Visibility-guided rendering for real time visualization of extremely large data sets

B. Brüderlin, S. Pfützner, M. Heyer
3DInteractive GmbH, Ilmenau & Technical University Ilmenau
www.3dinteractive.de

Tutorial Overview

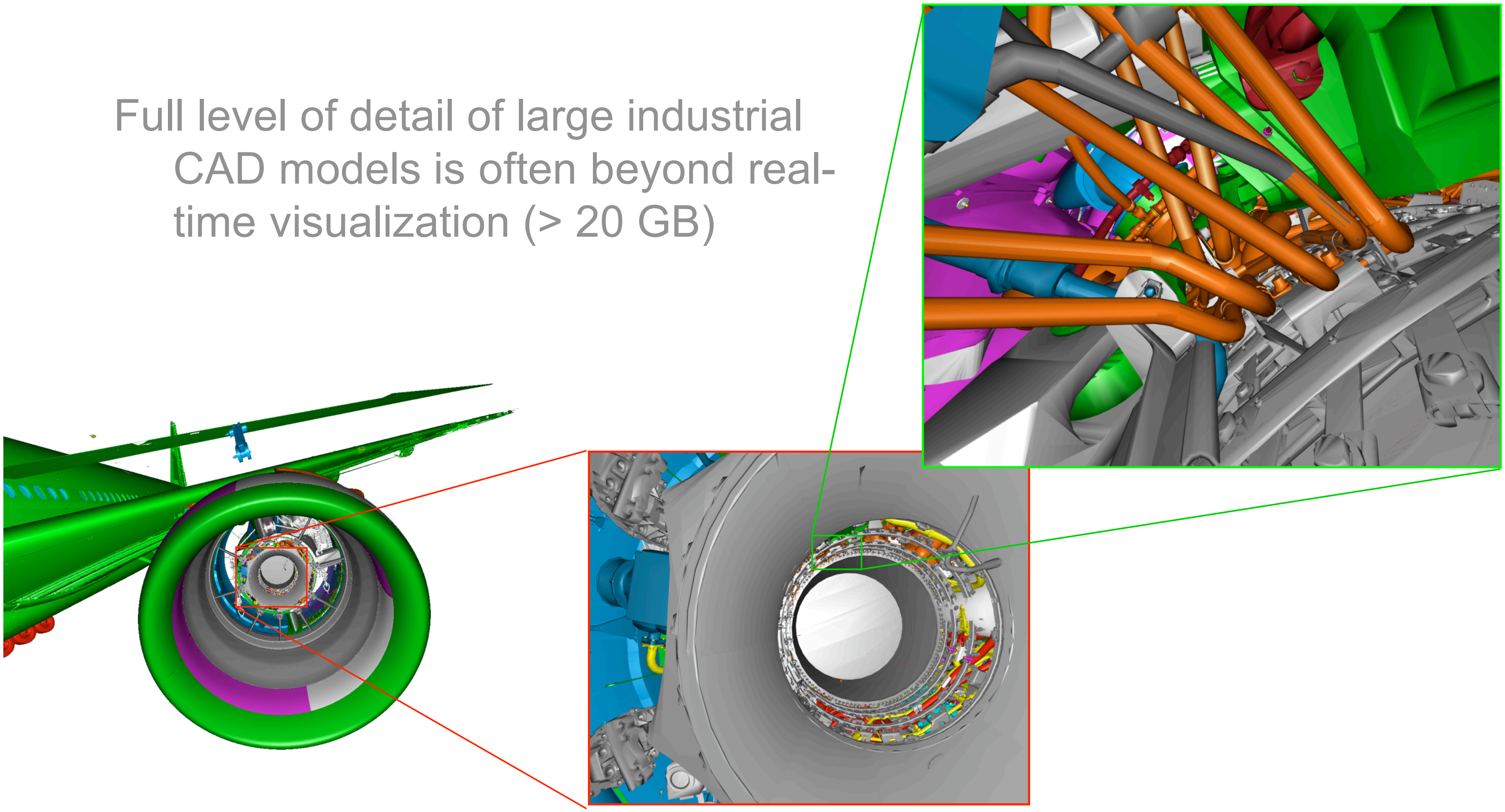
- Rendering Requirements for Engineering & Styling
- Looking at Classical Approaches
  - Rasterization (HW-based OpenGL)
  - Sampling-based (Raytracing)
- Visibility-guided Rendering (basic priciples)
  - Why ? How? How well?
- System Issues With Large Data Sets
  - out-of-core, memory, disk, preprocessing
- Conclusion & Outlook, Related Issues

Requirements of Engineering & Styling

- Engineering: Extremely large data sets (DMU) >> 4GB
  - CAD models
  - Real time interaction
- Styling: Realistic appearance: materials & lighting
  - High quality (off-line)
  - Virtual Reality (real time, interactive)
- Scalability: speed / quality / hardware cost
  - PDA; laptop, workstation, high-end render server

Extremely Large-scale CAD Models

Full level of detail of large industrial CAD models is often beyond real-time visualization (> 20 GB)



Scalability: Hardware Performance vs. Cost

Laptop

PC / Workstation

Graphics Server: multi CPU / GPU
SGI Prism

Standard PC Graphics Hardware / OpenGL
Linux 64 or Windows 32 operating systems

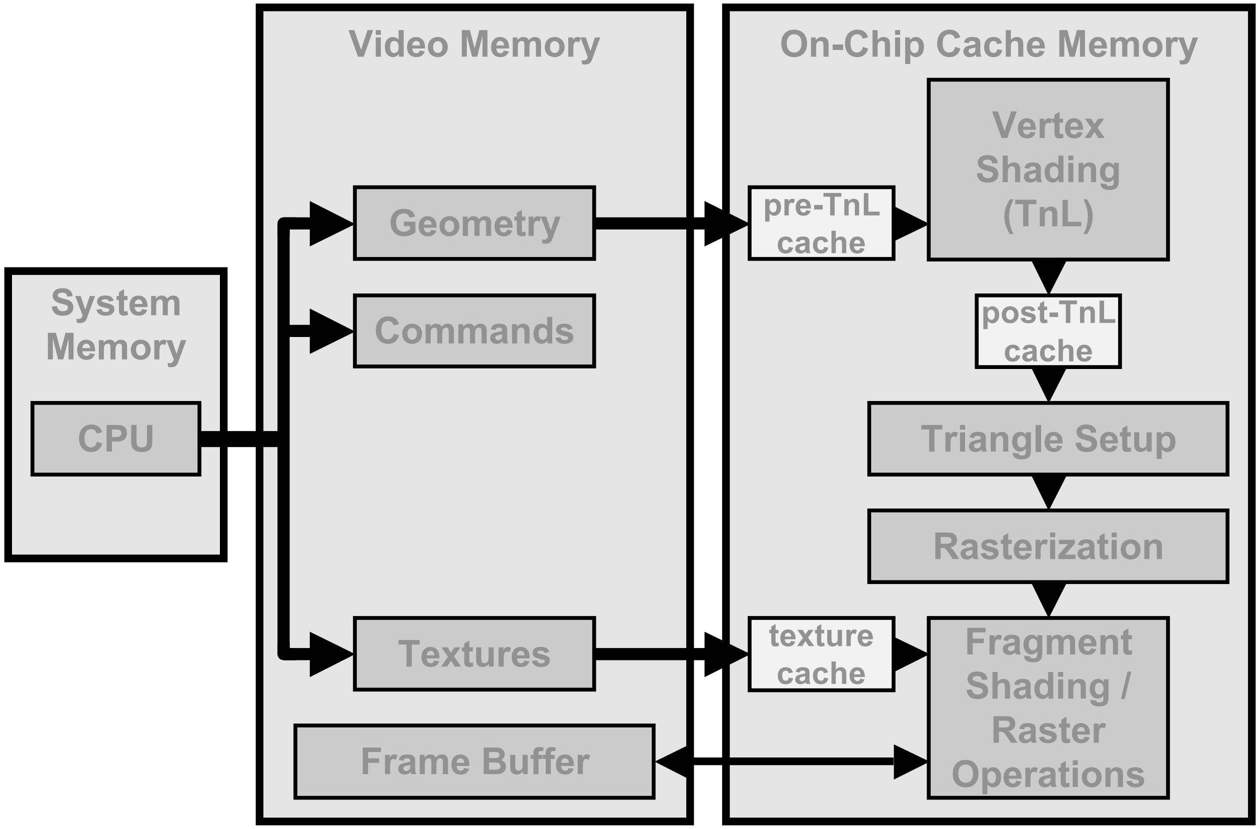
The Classical Approaches:

Graphics Hardware (GPU) Rasterization / Open GL

- Per Vertex Operation
  - Transformation (Object-, View Transformation)
  - Phong Illumination (Gouraud Shading)
    - → Polygon limitation!! O(n)
- Per Fragment (Pixel) Operation
  - Raster Conversion (polygon filling)
  - Shading (Interpolation or per pixel shading)
  - Z-buffer – Test
  - Pixel Shader GLSL
    - → Fill rate limitation!!
- Performance Measurement (peak performance)
  - 100 Mio Triangles (@ 100 pixels) / second
  - 10 Billion Pixels / Second
- Realistic: 1-5 Mio Polygons 10 fps (with vertex buffer)
  - Memory limitations, Render Calls, Bus Bandwidth (e.g. AGP 8 x, PCI Express)

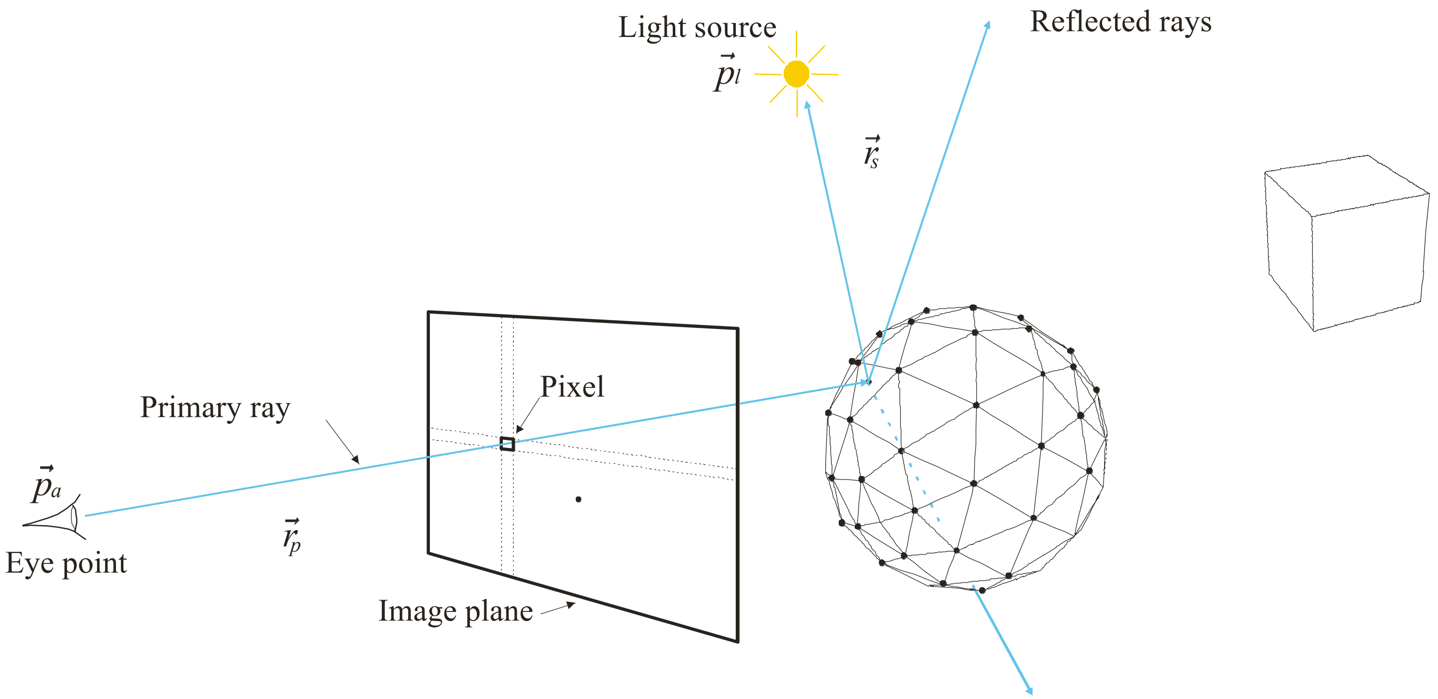
The Render Pipeline

Polygon → pixel
Visibility: Raster-based (z-buffer at end of pipeline) → enormous overdraw !



Raytracing

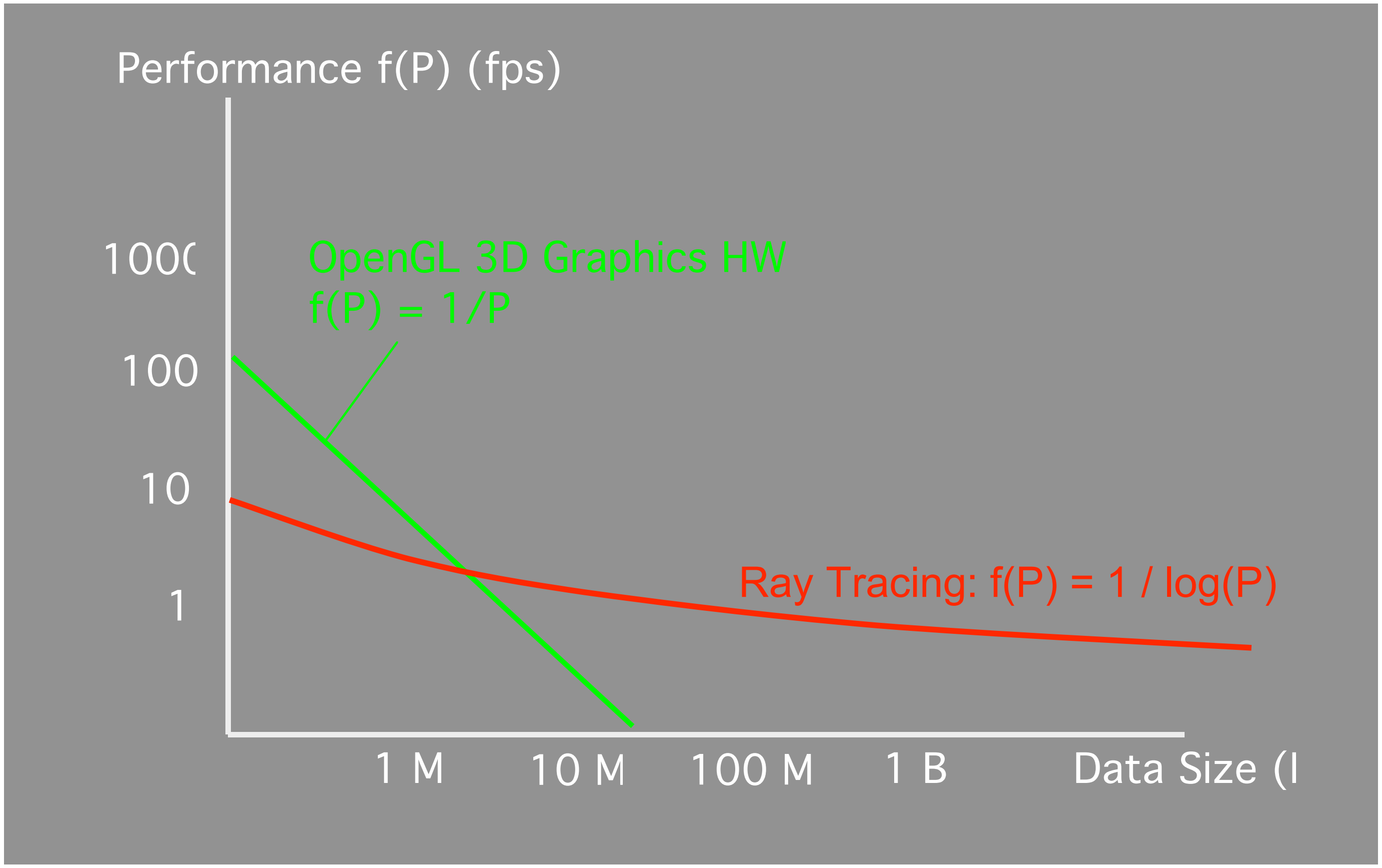
Pixel → polygon
Visibility: geometry based → no overdraw !



Per Pixel Operation:

- Search polygon in spatial data structure:
  O(log n); $O(\sqrt[3]{n} \cdot \log n)$
- Intersect, Shade: O(1)
- Indirect lighting (reflection, refraction, shadows)

Performance Comparison



Average PC (2005) 3GHz CPU, 3D graphics card, 500MB memory and 1M pixel display, direct Phong lighting, (no reflections, shadows, etc.).

Restrictions of the Classical Solutions & A Way Out

- Model resolution is reduced for real-time handling. Disadvantage:
  - Loss of detail
  - Additional costs through outsourcing
  - Manual work; Loss of time
- Is it really either, or?
  - Raytracing vs. OpenGL?
  - Real time vs. off line?
  - Photorealistic vs. interactive?
- The case for Visibility-guided Rendering!
  - Combine the advantages of both approaches

Basics of Visibility-guided Rendering
Why VGR?

Observation: Large scenes with > 100 M polygons / Screen 1 M Pixels

Each polygon covers < 1/100 pixel (on average)

Individual polygons often either

- are outside the view area
- have sub-pixel size
- are hidden by other polygons

Visibility Culling Techniques:

→ View frustum culling

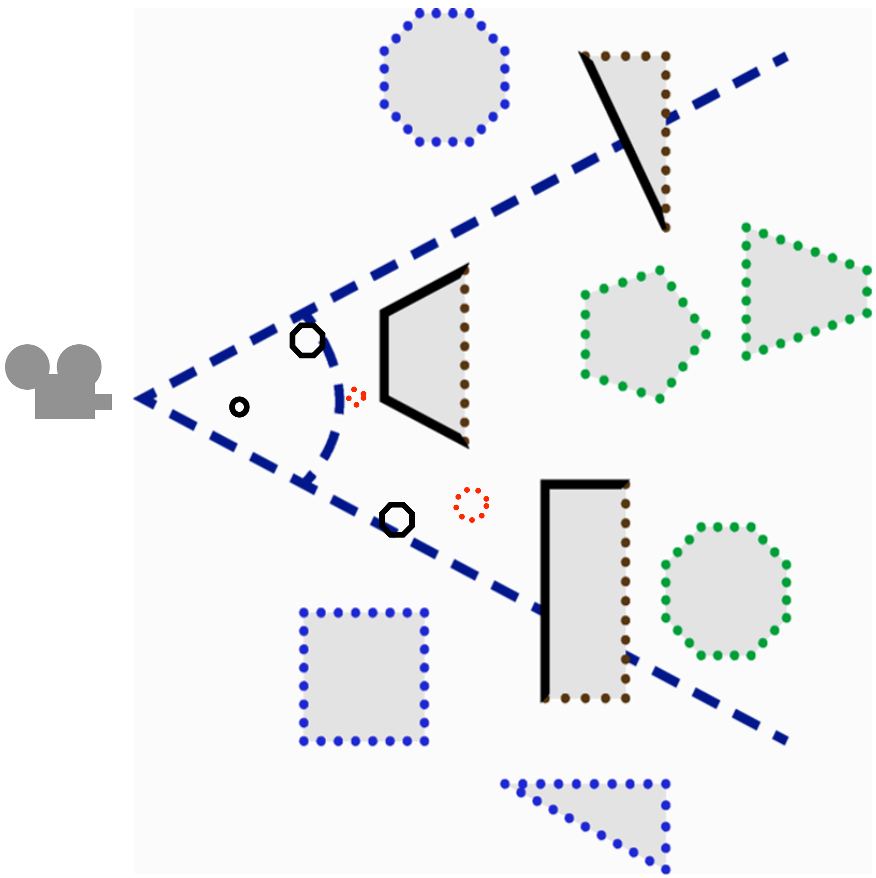
→ Detail culling, LOD

→ Occlusion culling

These polygons don‘t contribute visibly to the final picture, in general

Direct hardware rasterization doesn‘t handle these situations efficiently!!

Visibility Culling Methods



View-Frustum Culling

Occlusion Culling

Backface Culling

Detail Culling / LOD

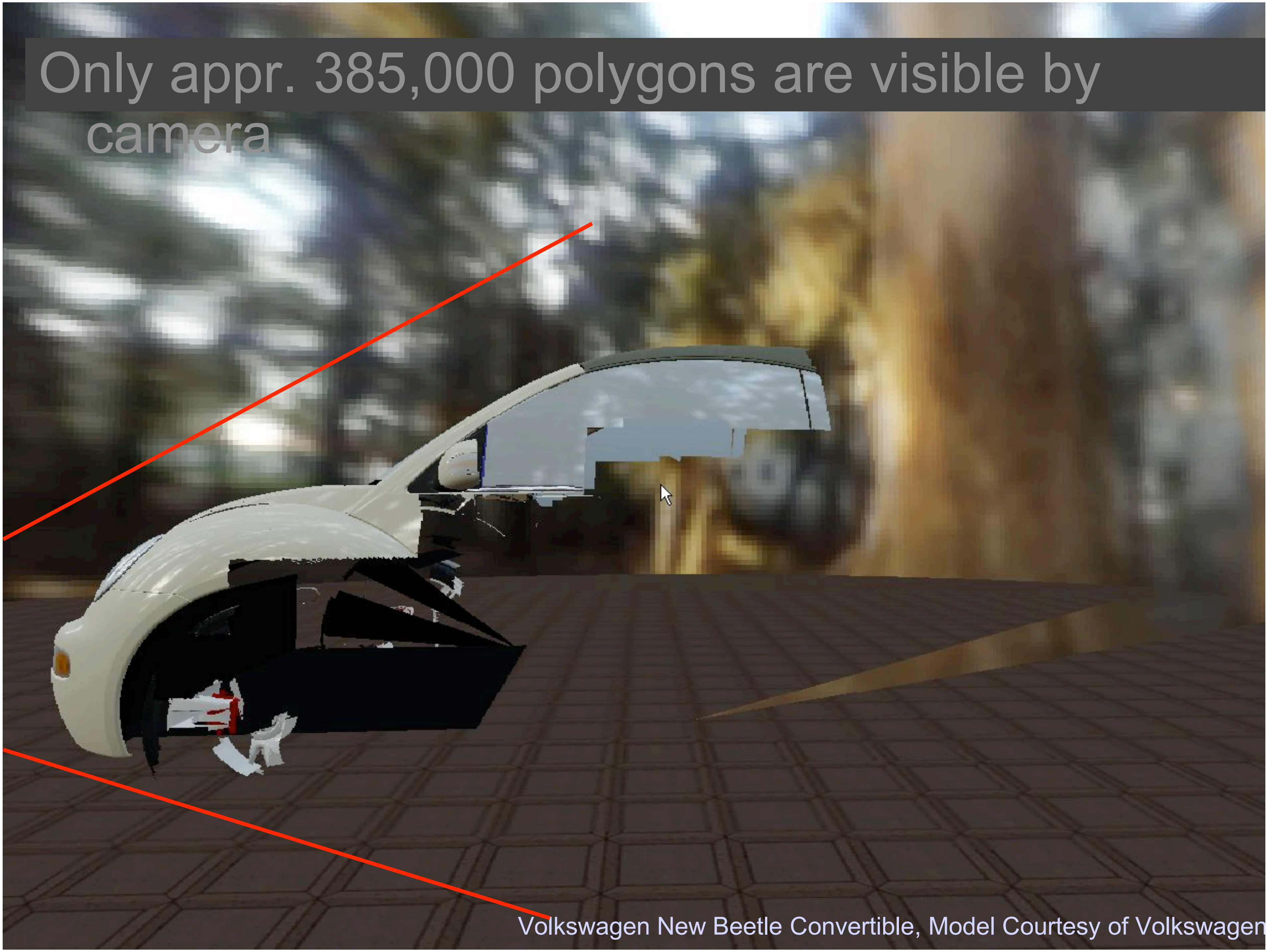
The VGR approach determines the visibility of polygons before GPU rasterization in real time, using new graphics hardware features and efficient data structures and algorithms

This drastically reduces the load on the graphics hardware

Example: Model contains 1,780,000 polygons

Volkswagen New Beetle Convertible, Model Courtesy of Vol

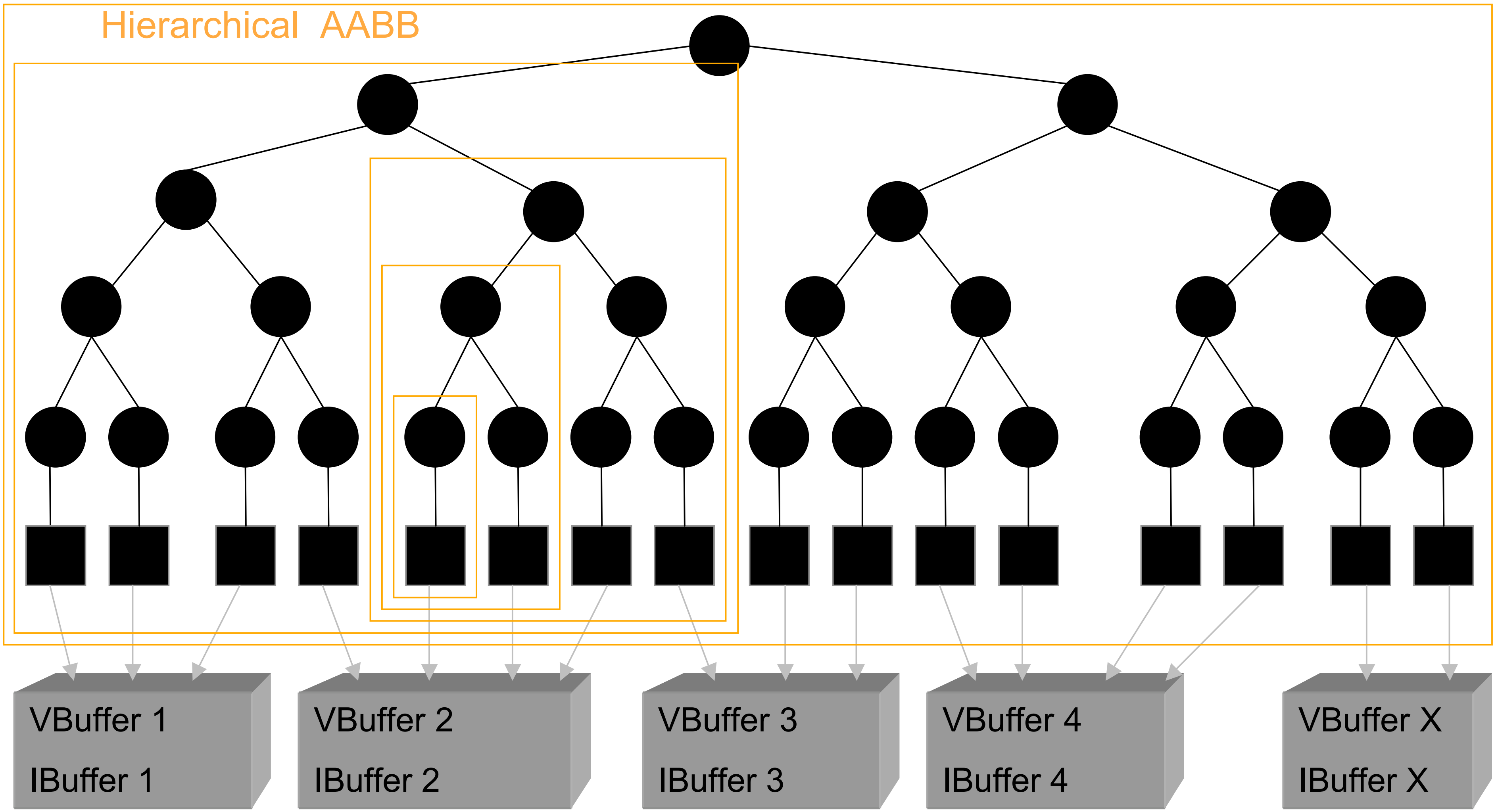
Only appr. 385,000 polygons are visible by camera




Occlusion culling: Approximately 1.4 Million polygons don‘t need to be rendered
Volkswagen New Beetle Convertible, Model Courtesy of Volkswagen AG

Spatial Tree Structure (kd-tree)

Binary Tree (split domain in x/y/z, periodically at each level)



Datastructure

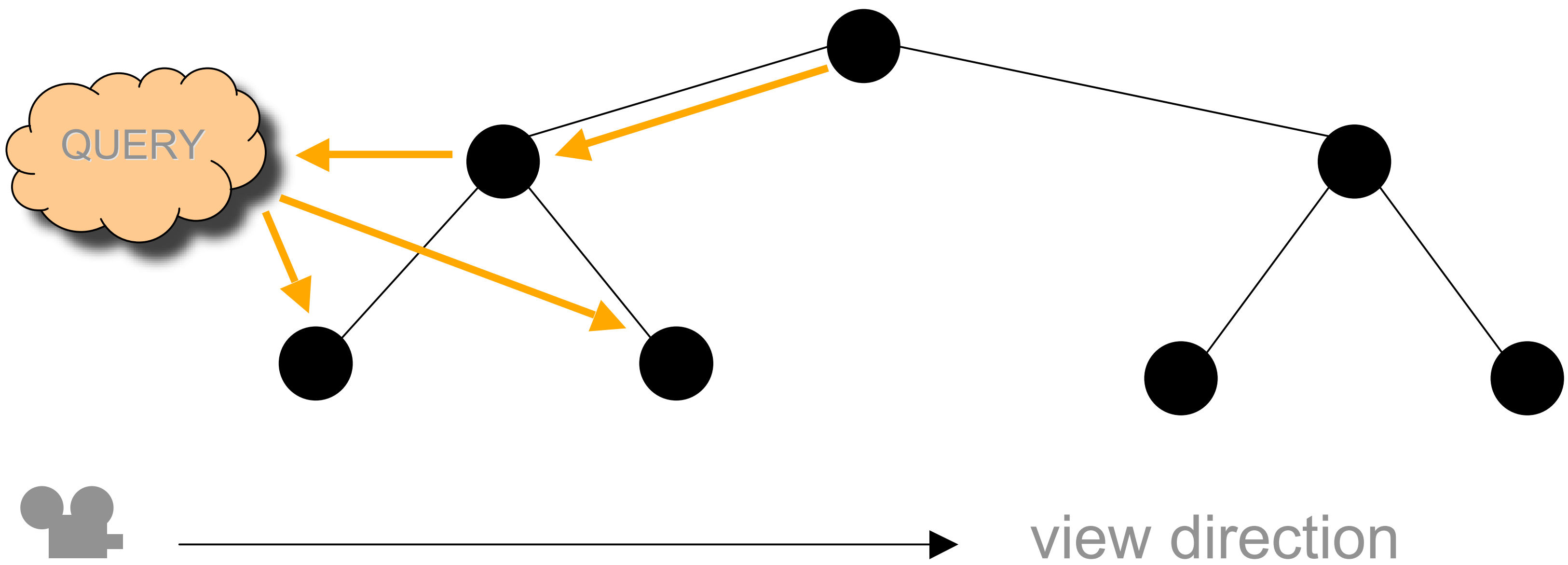
- kd-Tree + „loose“ AABB tree
- Almost as flexible as BSP tree
- Simple creation (like octree) but more adaptive
- Use:
  - Hierarchical occlusion culling with boundingboxes at nodes
    (binary relation) → Approximate front-to-back rendering
  - Hierarchical frustum culling (unary relation)
  - LOD, detail culling (unary relation) → point rendering

In the following we discuss occlusion culling (the most interesting part, because of object object interaction) in more detail

Hardware Occlusion Queries

- Test of an object for occlusion (*potential occludee*)
- Use simple bounding geometry
- Query should be significantly less work than rendering the object itself (several thousand polygons / BB)
- OpenGL 1.5: ARB_occlusion_query
- Hardware determines the number of visible pixels without writing to the z-/frame buffer
- Result of query available only after delay (latency)
- Premature querying of result causes pipe line stall

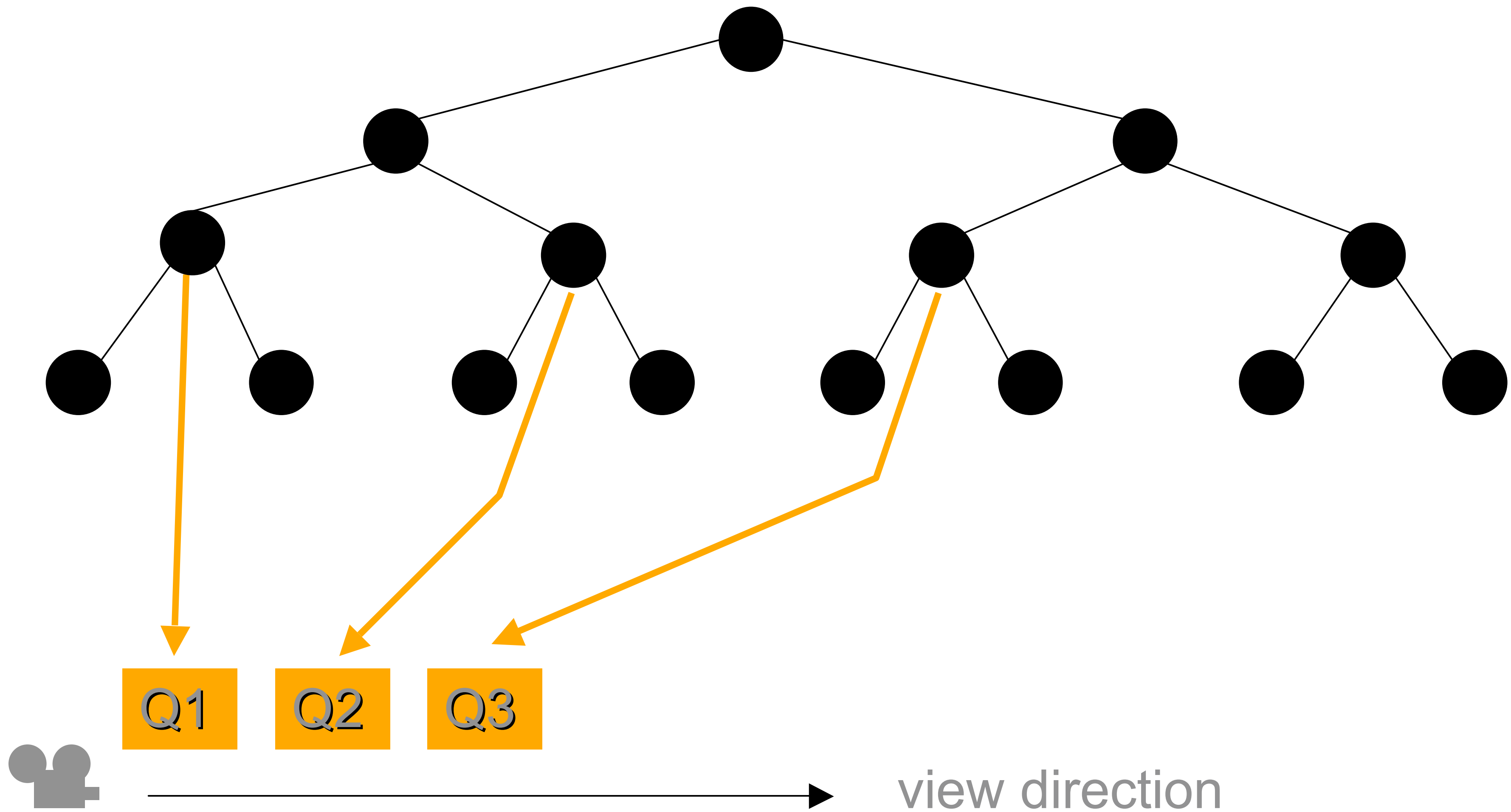
Latency vs. Traversal Order



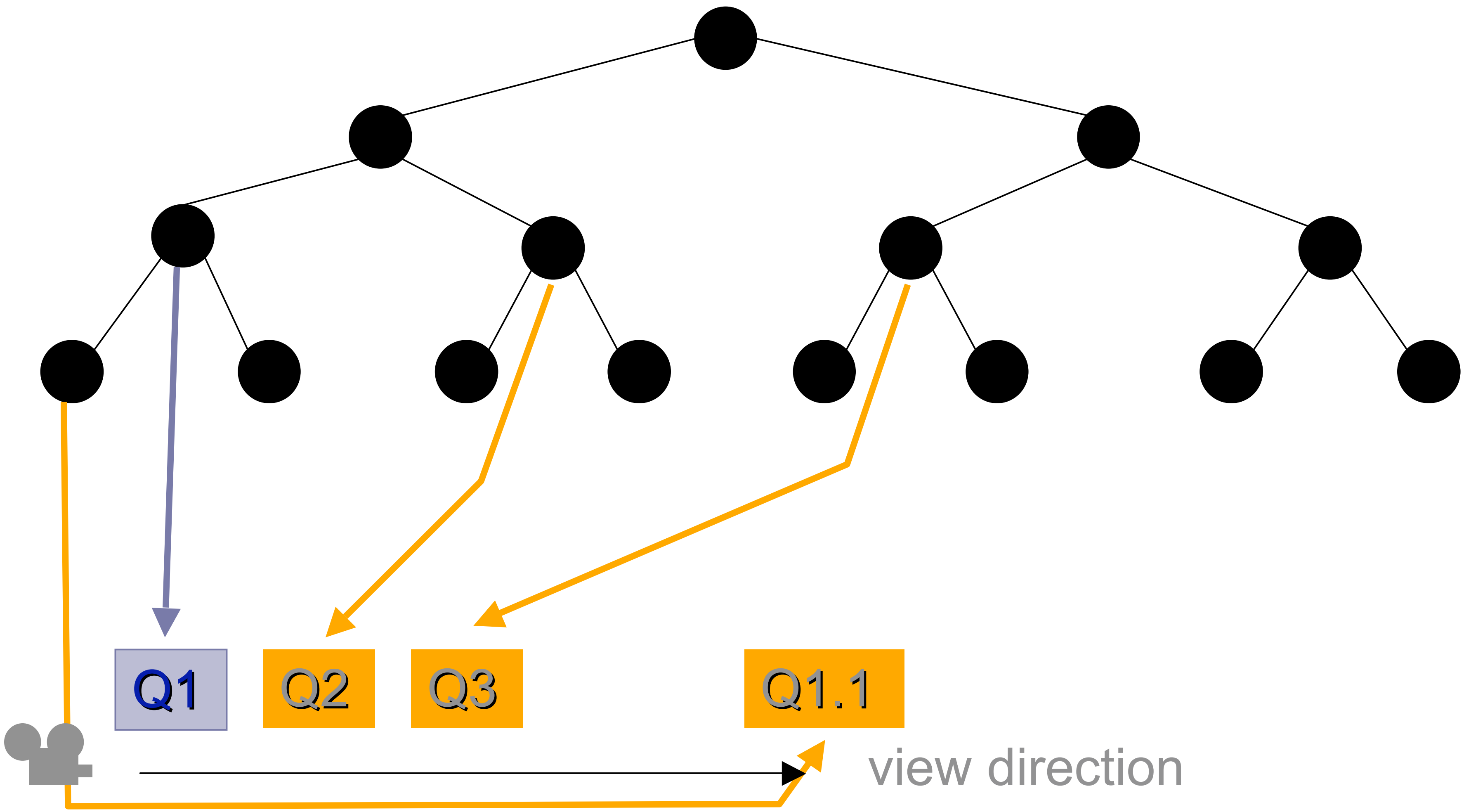
- Depth-first traversal of nodes (front to back & top down)
- Order depends on results of previous queries
- Synchronization between CPU and GPU → „stop and wait“

Query Queue

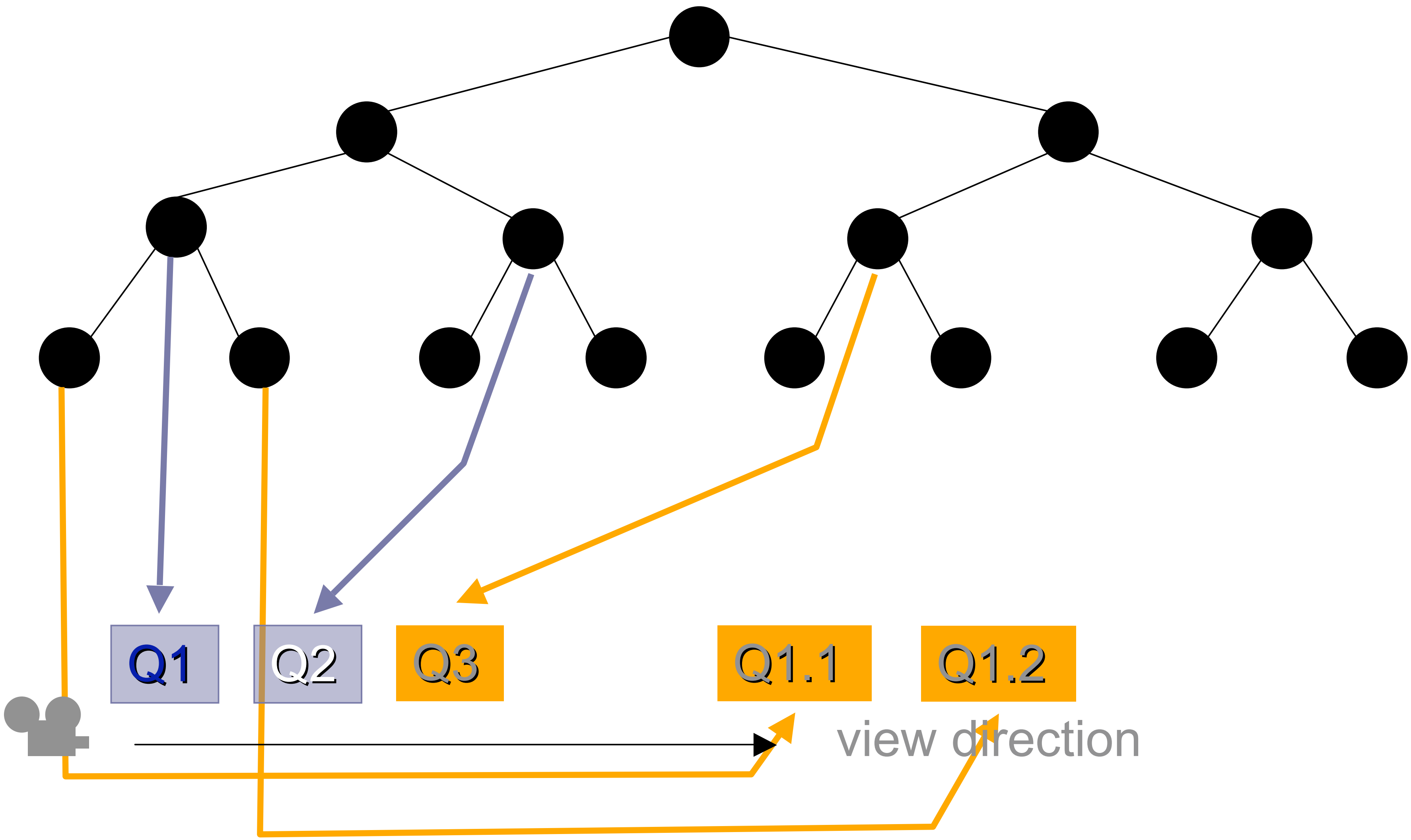
To avoid pipeline stalls, we introduce a query queue



Query Queue



Query Queue



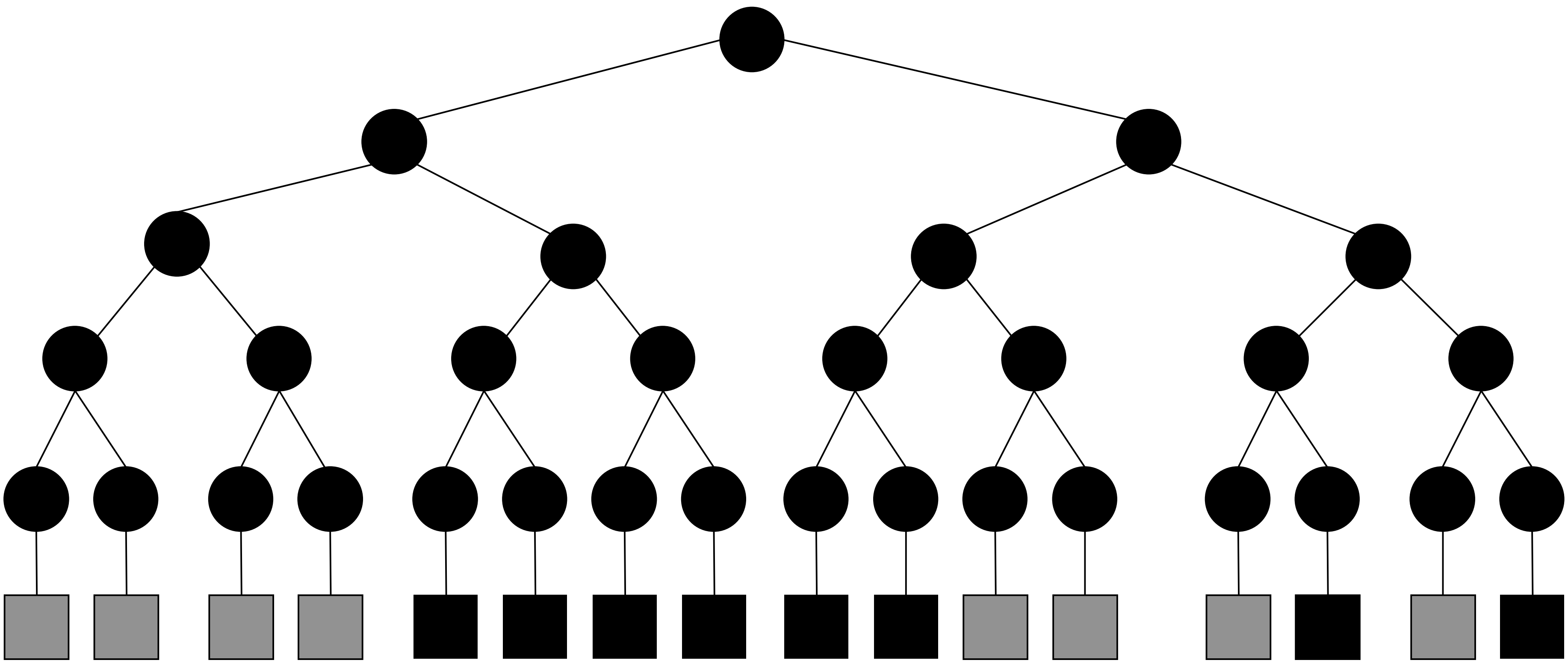
Queueing of Jobs and Queries

- Adaptation of query queue length to latency by asynchronous querying
- Ideally no wait for query results; there is always something to do.
- Problems:
  - No optimal queue length can be determined, due to changing latency
  - Strict top-down, front-to-back traversal is altered → more invisible triangles are handled unnecessarily (false positives)
- However, minimal number of rendered triangles is not necessarily the optimum, but also:
  - Minimize render calls
  - Avoid render pipeline stalls

Frame-to-Frame Coherence

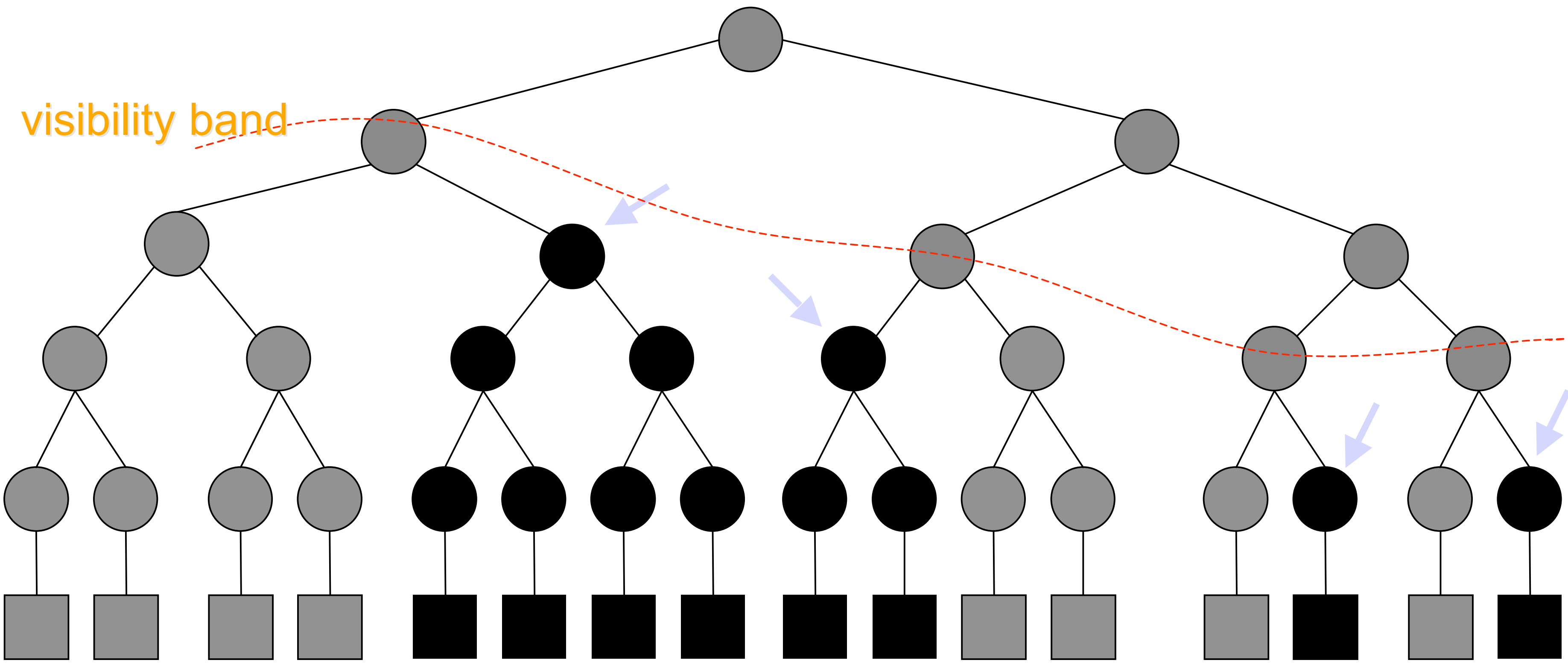
- Two consecutive frames contain largely the same visible objects.
- Exploit the z-sorted list of visible objects in frame *i* to initialize the z-buffer for frame *i + 1.*
- For every m-th frame carry out an occlusion query for every object on the list

Frame-to-Frame Coherence



- Invisible nodes:
- Visible nodes:
- Propagate the leaf visibility upward

Marking Nodes

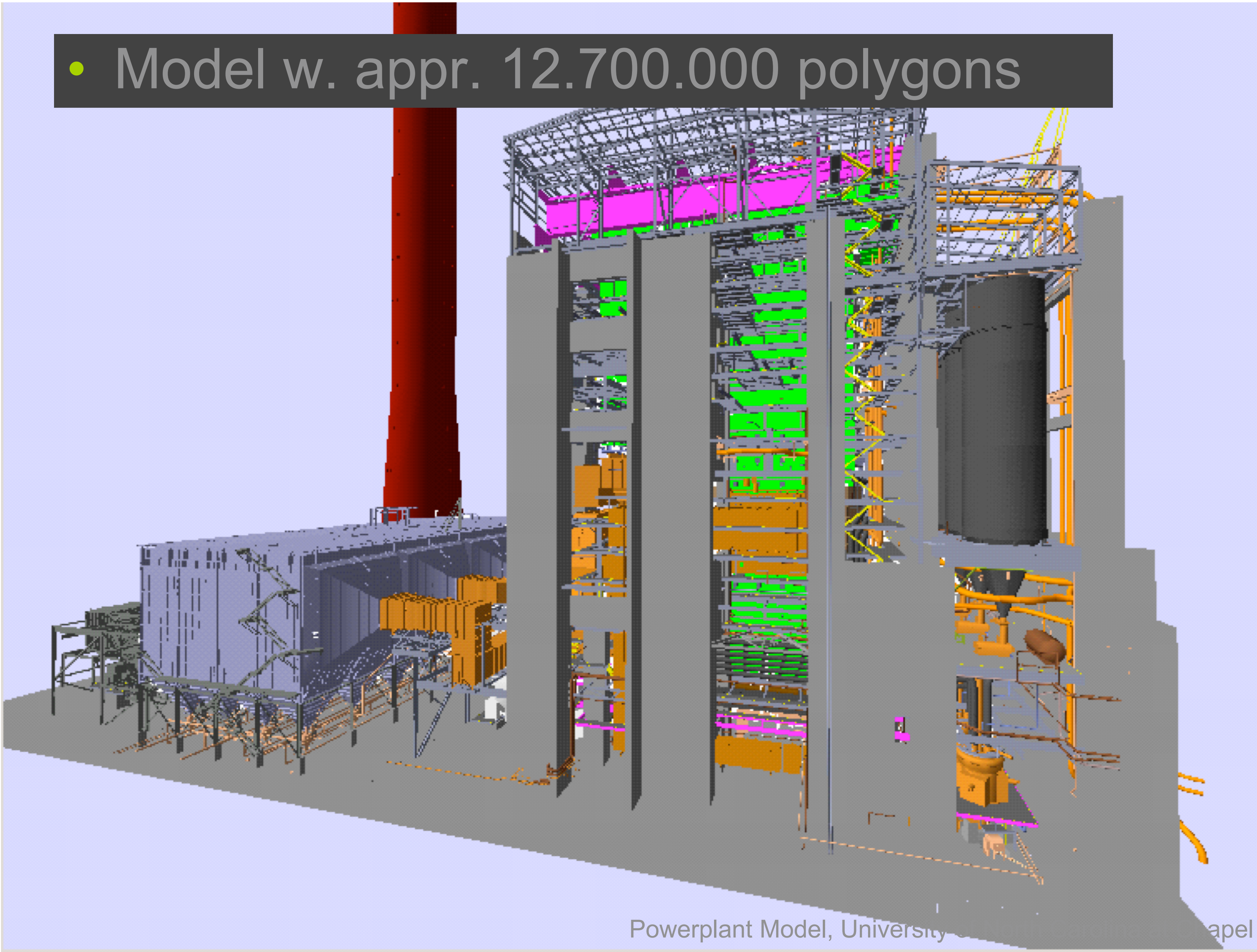


- Invisible nodes: ■
- Visible nodes: ■
- Only scene parts with unknown visibility are tested in every frame

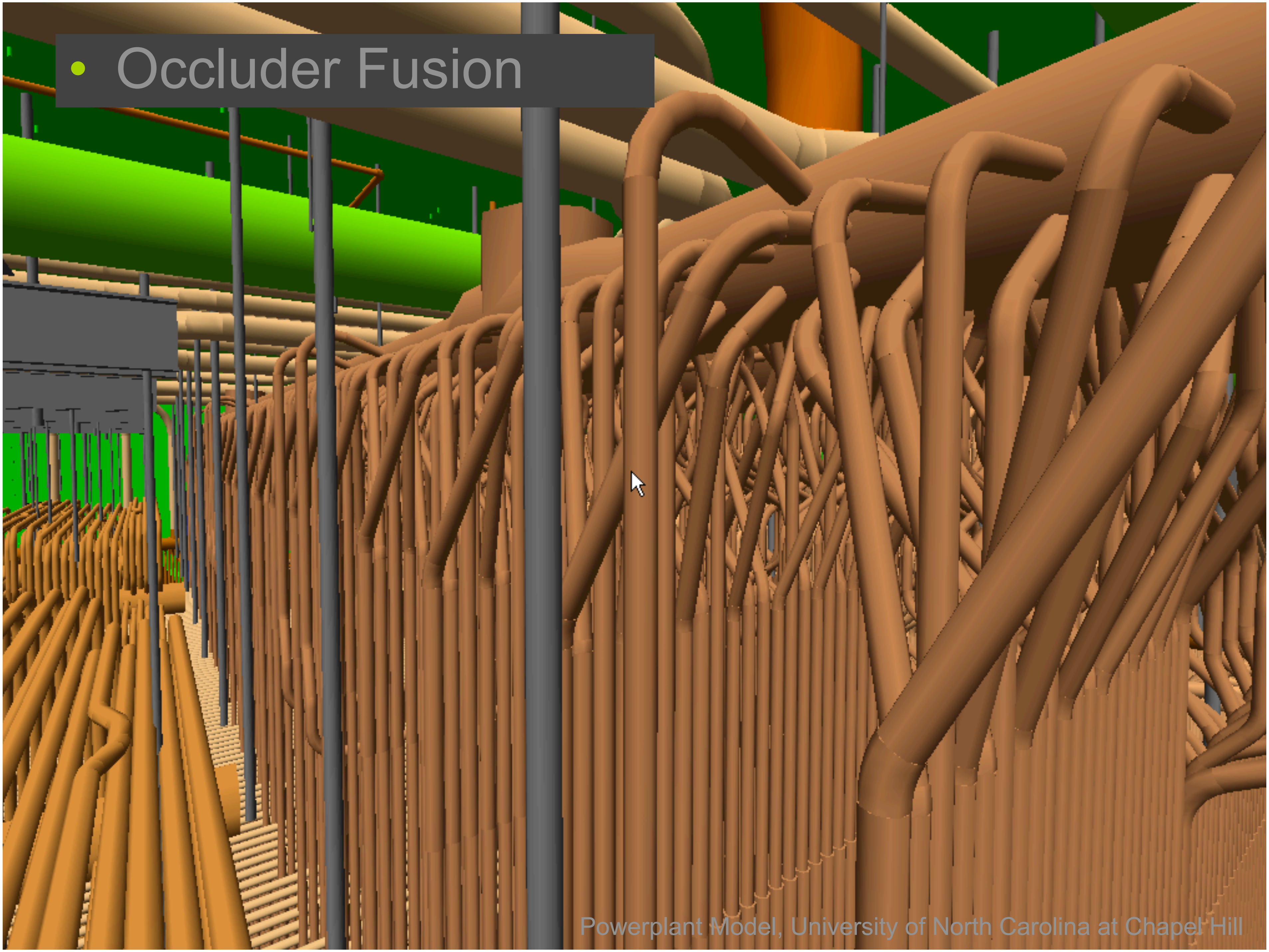
Marking Nodes

- Initial filling of the query queue with the root nodes of yet untested partial trees
  - Fewer but more useful queries
  - More stable prerequisites for the query queue
  - Significant performance gains

- Model w. appr. 12.700.000 polygons



- Occluder Fusion



Styling Lighting and Materials

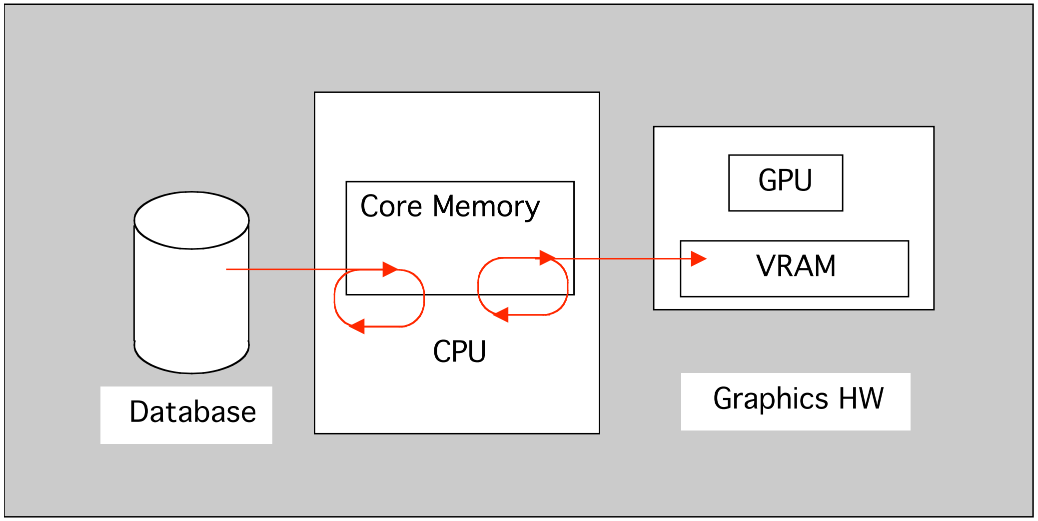
- Programmable Pixel and Vertex Shaders
- Image-based lighting
- Transparency
- HDRI, Reflections
- Blooming
- Tone Map, etc.

These techniques integrate well with visibility-guided rendering


GOLF
WOB VW 350

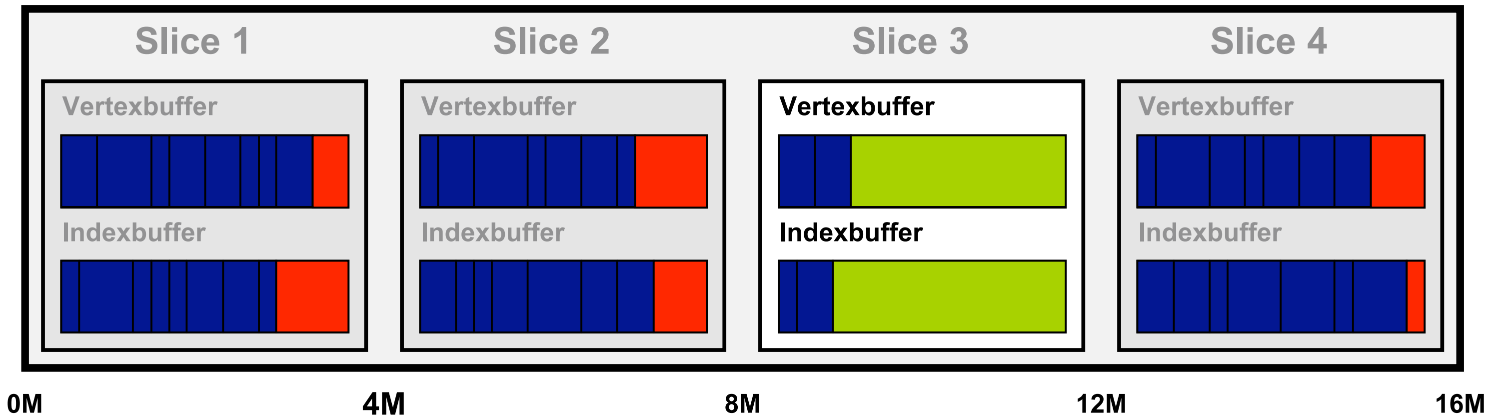
Out-of-core System Architecture

2-Level Caching:



- Spatial Database (10 GB – x TB)
  - Core Memory (500MB – x GB)
  - VRAM (64MB – 512MB)
- Out-of-core rendering
- Prefetching mechanism to avoid lag time
- Off-line clash detection

Subdivision of Graphics Memory

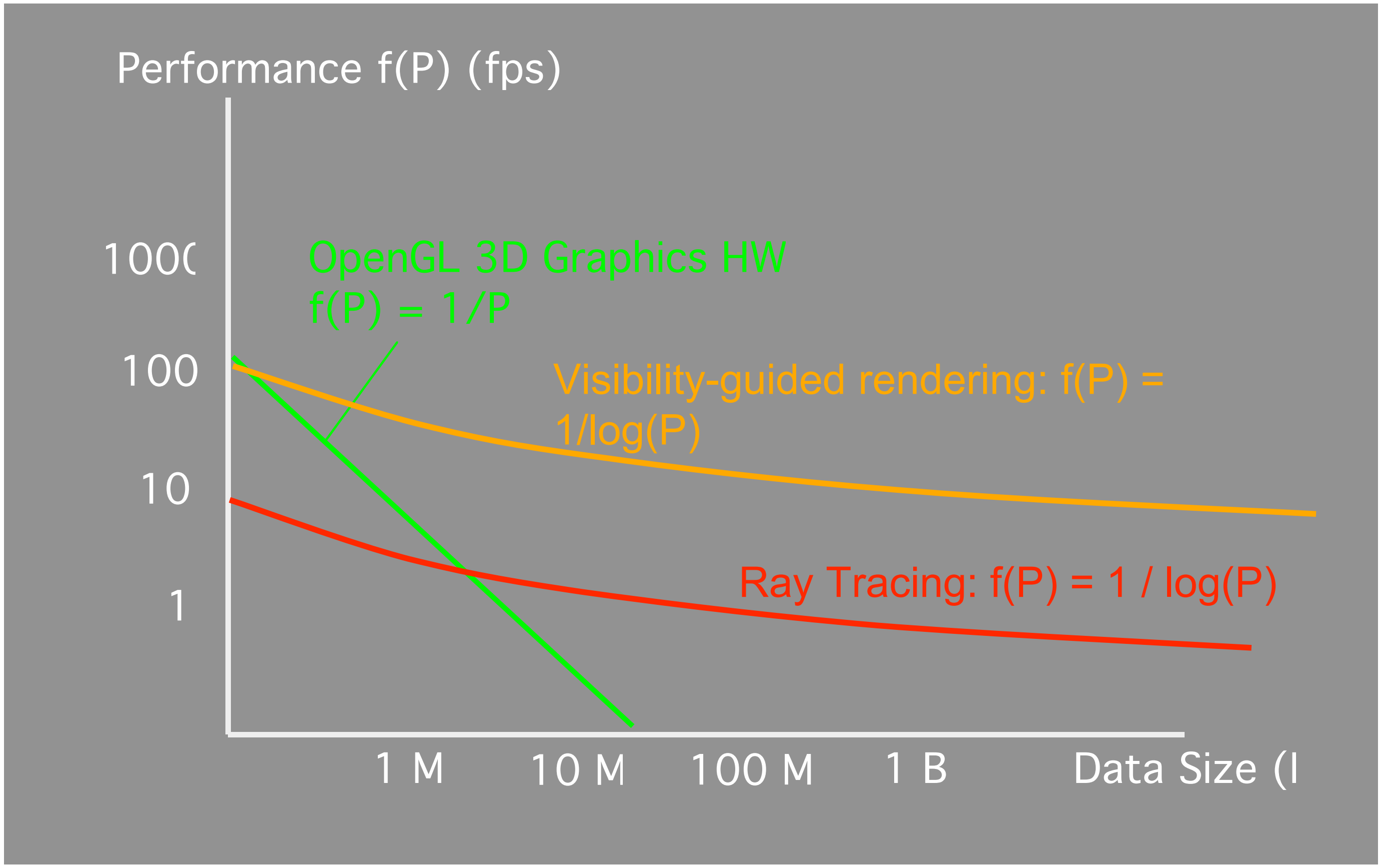


- LRU as replacement strategy of complete slices
- Maintaining data coherence
- Fewer buffer swaps, but „dead storage“

Prefetching from HD

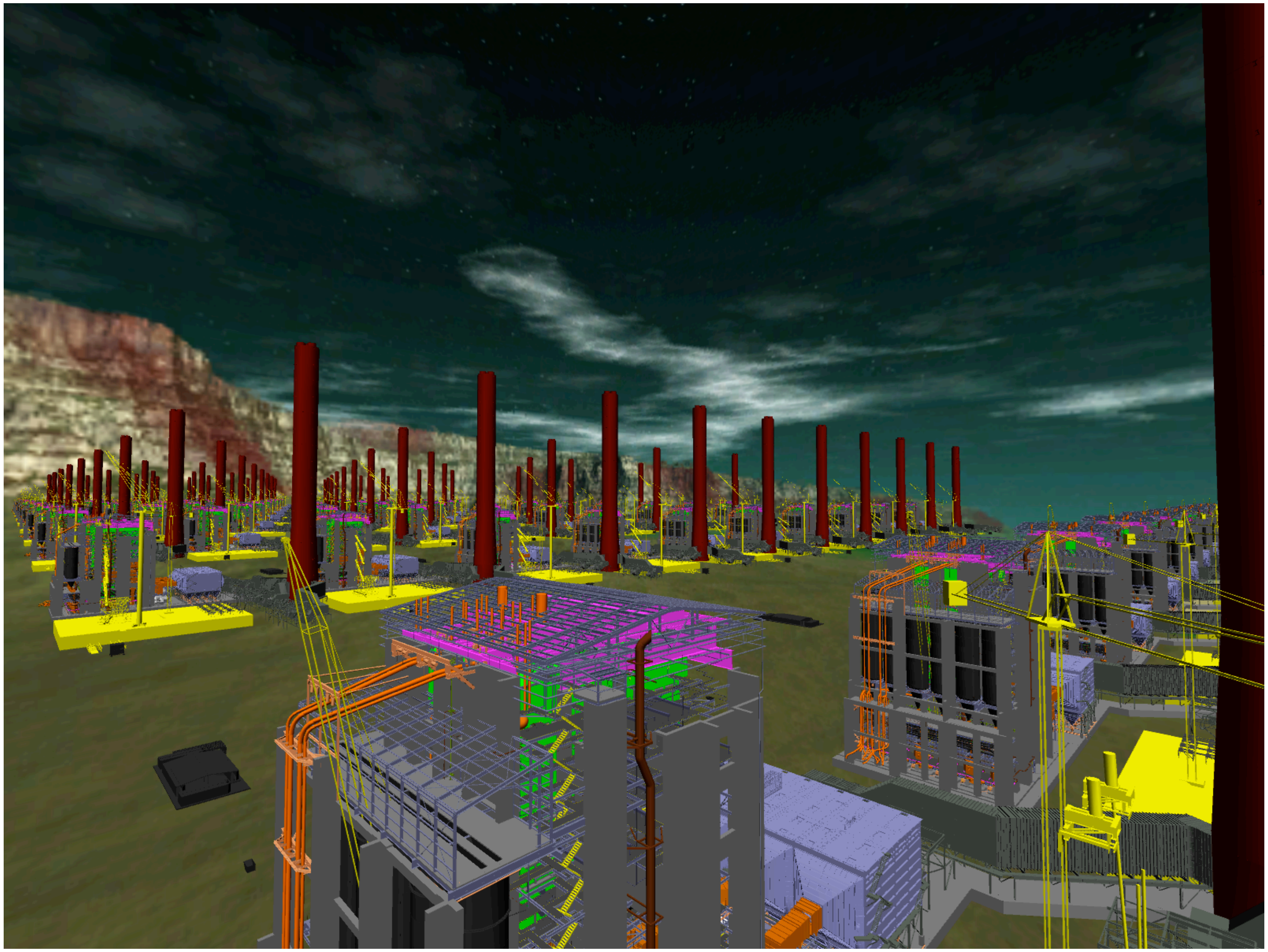
- In each frame, there is potentially a different distribution of priorities. However, due to the temporal coherence from frame to frame, only a few changes need to be applied simultaneously.
- The prefetching-and-replacement strategy tries to keep as many leaf nodes as possible in main memory, starting at the leaves with the highest priority.
- Leaves with lower priority that have stayed in memory for the longest time will be removed from the systems memory first
- Leaf nodes marked as “visible” will never be removed from memory, to avoid flickering.

Performance Comparison



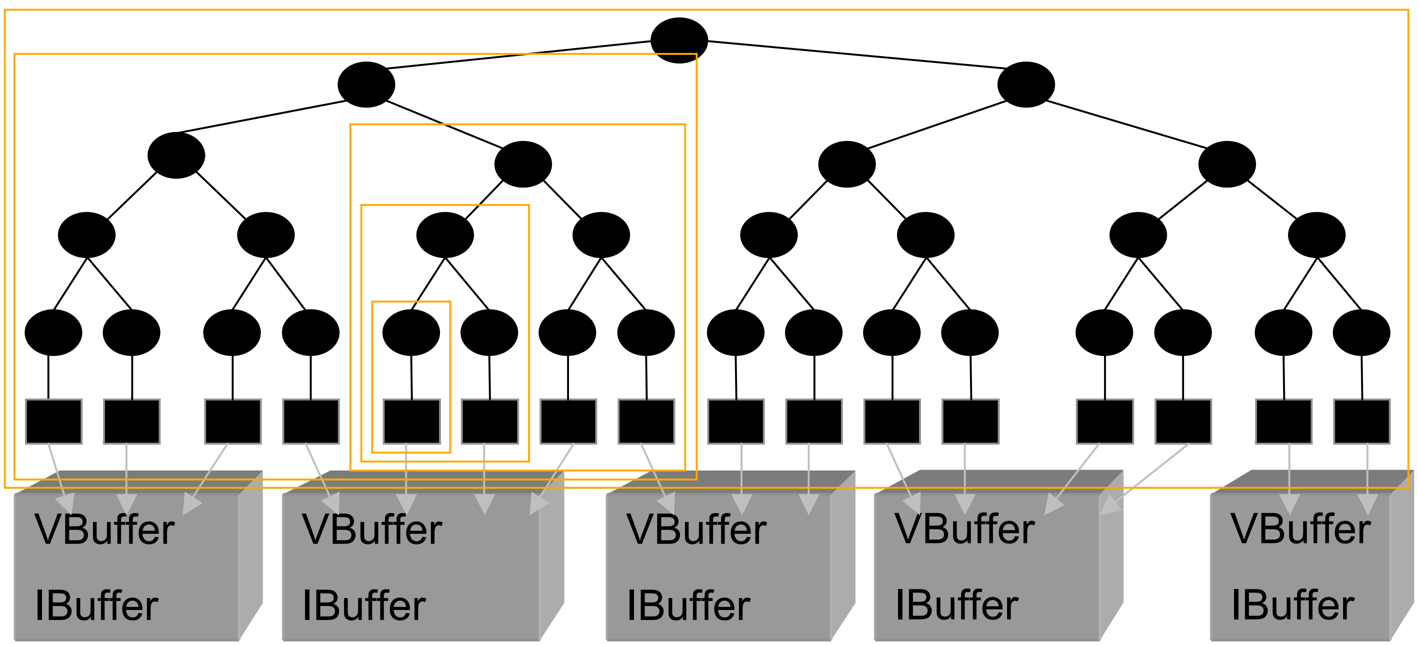
Average PC (2005) 3GHz CPU, 3D graphics card, 500MB memory and 1M pixel display, direct Phong lighting, (no reflections, shadows, etc.).

Scene with 1.300.000.000 Polygons



Preprocessing Data

- Goal: Hierarchically and spatially subdivide scene into sets of 1000 to 8000 triangles
- Sort these sets as leaves of a kd-tree.
  → Spatial Sorting (no complex pre-processing required)
- Needs to be done out of core:
- Examples of pre-processing times:
  – Power plant (12,7 Mio. triangles) in 1.5 min
  – Boeing 777 (350 Mio. triangles) in 70 min.



Two memory management strategies

- Memory mapped files &
- Explicit swapping of geometry chunks

Memory mapped files

- Make use of optimized swapping strategy of the operating system:
- Principle: The virtual address space of a process is subdivided into pages (usually 4 kB in size) which can reference pages in main memory, external devices, or on the hard disk.
- Linear view on all data simultaneously. Access via 64-bit indices. Requirement of the necessary sizes in megabytes.
- Storage may be requested and modified by means of defined windows on the data that are mapped to systems memory (memory-mapped files)
- Memory can be read or written transparently. Altered pages will automatically be written to the hard disk.
- By only using one storage window at a time, the swapping of data is completely managed by the operating system, which reduces fragmentation of the address space.

Problems of memory mapped files:

- Swapping strategies of the operating systems are kept general and are not optimal for the specific application case.
- Swapping is done only when the system memory fills up and memory requests can no longer be met.
- For instance under MS-Windows only single pages are written out and immediately afterward other pages are read back in. This leads to inefficient I/O behavior.
- The linear view does not fit well with the hierarchical organization of the geometry. This requires continuous copying to maintain coherence.

Explicit Memory Management

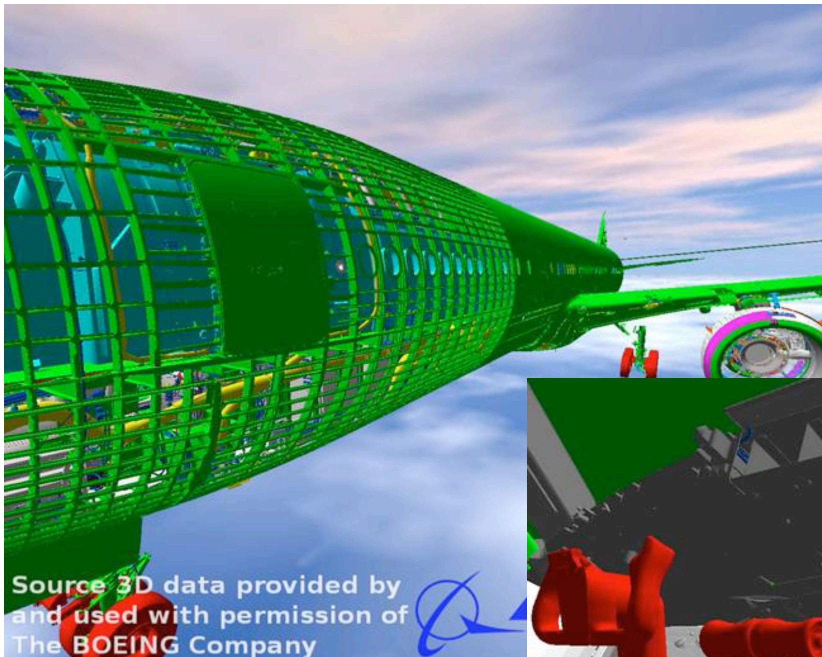
- Considers the hierarchical subdivision of data
- Explicitly controls read and write actions.
- For every node of the tree, the complete sub tree data needs to be read, handled and written back to disk

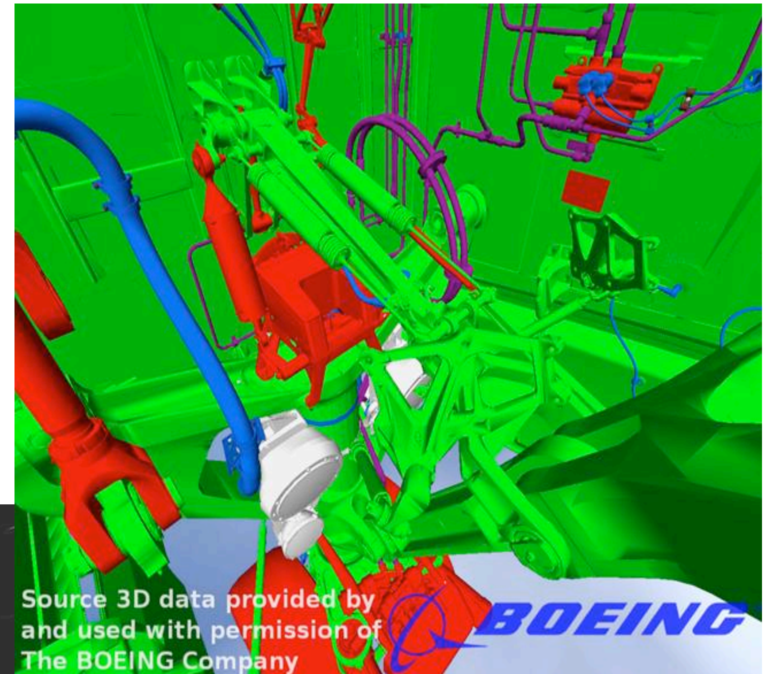
  →amounts to the complete data set once per level of recursion)
- If the necessary data is smaller than the available systems memory, the complete leaf data can be handled in core.
- Otherwise, the subdivision has to be carried out in stages, where only a subset what fits into memory is handled simultaneously.

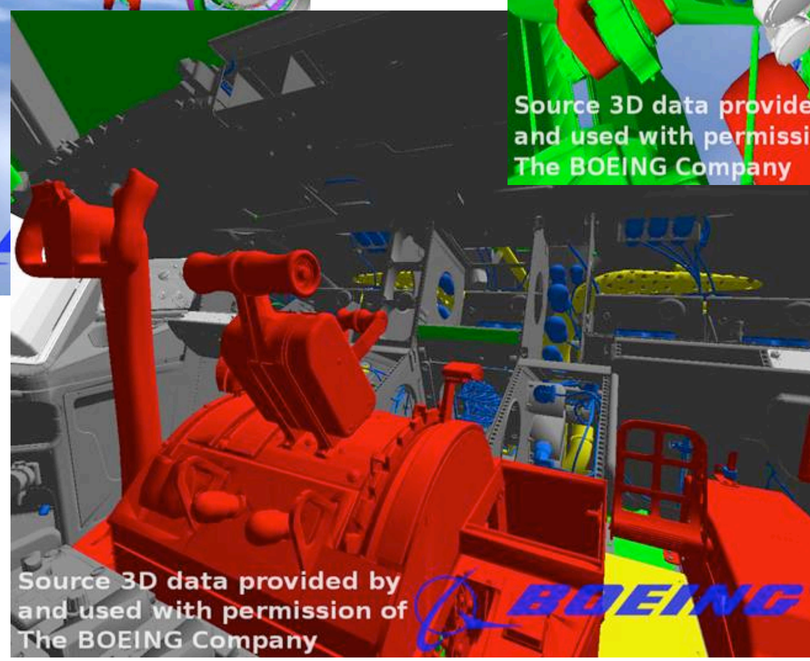
Advantage over the memory-mapped file approach:

- Subdivision of geometry data into chunks that can be handled completely in core.
- Swapping more efficient 50MB/s vs. only 1 – 5 MB/s with mmf
- Acceleration of pre-processing by up to a factor of 10.
- Asynchronous writing of data during generation.

Engineering Example: Boeing 777

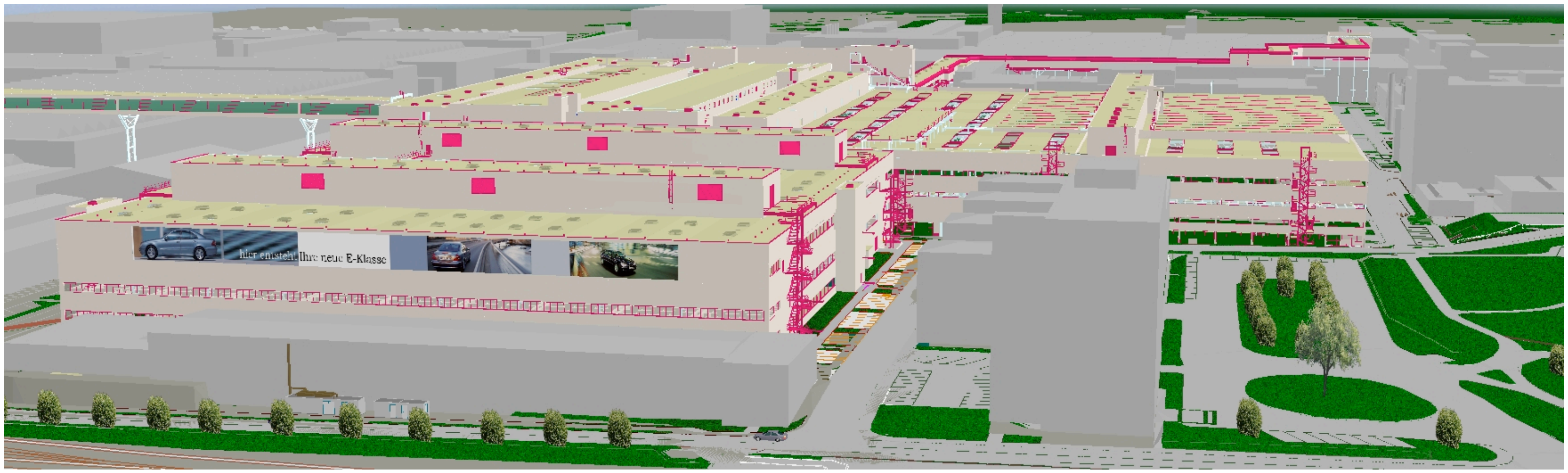




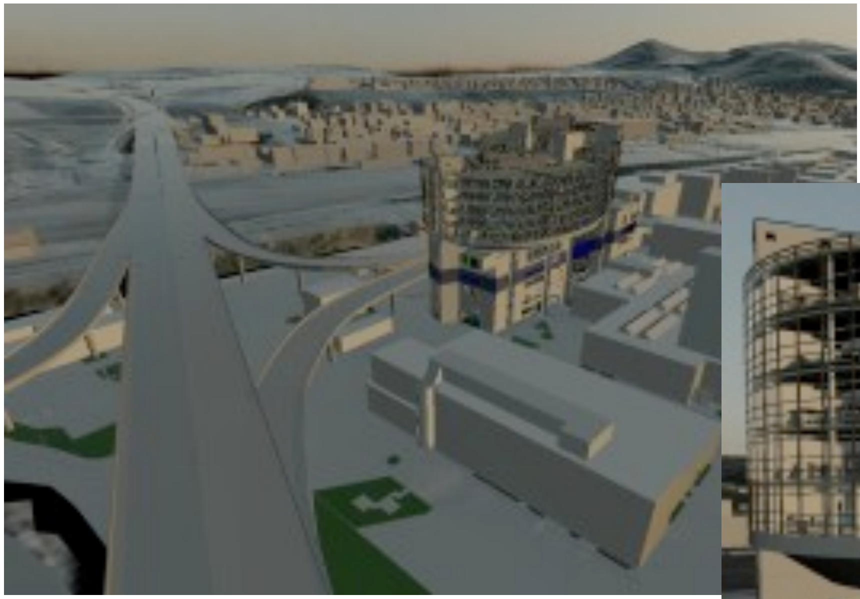


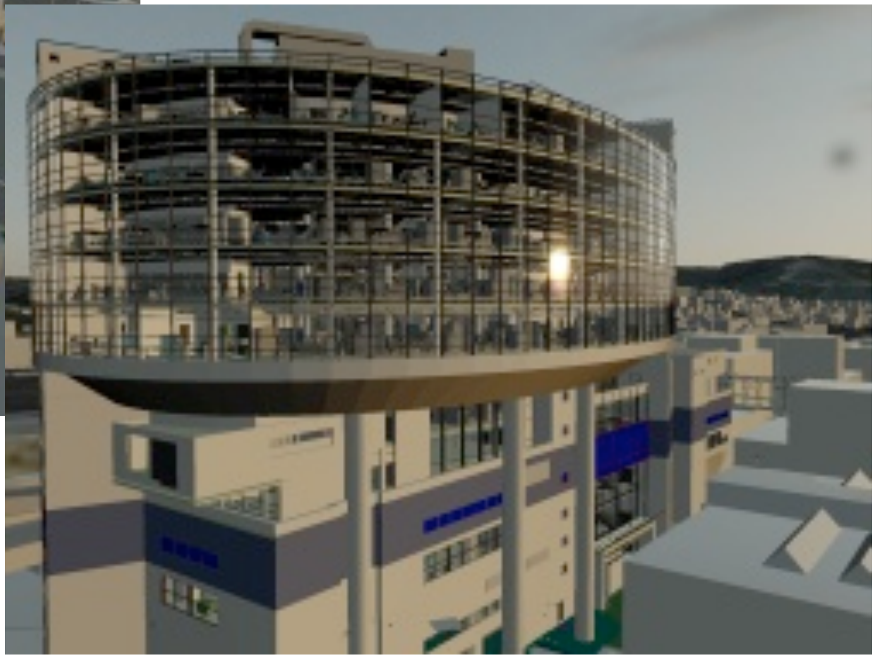
Extremely detailed CAD Model: - 350 Million Triangles in Real Time

Veo:Factory: DaimlerChrysler



DaimlerChrysler VTC
Van Technology Center





Satellite Data + CAD Model (modeled with Bentley MicroStation)


GOLF
WOB VW 350

Conclusion and Outlook

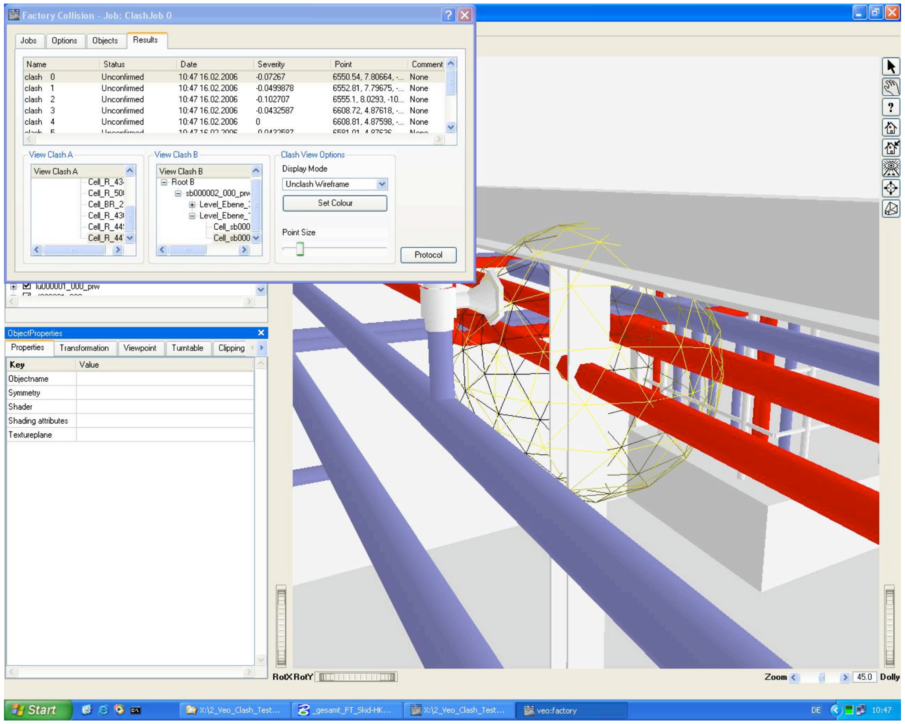
Advantages of **inter**views3D‘s Visiblity-guided Rendering:

- Only one set of data! Utilize simultaneously for
  - Engineering (DMU) and Styling
  - Render quality and quantity
- No more need to simplify models!!
  - Visualize large, detailed models directly from CAD
  - Save time and money
- Scalability: One software system for
  - PC workstation, laptops
  - Render server for power wall presentations

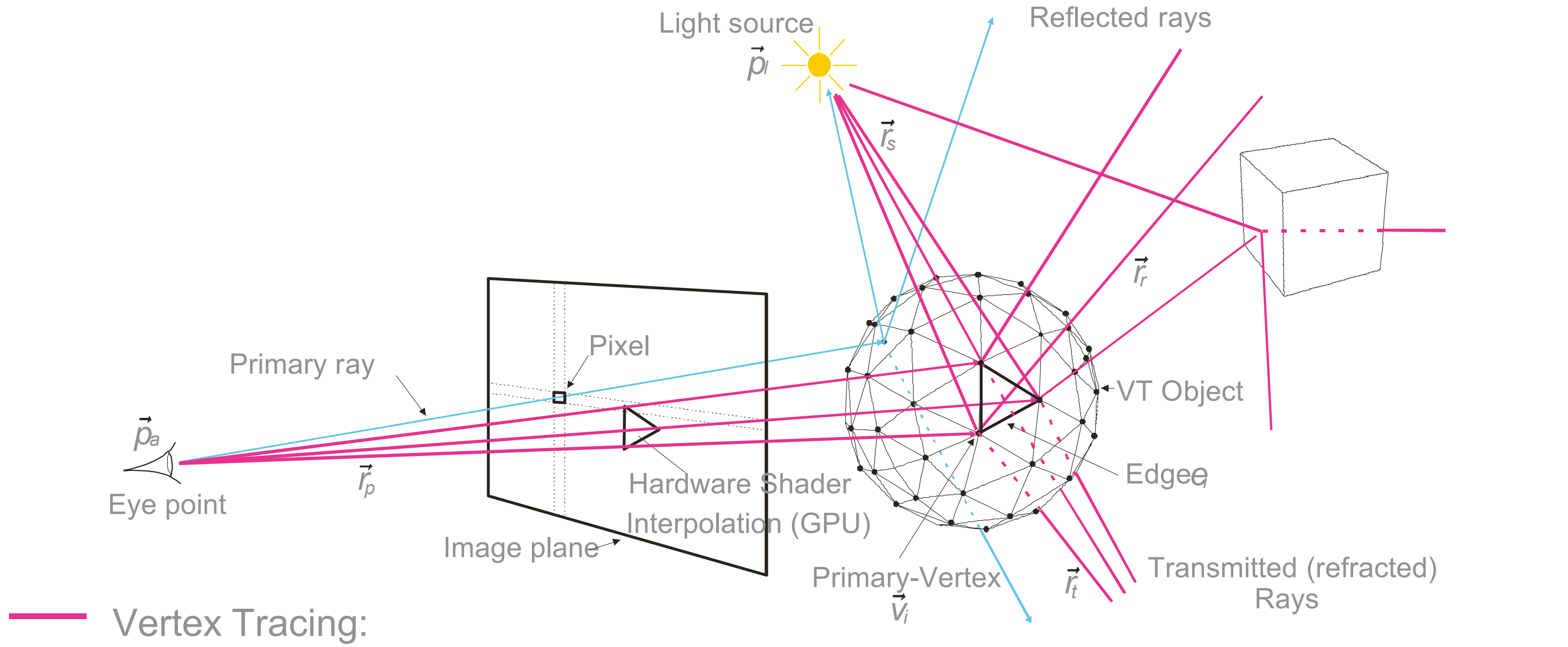
Conclusion and Outlook

- Support of PLM (product life cycle management) – selective updates & version
- Clash detection (Veo:Factory):
  - DMU tool for factory planning
  - Graphical User Interface
  - Power Wall Presentation:

    3000 x 1200 Pixels, Stereo (4 Projectors)



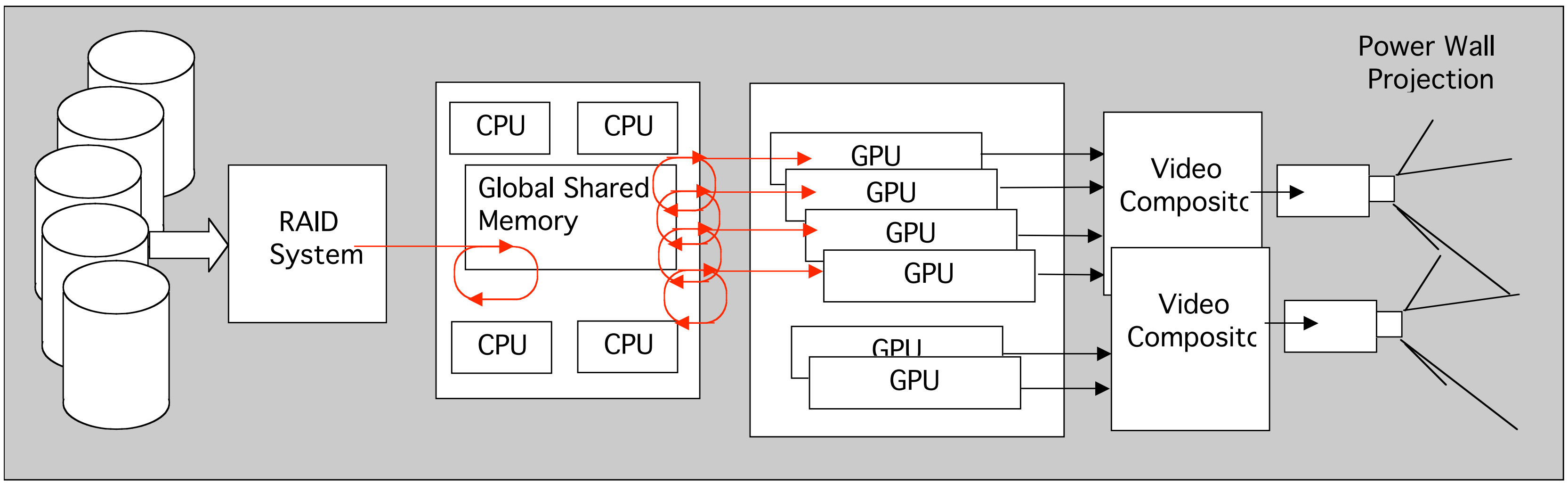
Vertex Tracing



- No primary ray intersection
- Graphics hardware for reconstruction (pixel- vertex shaders)
- Only visible objects generate secondary rays (use VGR)

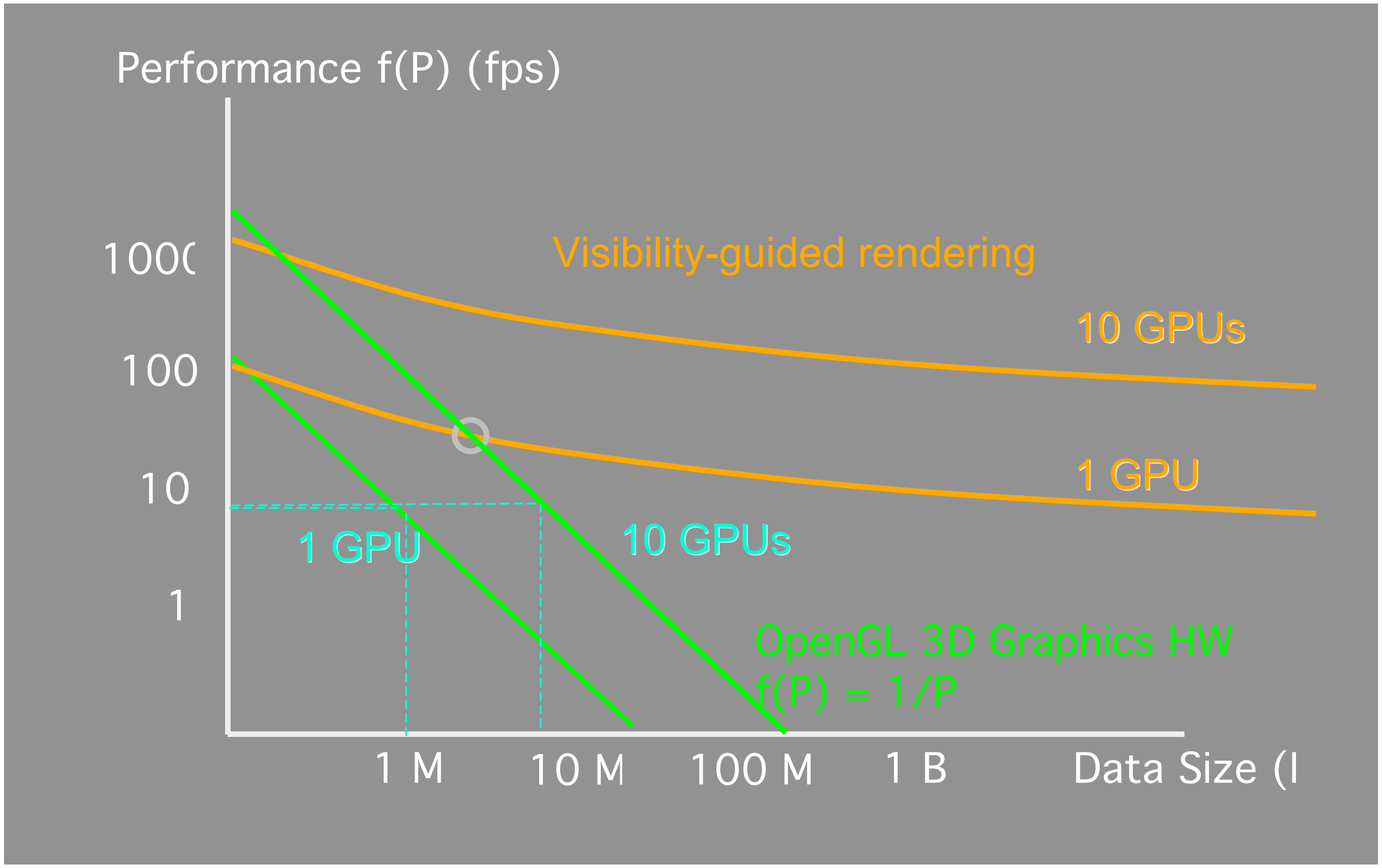
Scalability

Graphics Server SGI Prism: Multi CPU, multi GPU, global shared memory, high bandwidth RAID



Performance gain for visibility-guided rendering?

Theoretical Scalability of Multi-GPU Systems



Already with one GPU, Visibility-Guided Rendering VGR is faster than direct hardware rendering witb 10 GPUs.

However: VGR also scales appr. 10 times with 10 GPUs

Scalabilitiy in Performance?

- Not necessary to render more polygons! (frame rate depends little on data size)
- However, high-end hardware power enables:
- Higher frame-rate through multi-GPU:
  - Time sequential / AFR (linear speed-up, some latency)
  - Tiling mode / SFR (near linear speed-up, no latency
- High performance, global shared memory: e.g. 20 GB can load entire Boeing 777 into core memory
- Fast RAID system for even larger models: Navigate through terabites of data directly from database without much latency

Graphics Server Advantages:

- Additional performance for improvement of quality at consistently high frame rate:
  - Sophisticated and complex vertex and pixel shaders for special materials and lighting effects
  - Materials: E.g. brushed metals, leather, coating, etc.
  - Hardware anti-aliasing

Higher Frame Rates With Realistic Rendering:

- HDRI: High dynamic range imaging, for realistic light reflections
- Soft shadows, indirect lighting in real-time
- Real specular object inter-reflections (refractions) through hybrid, real-time ray tracing (multi-CPU)

Thank You!

For further information, please contact:

3D interactive

www.3dinteractive.de

info@3dinteractive.de