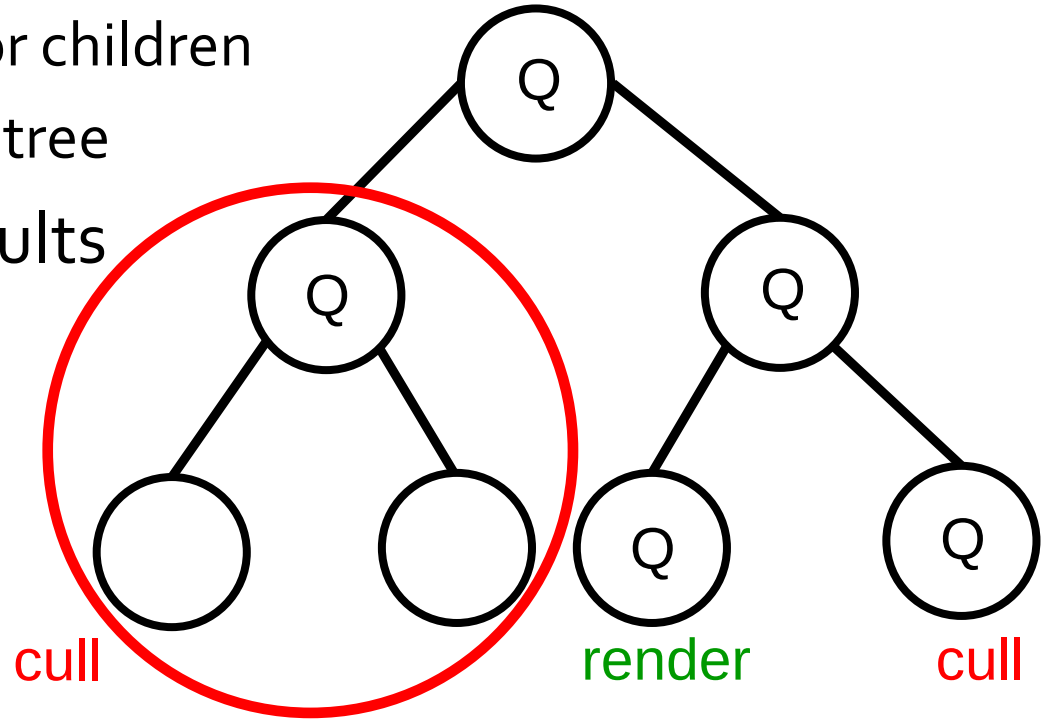
Hardware Occlusion Queries

+ Fast!
– Still not for free
– Latency until query result available on GPU
  → leads to CPU stalls (GPU starvation)

fortunately query status can be checked!

Naive method: Hierarchical Stop & Wait

- Approx. front-to-back traversal (priority queue)
- Geometry in the leaves
- Issue occlusion query (starting from root node)
  - Visible → repeat for children
  - Invisible → cull subtree
- Always wait for results → CPU stalls
- Wasted queries

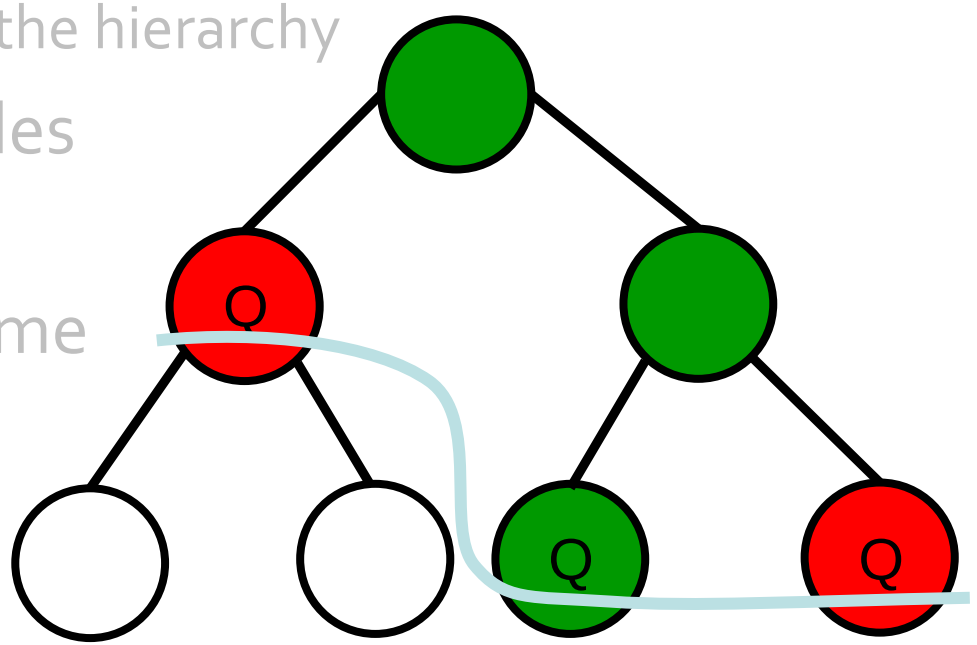


Hardware Occlusion Queries Made Useful

- Coherent Hierarchical Culling (CHC) [Bittner04]
- Use temporal coherence to reduce
  - Query number
  - CPU stalls

CHC: Reduce query number

- Assume visibility does not change!
- Store cut in the hierarchy
- Previously invisible nodes
  - Start queries at cut in the hierarchy
- Previously visible nodes
  - Always query leaves
- Adapt cut for next frame



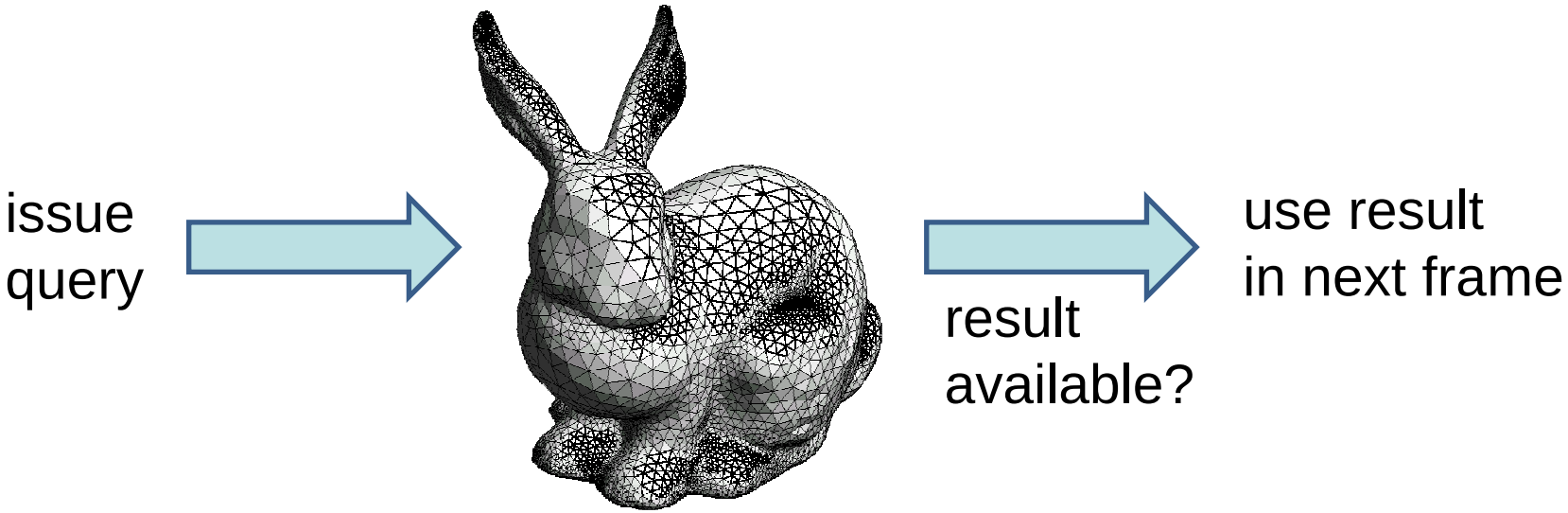
CHC: Reduce CPU stalls

- Interleave querying / rendering
- Store issued queries in query queue
- Check if first query result available
  - Not yet →
    do some traversal / rendering

but what to render?

CHC: Reduce CPU stalls

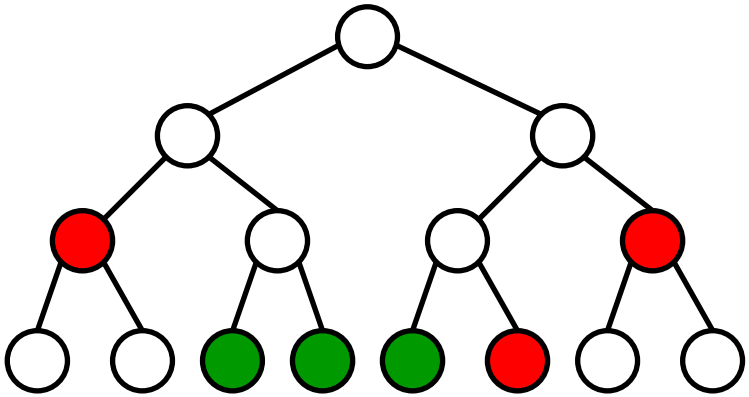
- Previously visible nodes
  - Assume that nodes stay visible (TC)
    → don't wait for query result



render geometry immediately

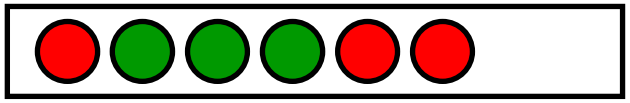
m2

Concept of CHC



traversal queue

issue queries



meanwhile,
do some work ...

fetch query results

query queue

m2 update picture
matt; 30.03.2008

CHC: Comparison Video

- View frustum culling (VFC) vs. CHC
- Note VFC frame rate drops!

framerate VFC

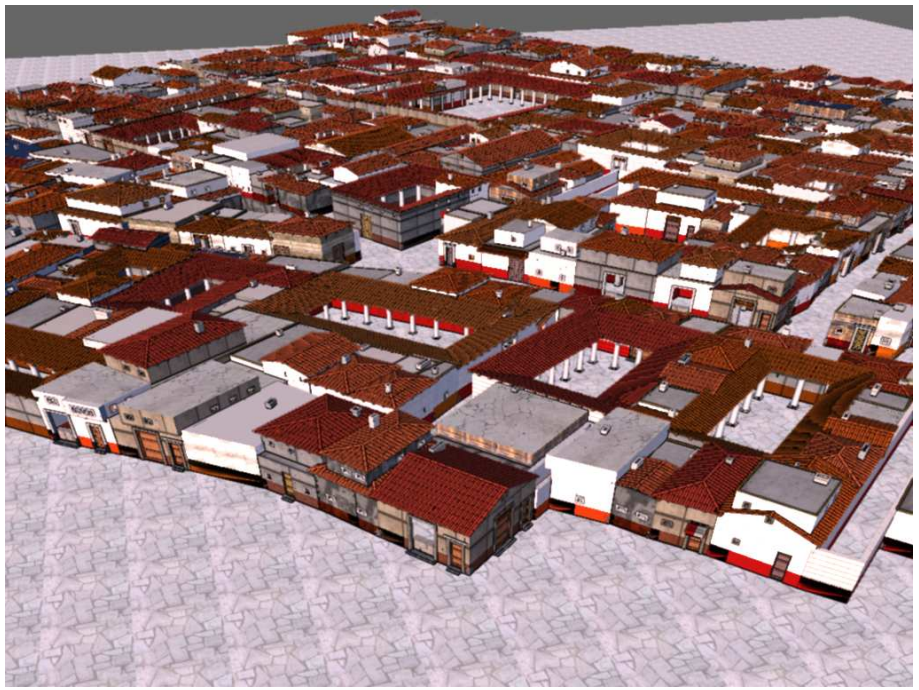
framerate CHC

bird‘s eye view

bird‘s eye view

Problems of CHC

- Queries all visible leaves → many wasted queries
- Query cost
- For overhead viewpoints often slower than VFC



CHC: Improved Algorithms

- Near Optimal Hierarchical Culling (NOHC) [Gutheo6]
- CHC++: Coherent Hierarchical Culling revisited [Mattausch08]

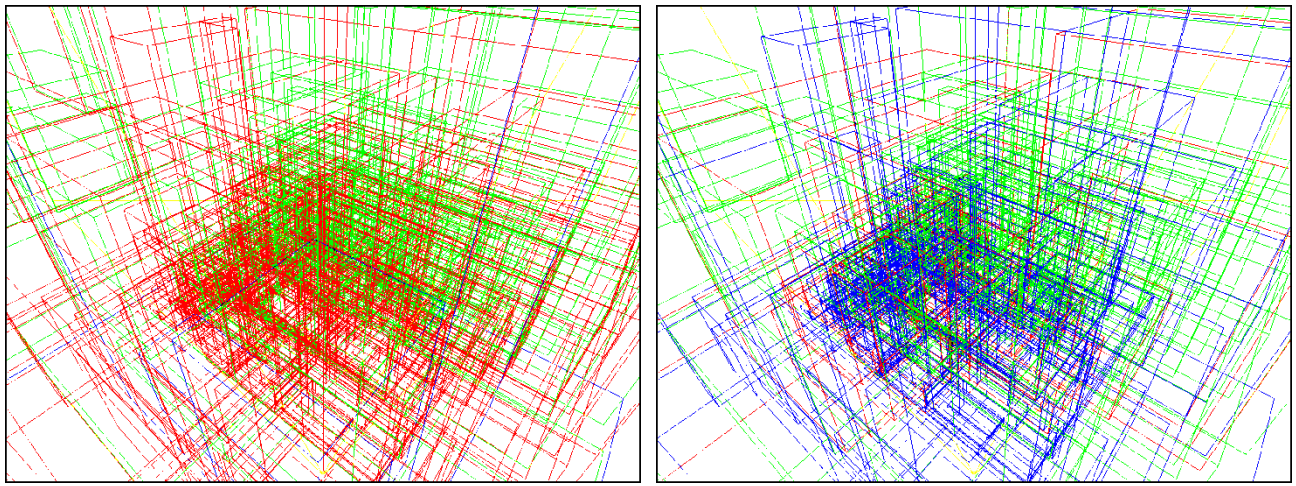
NOHC

- Estimation of query feasibility
  - Based on screen coverage and query cost
- Query only if cheaper than rendering

NOHC

+ Reduce number of wasted queries
- Hardware calibration step
- Complex set of rules

CHC NOHC



query visualization

[image courtesy of Guthe et al.]

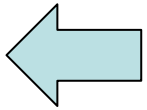
green	issued
red	wasted
blue	no query

CHC++: Make Further Use of TC

- Reduction of query number
  - Previously invisible nodes: multiqueries
  - Previously visible nodes: randomization
- Reduction of query cost
  - Query batching
- Keeps conceptual simplicity of CHC

Improved algorithm: CHC ++

- Reduction of query number
  - Previously invisible nodes: multiqueries
  - Previously visible nodes: randomization
- Reduction of query cost
  - Query batching



CHC++: Multiqueries

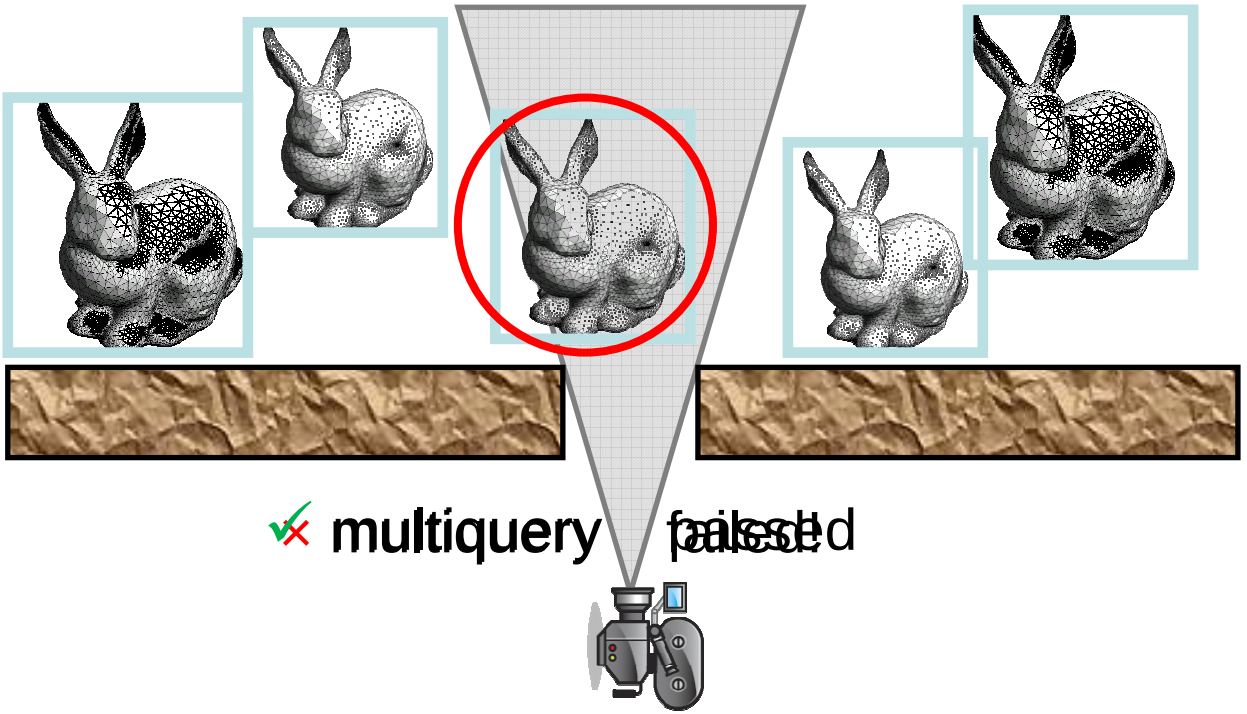
- Idea: If nodes invisible for long time
  - Likely to stay invisible
  - Example: car engine blocks in cars in a parking lot
- Cover many (distributed) nodes with single query

CHC++: Multiqueries

- Test q nodes with single (multi)query
  - Invisible ⟶ saved $(q-1)$ queries
  - Visible

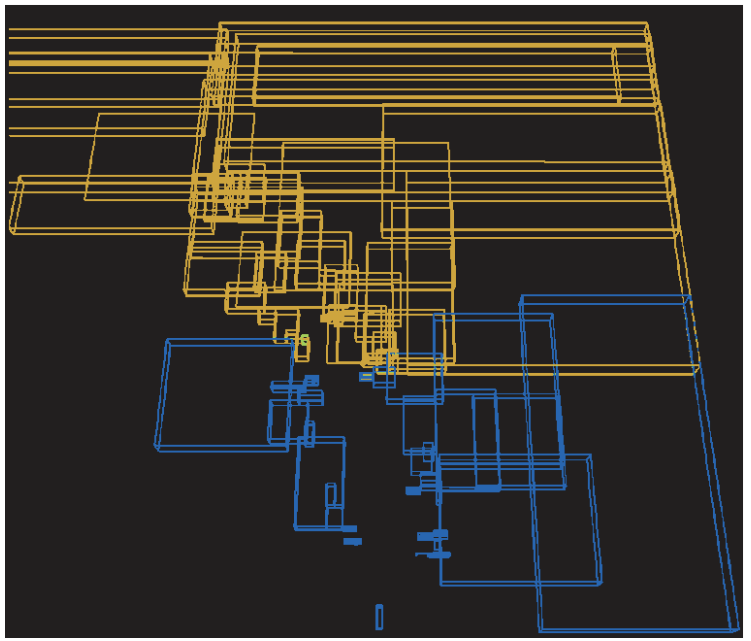
    ⟶ q more individual queries - wasted one query!

case 1:



Multiqueries: Greedy Algorithm

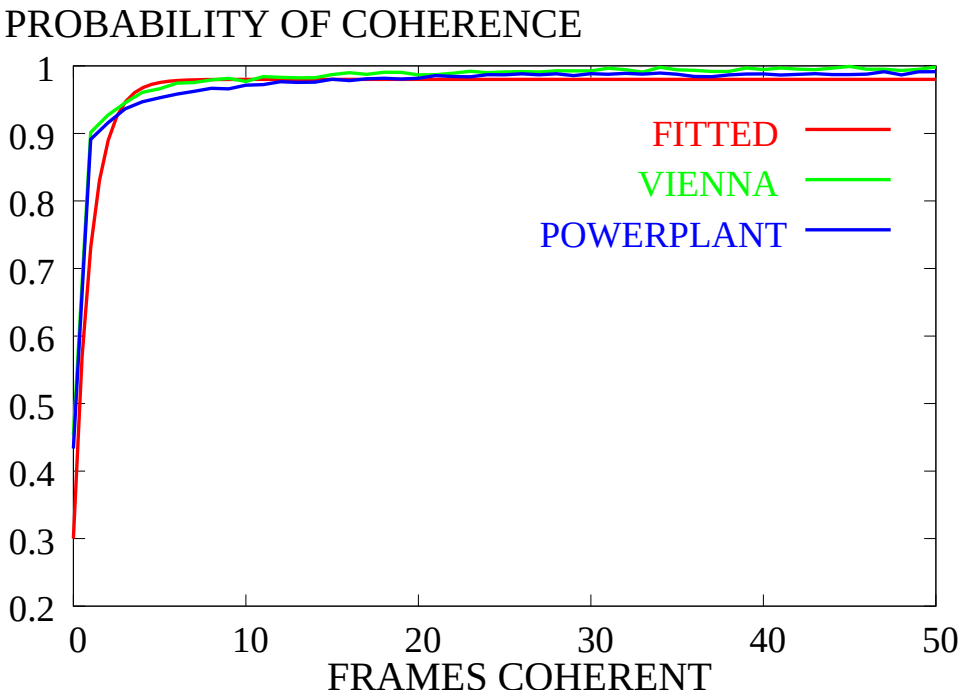
- Use probability that node stays invisible
- Find optimal query size with cost / benefit model



view covered with 3 multiqueries

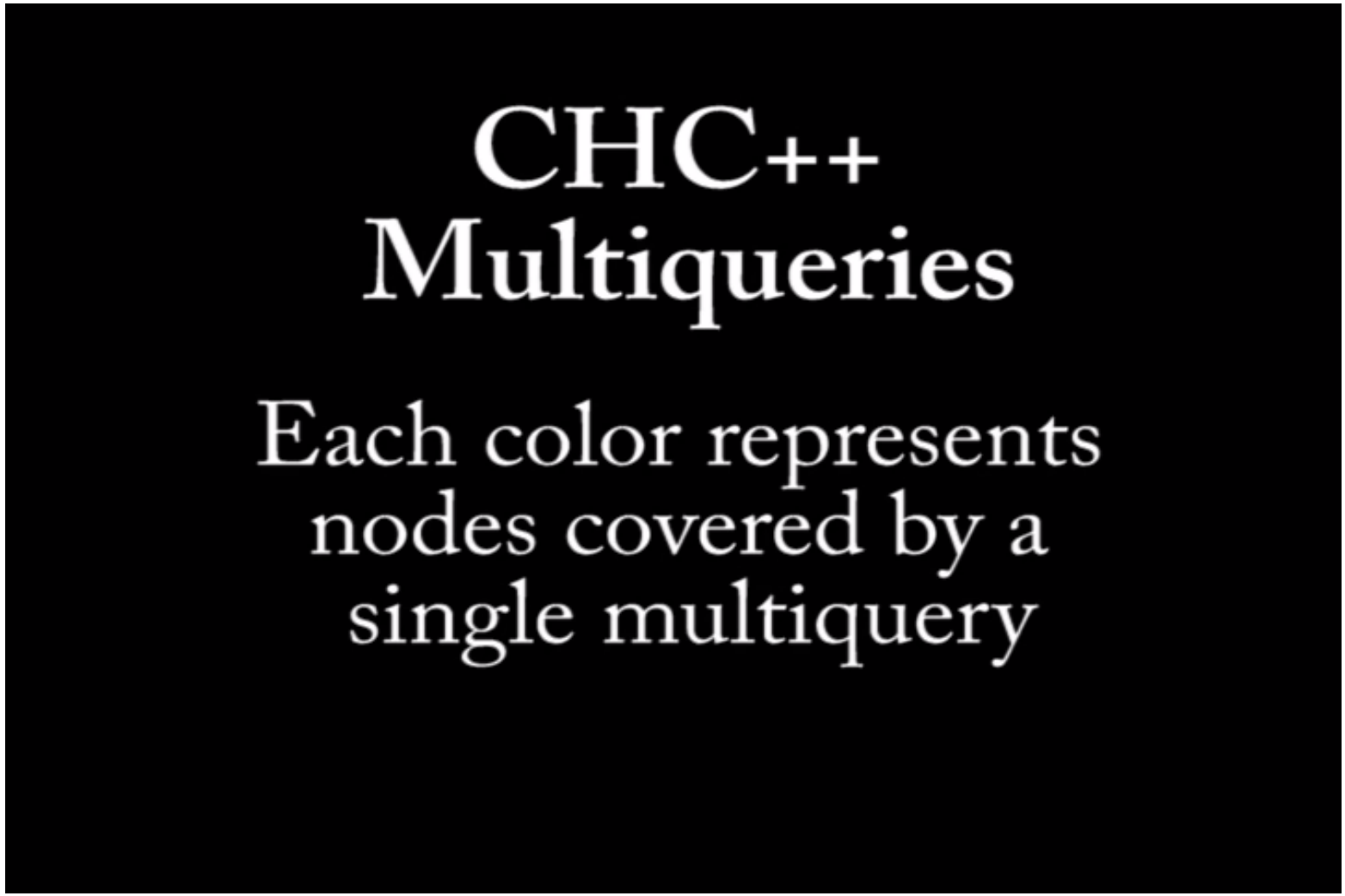
Multiqueries: Query probability

- Estimate probability that previously visible node will stay invisible if it was invisible for *k* frame
- Measurements behave like certain exp() function → sufficient in practice



Fitted and measured functions (for two scenes)

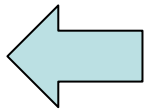
Multiqueries: Video



CHC++: each color represents a multiquery

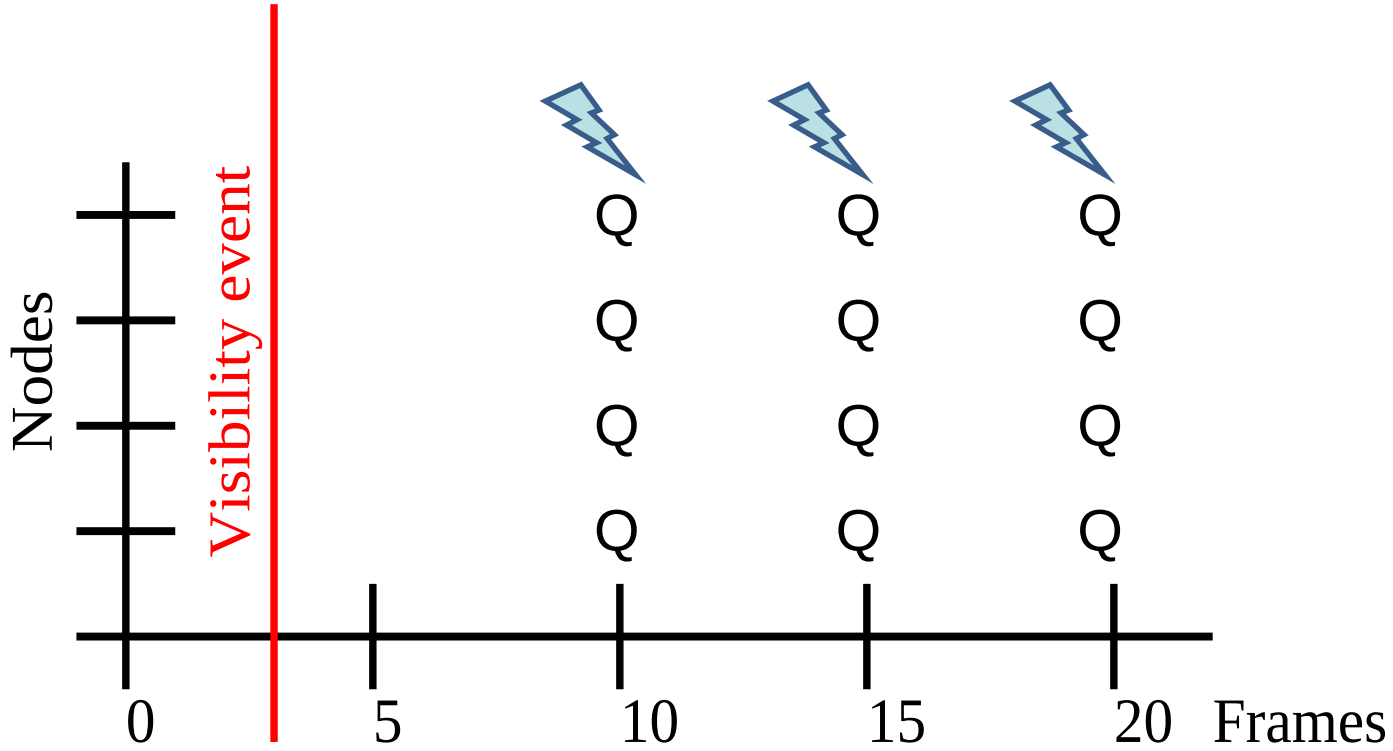
Improved algorithm: CHC ++

- Reduction of query number
  - Previously invisible nodes: multiqueries
  - Previously visible nodes: randomization
- Reduction of query cost
  - Query batching



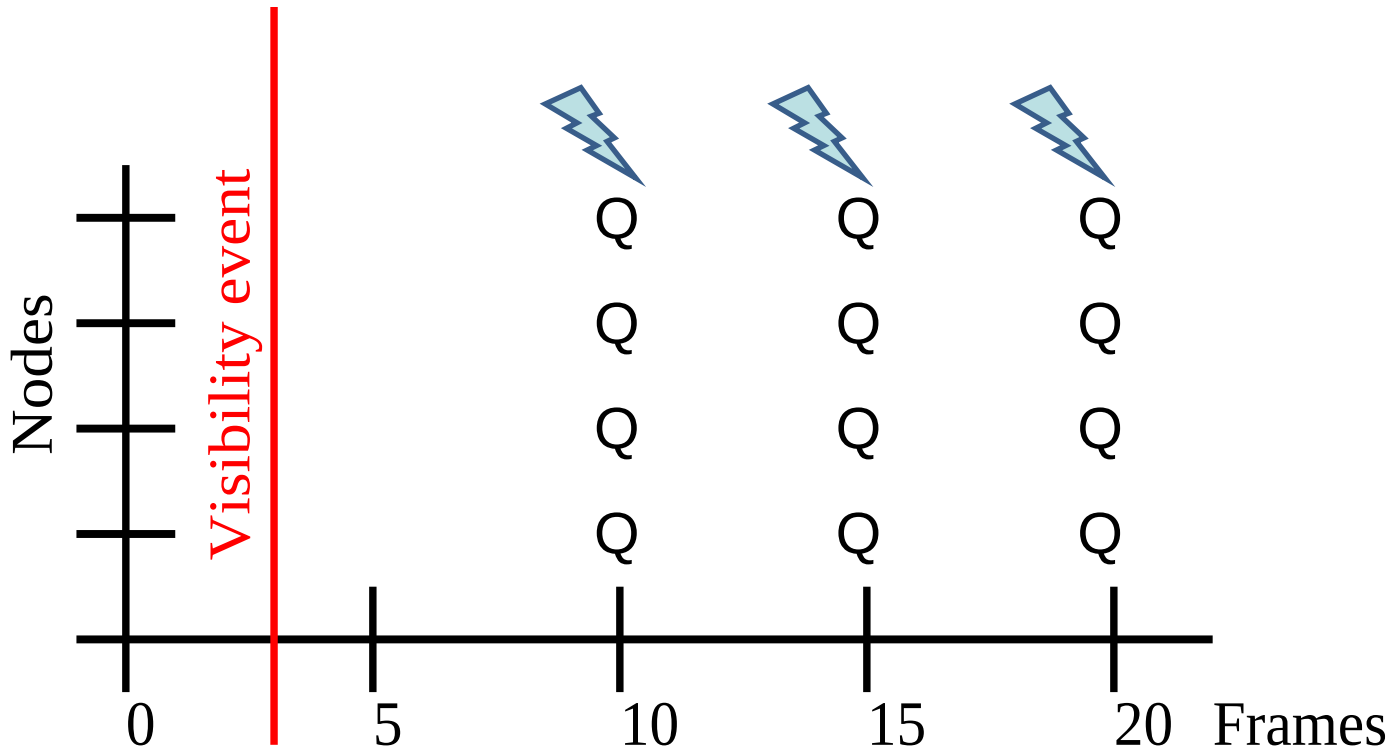
CHC++: Randomization

- Query visible nodes each frame → queries wasted
- Assume visible for *t* frames → spikes in frame time (if many nodes become visible simultaneously)



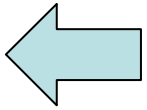
CHC++: Randomization

- When node becomes visible
  - Randomize first invocation between [1.. *t*]
  - Afterwards, test every *t* frames



CHC++: Make further use of TC

- Reduction of query number
  - Previously invisible nodes: Multiqueries
  - Previously visible nodes: Randomization
- Reduction of query cost
  - Query batching



CHC++: Query Cost

m3

- Switch between render / query mode
  - Induces GPU state change
  - Query must not change depth buffer → depth write on / off
  - State changes can be very costly on modern GPUs!
  - Main source of query overhead
- Interleaving of queries / rendering
  → CHC induces one state change per query

m3 show image of engine block

matt; 13.05.2007

CHC++: Query Batching

m4

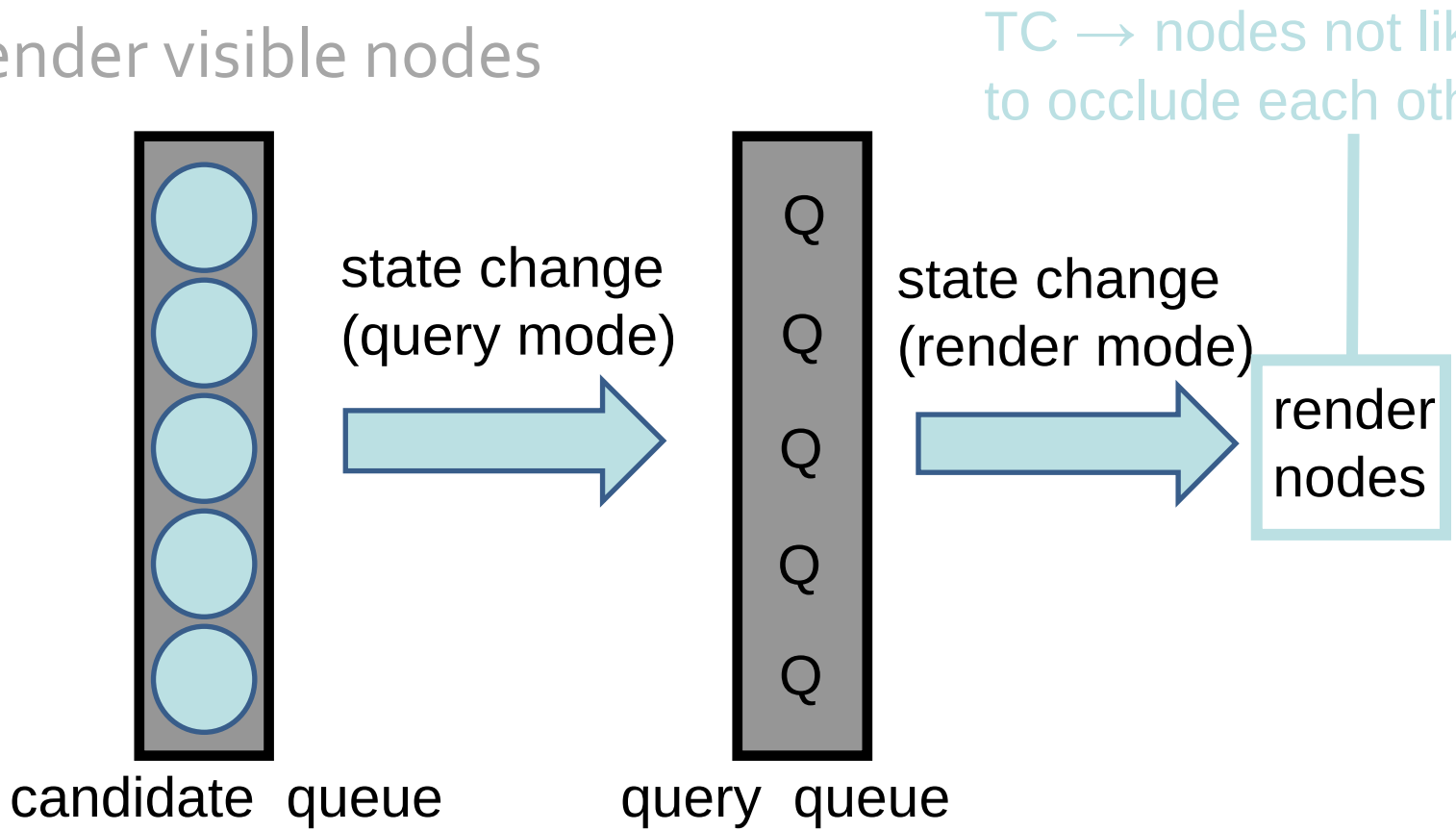
- Reduce state changes
  - By reducing switches between render / query mode
- Store query candidates in extra queues
  - Separate queues for previously (in)visible nodes
- Previously invisible nodes
  - Issue many queries in parallel
  - Render nodes afterwards
  - Assume that nodes don't occlude each other because of TC

m4 show image of engine block

matt; 13.05.2007

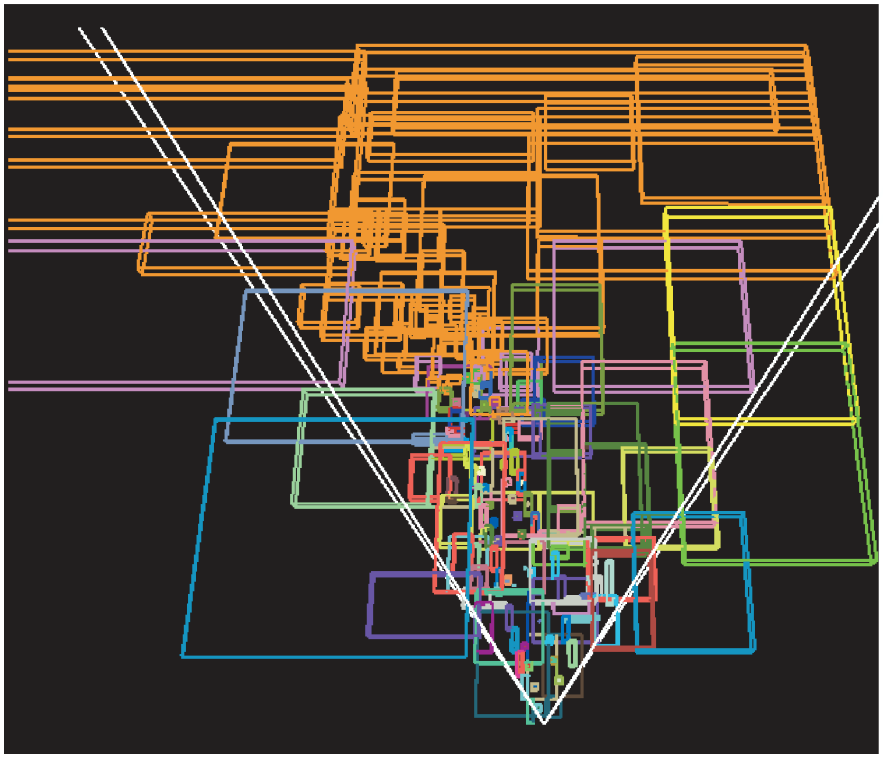
Query Batching: Previously Invisible Nodes

- Collect *n* nodes
- Query nodes in parallel
- Render visible nodes

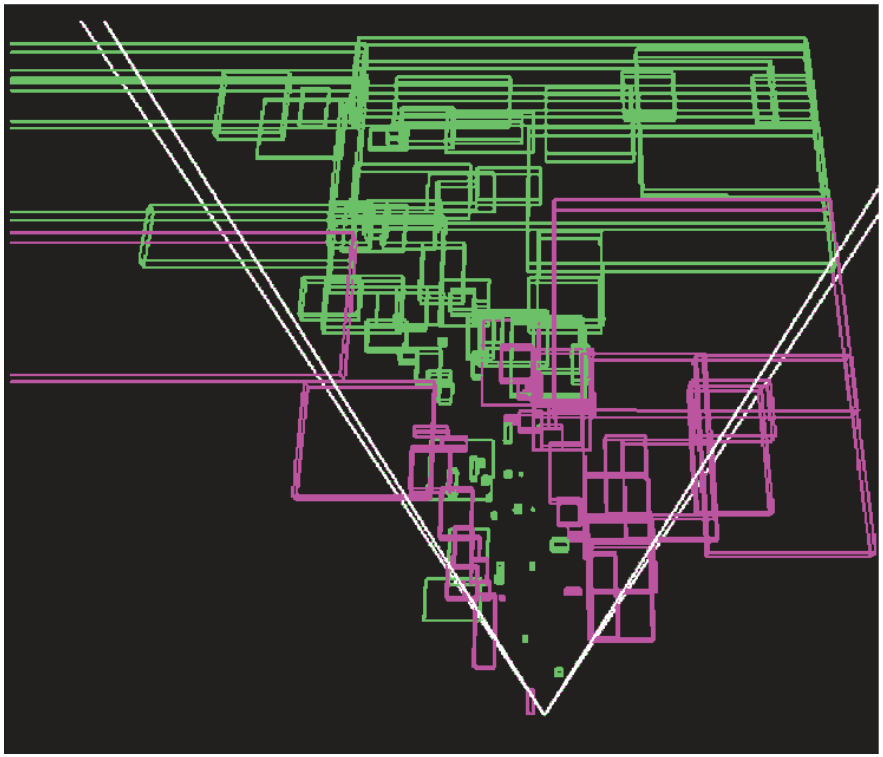


Query Batching: Visualization

each color represents a state change



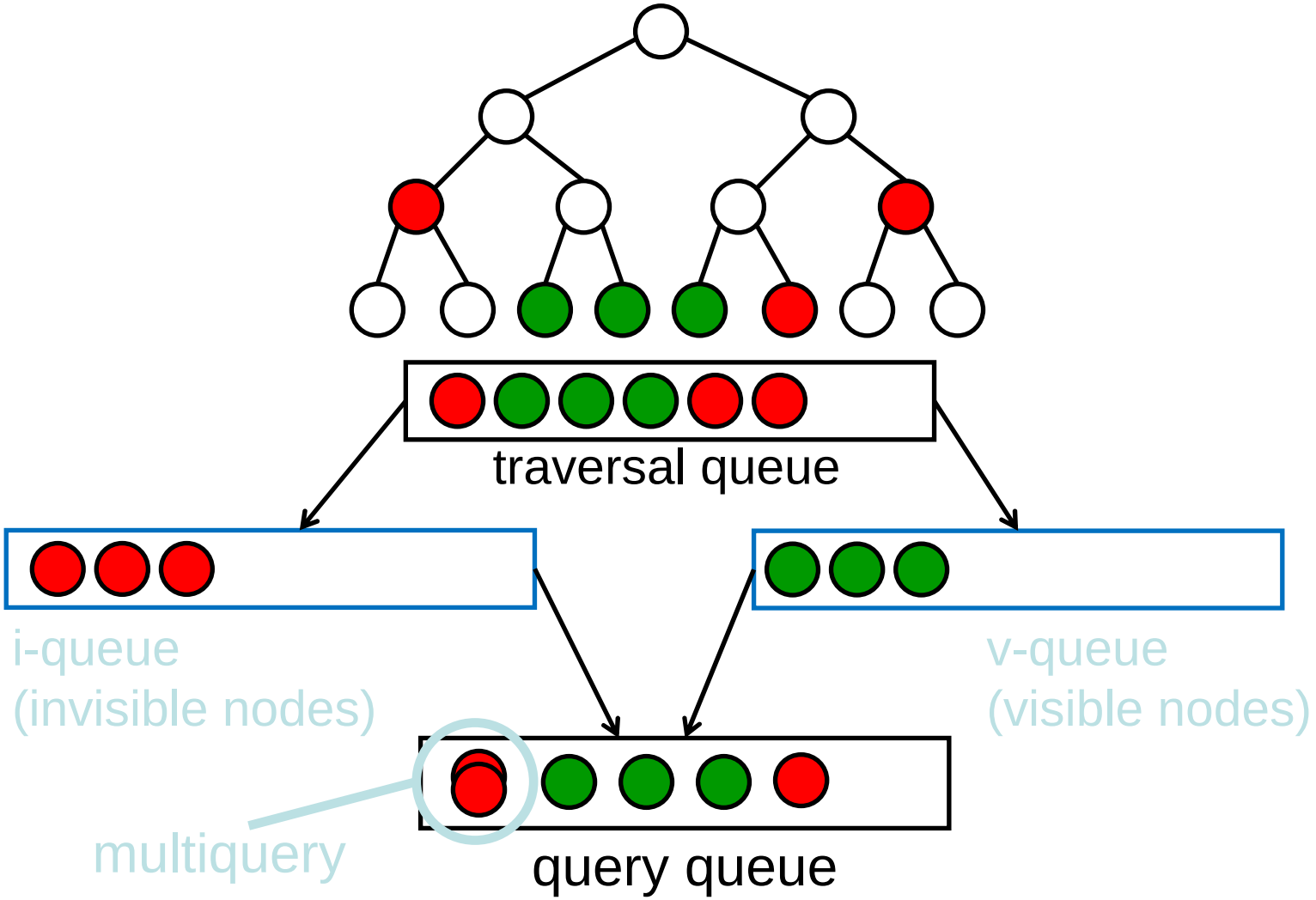
CHC (~100 GPU state changes)
→ query overhead!



CHC++ (2 GPU state changes)

m5

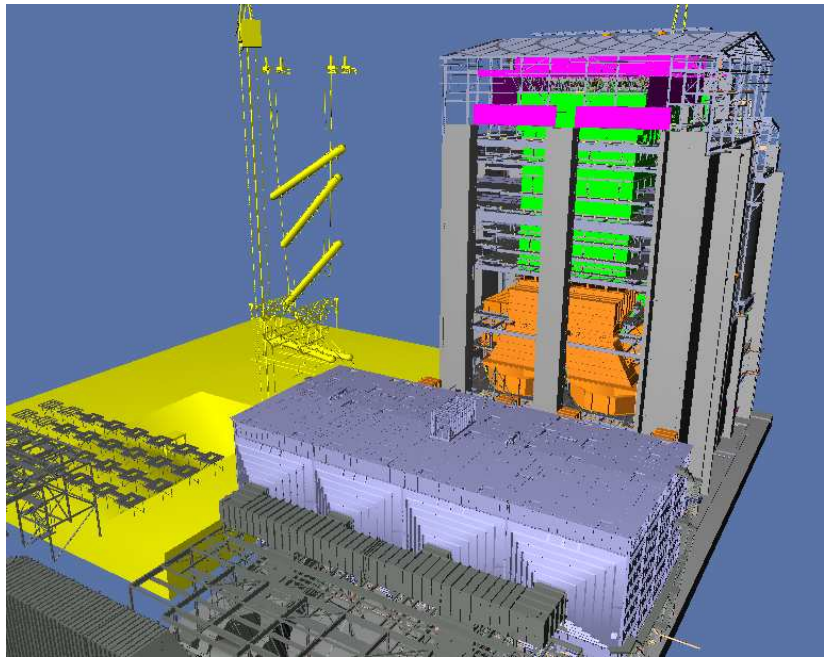
Concept of CHC++



m5 update picture
matt; 30.03.2008

m6

Results: Powerplant (12M triangles)

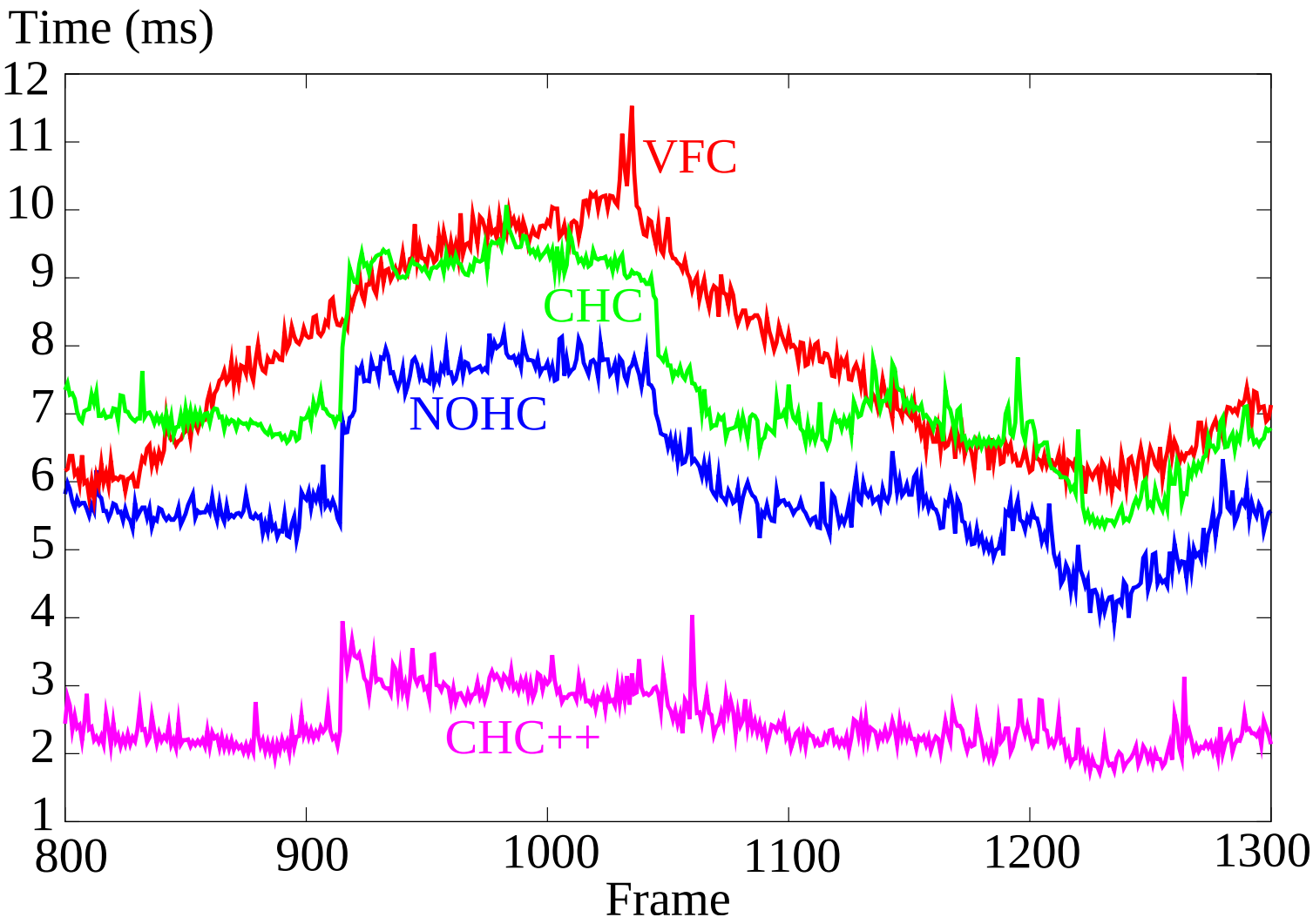


VFC	view frustum culling
PVS	potentially visible sets (= preprocessing)
NOHC	near optimal culling
CHC	coherent hierarchical culling
CHC++	

m6 show teaser: state change reduction or just graph!
only show vfc, chc and chc ++
matt; 06.03.2008

m7

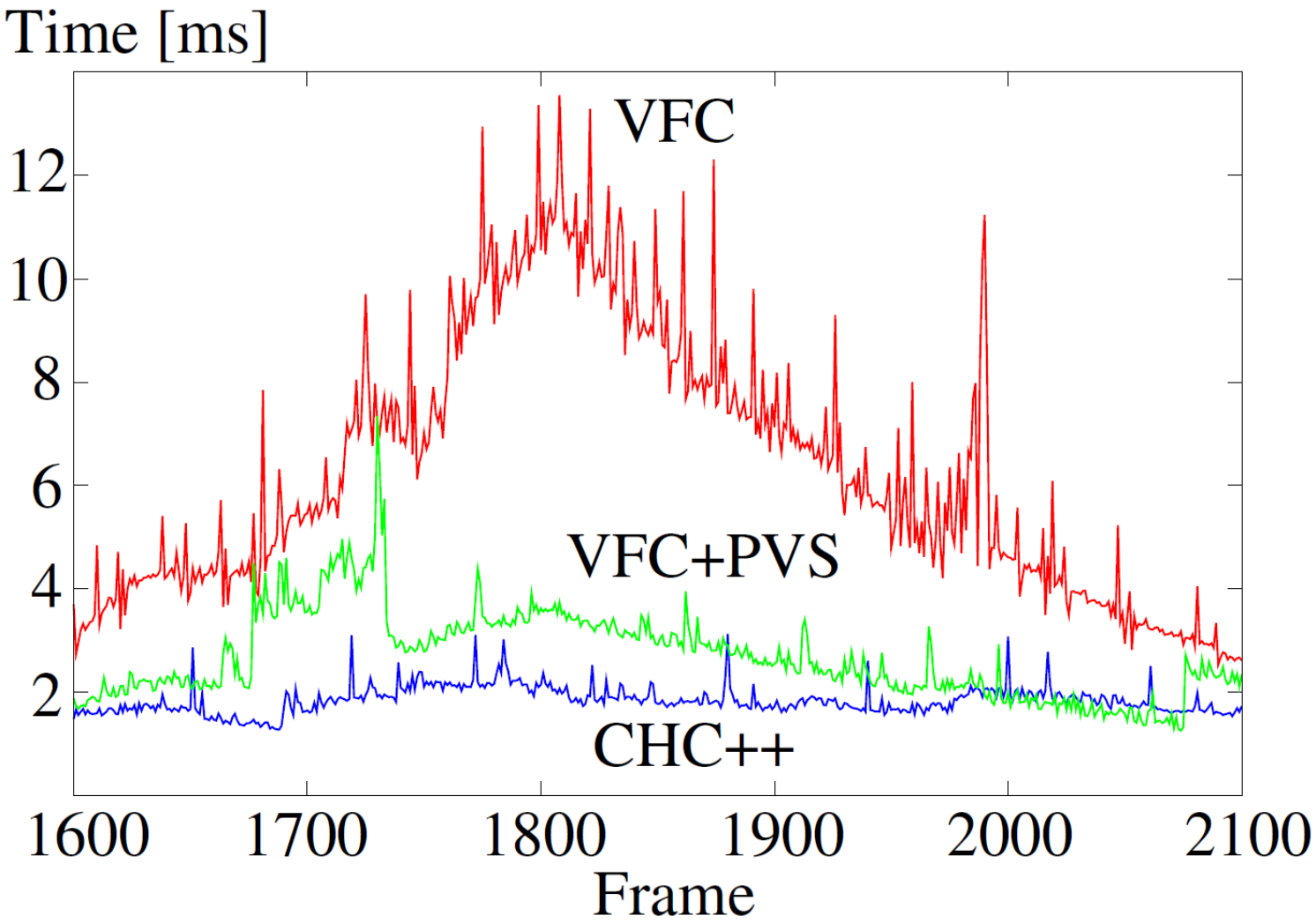
Results: Difficult walk in Powerplant



m7 show teaser: state change reduction or just graph!
only show vfc, chc and chc ++
matt; 06.03.2008

m8

Results: Powerplant (12M triangles)



m8 show teaser: state change reduction or just graph!
only show vfc, chc and chc ++
matt; 06.03.2008

Online Occlusion Culling: Conclusions

+ Can be highly optimized with TC!
  + CHC uses TC to reduce CPU stalls and query numbers
  + CHC++ uses TC to further reduce query number and cost
+ Fully dynamic scenes
+ Often faster than preprocessing
– Still some minor query overhead
– No guarantee to fully eliminate CPU stalls

Temporal coherence in Object Space