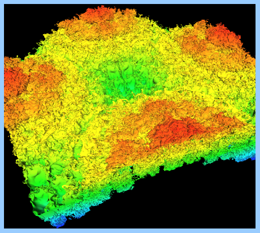
Interactive Out-Of-Core Visualization of Large Datasets on Commodity PCs



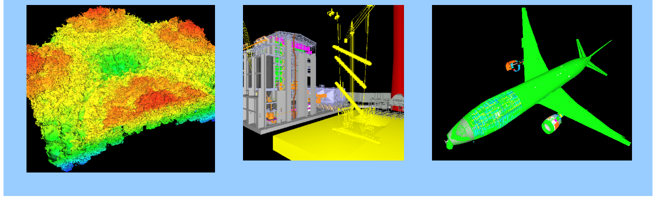
Wagner Corrêa
Research Staff Member
IBM Watson Research Center

Goal

- Interactive visualization of large datasets on inexpensive PCs
  - interactive: 10 or more frames per second
  - large: larger than main memory
  - inexpensive: under $2,000 per PC

Motivations

- Large datasets have many applications
  - CAD
  - modeling and simulation
  - virtual training



Motivations (cont.)

- PCs are good alternative to high-end workstations
  - better price/performance
  - easier to upgrade

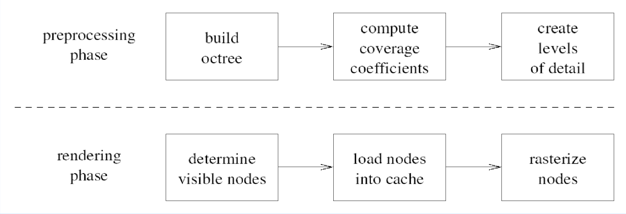
Challenges

- Datasets are larger than main memory
- High I/O latency and low I/O bandwidth
- Only one graphics pipe per PC
- Low screen resolution

Solutions

- Out-of-core preprocessing algorithms
  - spatialization, visibility precomputation, and simplification
- Out-of-core rendering algorithms
  - approximate visibility and prefetching
  - hardware-assisted conservative visibility
- Out-of-core parallel rendering algorithms
  - rendering on multi-tile screen using PC cluster

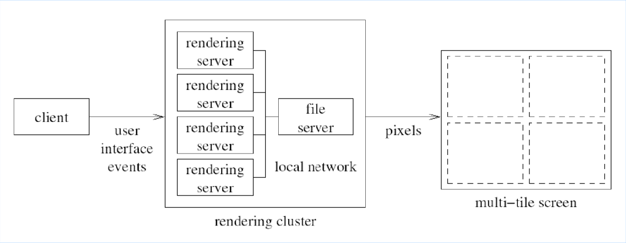
Preprocessing and Rendering



preprocessing phase
build octree
compute coverage coefficients
create levels of detail

rendering phase
determine visible nodes
load nodes into cache
rasterize nodes

Parallel Rendering



client
user interface events
rendering server
rendering server
rendering server
rendering server
file server
local network
rendering cluster
pixels
multi−tile screen

Talk Outline

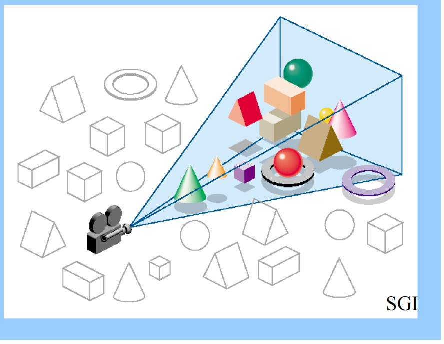
- Out-of-core preprocessing
- Out-of-core rendering
- Out-of-core parallel rendering
- Conclusions

Out-Of-Core Preprocessing

- Build an octree
  - Hierarchical frustum culling
  - Working set management
- Compute visibility coefficients
  - Occlusion culling
  - Prefetching
- Create simplified versions
  - Level-of-detail control

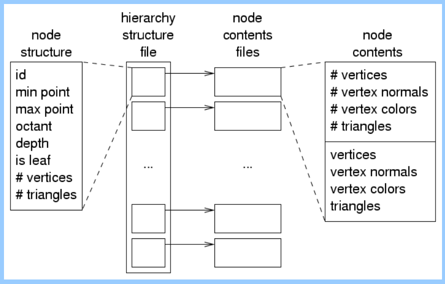
View-frustum Culling

- Clark76



SGI

Building an Octree



node structure
id
min point
max point
octant
depth
is leaf
vertices
triangles

hierarchy structure file

node contents files

...

...

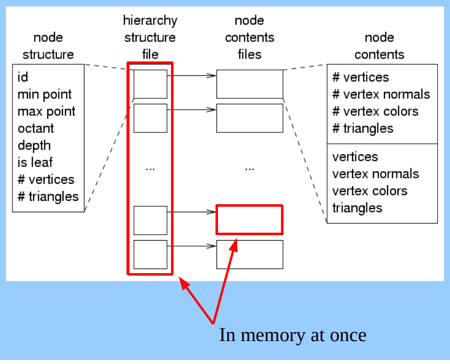
node contents
vertices
vertex normals
vertex colors
triangles

vertices
vertex normals
vertex colors
triangles

Building an Octree

- Break model in sections that fit in memory
- For each section
  - read hierarchy structure (HS) file
  - perform fake insertions
  - for each touched node
    - read old contents
    - merge old + new
    - update contents on disk
  - update HS file on disk

Building an Octree



In memory at once

Advantages of Our Spatialization Algorithm

- Out-of-core
  - we need memory for the section, the HS file, and the contents of one leaf
- Incremental
  - only updates regions touched by the section
  - important for 3D scanning
- Efficient
  - only reads a modified node once per section

Computing Visibility Coefficients

- For each node, for each viewing direction
  - compute coefficient:
    projected area of data/projected area of bbox
- Used to determine node priority at runtime

Detail Culling

- Avoid rendering unimportant details
- Also known as level-of-detail management
- LOD switching approaches
  - based on distance from viewer
  - optimized (Funkhouser93)
    - maximize image-quality (benefit)
    - given time and geometry constraints (cost)
  - based on visibility information

Creating Levels of Detail

- Several static LODs per octree node
  - uses vertex clustering [Rossignac and Borrel 93]
  - limitations: popping, different levels between adjacent nodes
- Possible improvements:
  - dynamic LODs (slower, less suitable for HW)
  - hysteresis (don't switch LODs too often)

Advantages of Vertex Clustering

- Fast and robust
- Only needs to traverse the data once
- Produces good enough approximations
- Has an intuitive, user-controlled accuracy dial
- Does not need topological adjacency graph

Preprocessing Tests

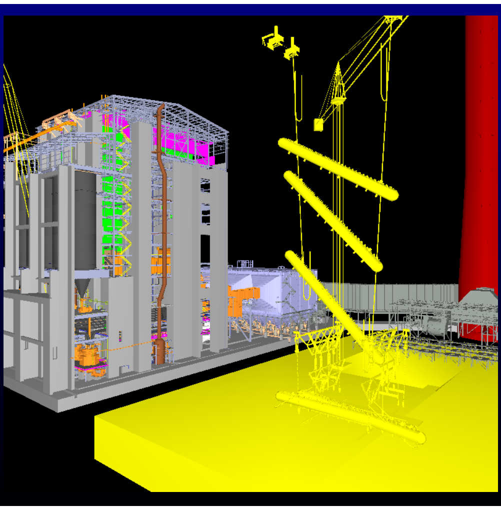
- Measure time to preprocess datasets
- Study tradeoff between spatialization granularity and octree size
- Assess quality of approximations

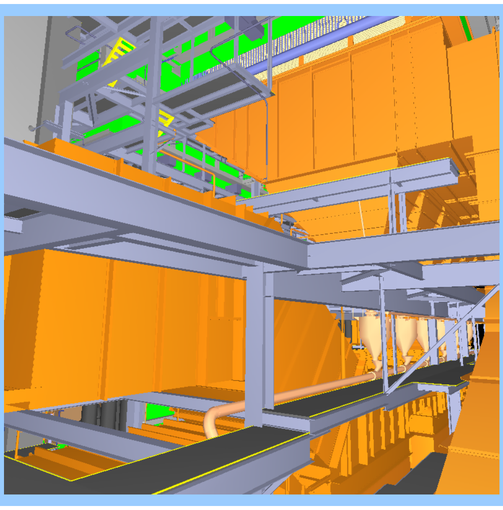
Test Datasets

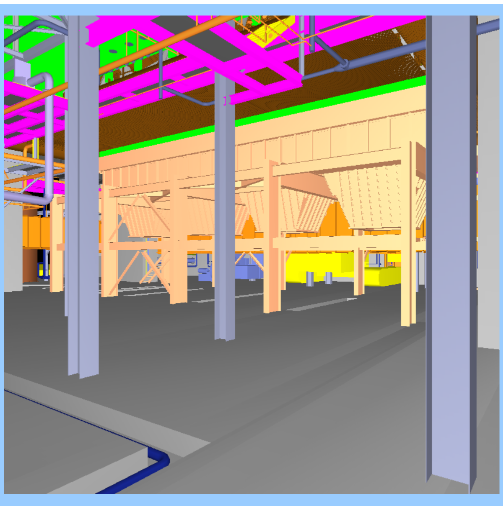
- UNC power plant
- LLNL isosurface
- Boeing 777

UNC Power Plant

- CAD model
- 13 million triangles
- High depth complexity
- 363 MB of raw data
- 1GB after preprocessing

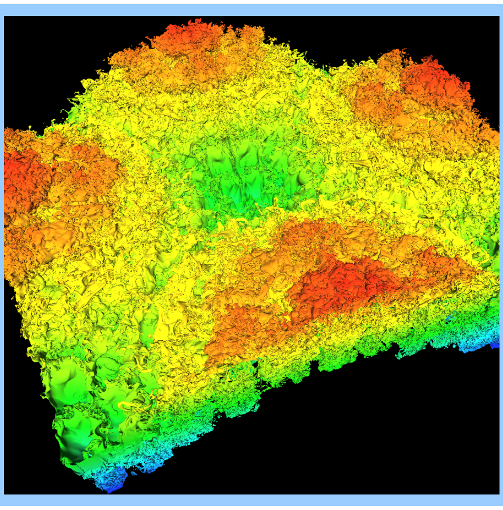






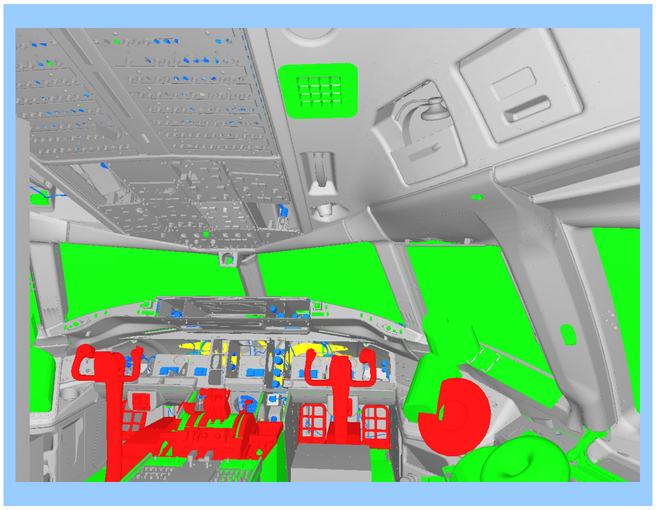
LLNL Isosurface

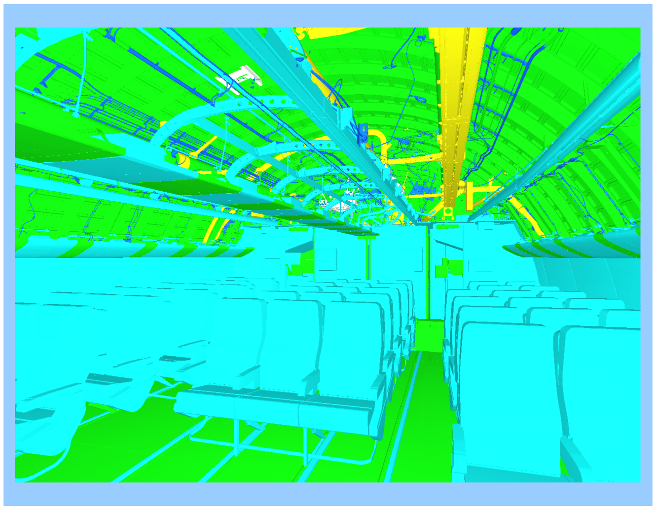
- Isosurface of turbulent boundary between two mixing fluids
- 473 million triangles
- 10GB of data



Boeing 777

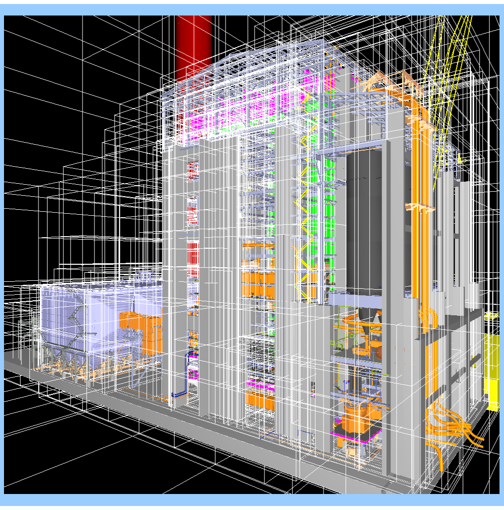
- CAD model
- 13,525 parts
- 352 million triangles
- 5GB of data





Test Machine

- 2.4 GHz Pentium IV
- 512 MB RAM
- 250 GB IDE disk
- NVIDIA GeForce Quadro FX 500 graphics
- Red Hat Linux 8.0
- Cost: about $1,000

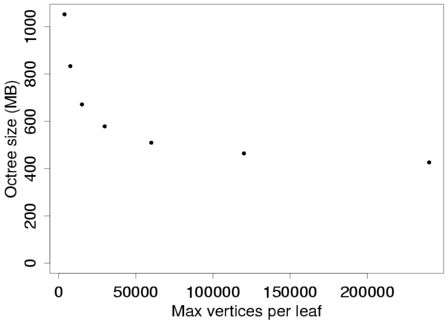


Power Plant Results

- Effect of spatialization granularity

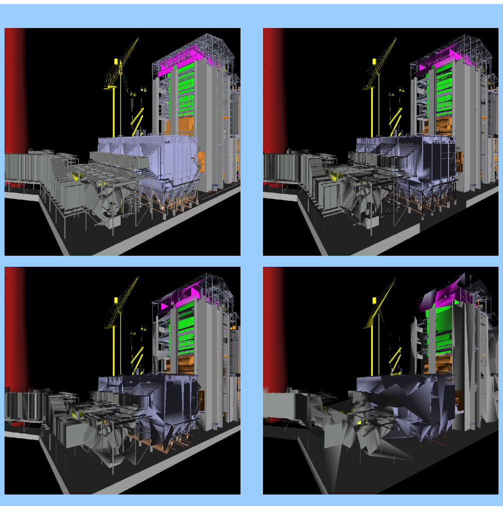
Max vert/leaf	Build time	Size (MB)	Depth	Leaves	Nodes	Triangles
3750	10m 03s	1052	11	72,416	82,761	30,461,154
7500	7m 51s	833	11	33,944	38,793	25,985,206
15000	6m 24s	671	10	15,177	17,345	22,073,219
30000	5m 17s	578	9	6,847	7,825	20,088,458
60000	4m 45s	510	9	3,354	3,833	18,301,106
120000	4m 16s	465	8	1,744	1,993	17,509,750
240000	3m 57s	426	8	701	801	16,215,938

Power Plant Results



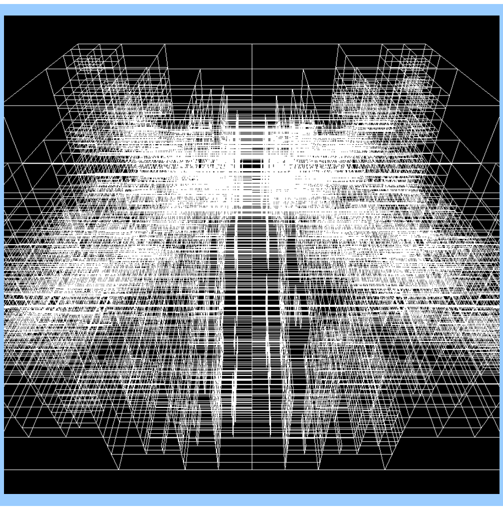
Power Plant Results

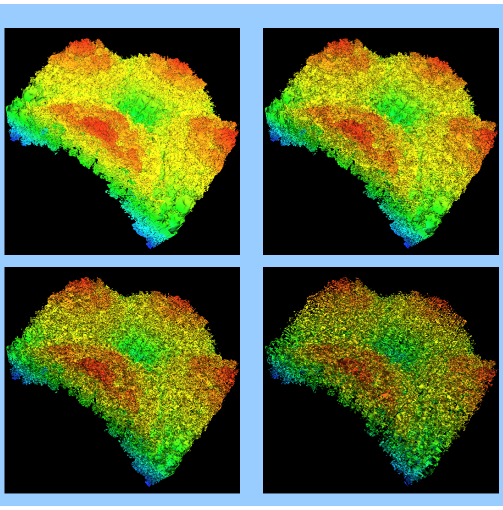
- Octree (15,000 triangles per leaf)
  - 6m 24s, 15,177 leaves
  - 3.4 MB for structure, 671 MB for data
- Visibility coefficients (20 dirs, 64x64 window)
  - 2m 36s, 711KB
- Levels of detail (up to 5 levels, 1/4 each time)
  - 8m 5s, 268 MB
- Total: about 17m and 1GB of data



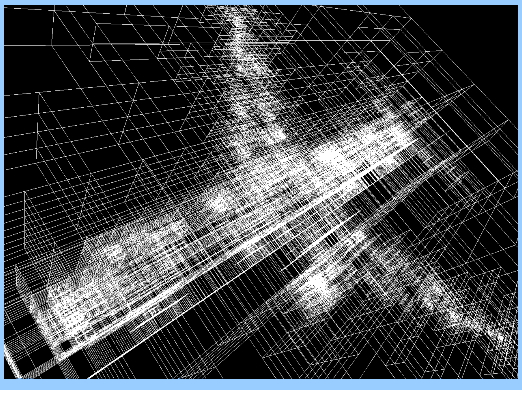
LLNL Isosurface Results

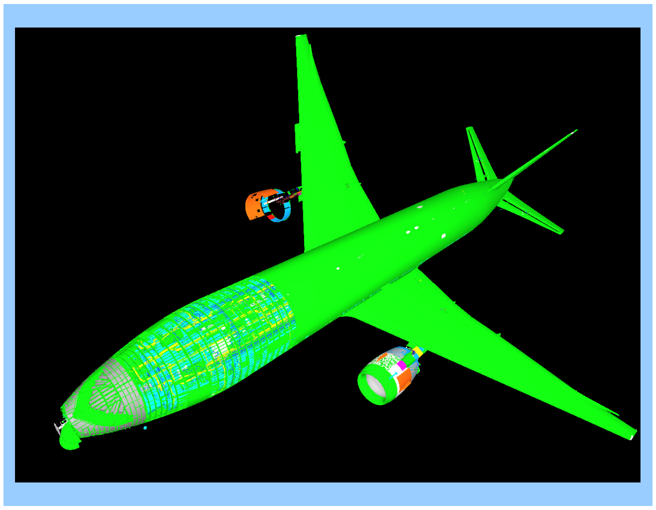
- Octree (480,000 triangles per leaf)
  - 1h 24m, 6,469 leaves
  - 1.3 MB for structure, 10 GB for data
- Visibility coefficients (20 dirs, 64x64 window)
  - 26m, 303 KB
- Levels of detail (up to 5 levels, 1/4 each time)
  - 1h 16m, 2.3 GB
- Total: about 3h and 12 GB





Boeing 777 Results





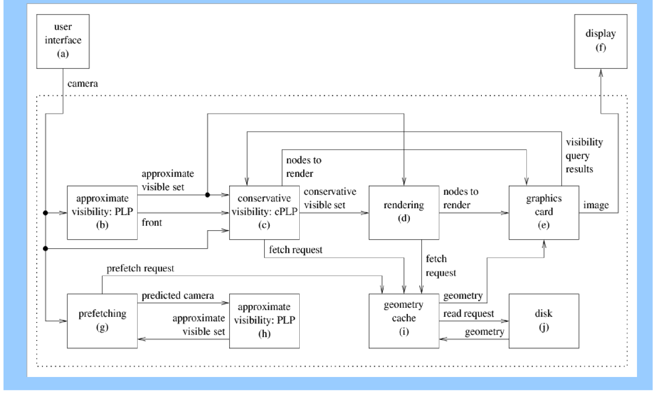
Summary of Preprocessing Results

- Spatialization
  - 5X faster than best similar approach (Wald01)
- Visibility precomputation
  - negligible time and storage requirements
- Simplification
  - fast, good enough, low storage requirements

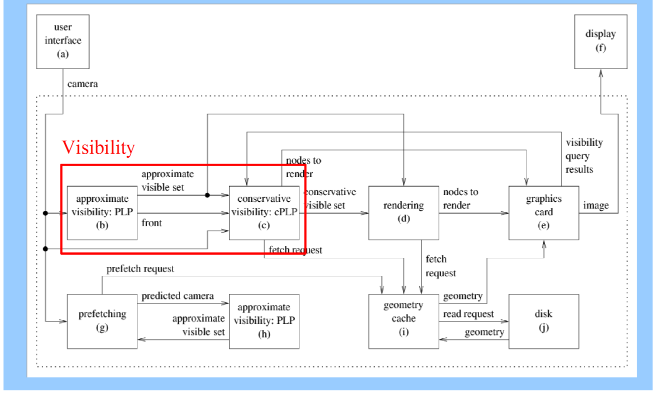
Out-Of-Core Rendering

- Load the visible nodes on demand
- Multiple threads (as opposed to processes)
  - visibility computation
  - cache management
  - prefetching
  - rasterization

The iWalk System

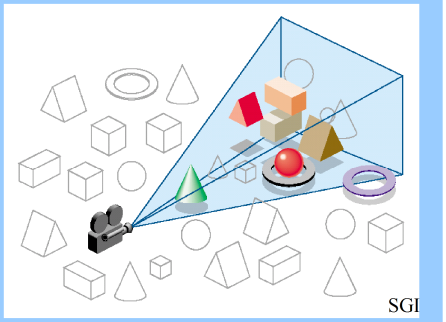


The iWalk System



Occlusion Culling

- Teller91, Greene93, Zhang97, Durand99, Klosowski99, Wonka99, Cohen-or02 Hall-Holt03



SGI

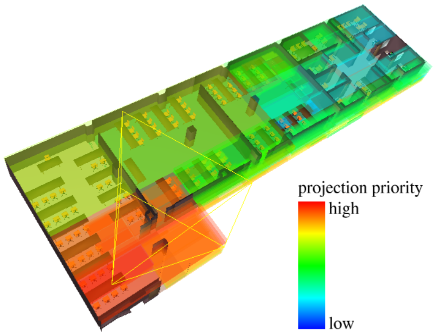
Occlusion Culling

- Classification criteria for occlusion culling algorithms
  - from-point vs. from-region
  - precomputed vs. online
  - object space vs. image space
  - conservative vs. approximate

The PLP Algorithm

- Approximate volumetric visibility
- Keeps the octree nodes in a priority queue called the *front*
- First visits nodes most likely to be visible
- Stops when a budget is reached
- **Doesn't need to read the geometry**
  - estimates the visible set from the hierarchy structure (HS) file

The PLP Algorithm

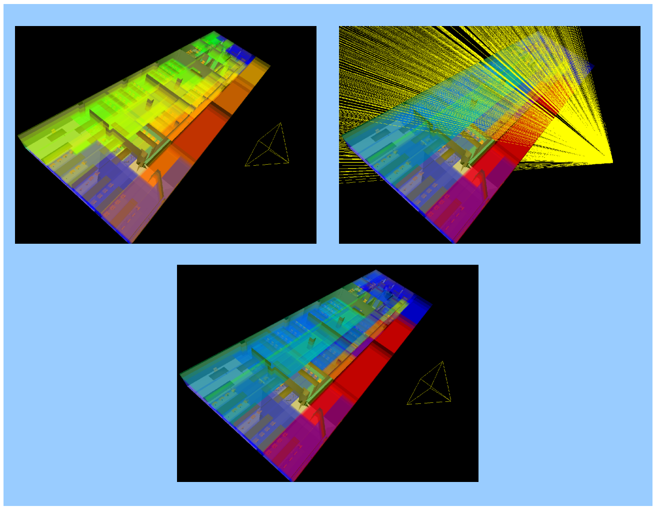


The cPLP Algorithm

- Conservative extension of PLP
- Uses PLP to compute initial guess
- Adds nodes to guarantee correct images
- Unlike PLP, needs to read geometry
  - can't determine visible set from HS file only
- Three implementations
  - item buffer, HP test, NV occlusion query

Improving the Accuracy of PLP

- Use precomputed visibility coefficients to estimate node's opacity for current view
- Shoot rays from user's viewpoint to estimate projection priority of octree nodes
- Ray contribution is initialized to 1
- Attenuate contribution based on opacity of nodes hit along ray path



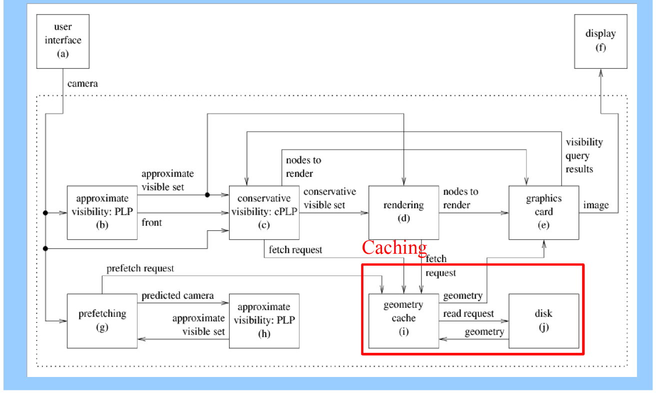
Advantages of Improved Heuristic

- Better images in approximate mode
- Better frame rates in conservative mode
  - less work for cPLP
- Better prefetching
  - less cache pollution
  - fewer cache misses
- Better visibility-based LOD selection

Improving the Running Time of cPLP

- Item buffer
  - slow, multiple tests at a time, int result
- HP occlusion test
  - fast, one test at a time, boolean result
- NV occlusion query
  - fast, many tests at a time, int result

The iWalk System



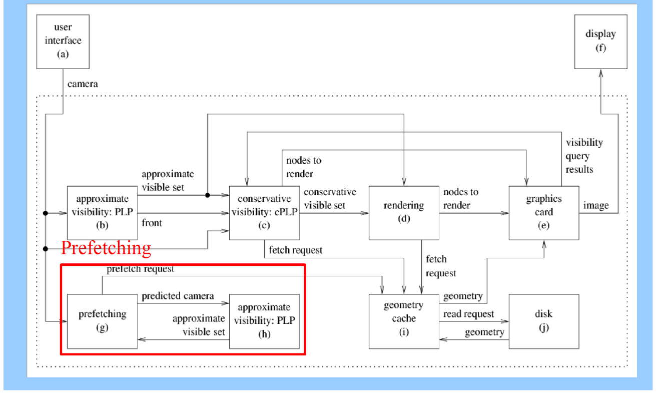
Geometry Caching

- Keep bulk of data on disk
- Bring data into memory on demand
- Keep in memory the least recently used data

The Geometry Cache

- User-defined maximum size
- Blocks of variable size
- Global lock
- Busy flag per block
- Work queue of fetch requests
- Work queue of prefetch requests
- LRU replacement policy

The iWalk System



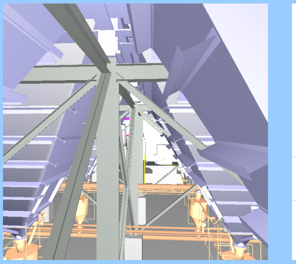
Geometry Prefetching

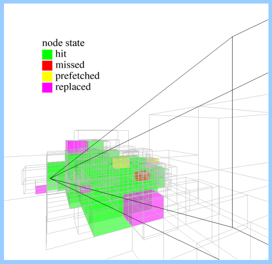
- Guess what data will be needed next
- Read data ahead of time
- Hides I/O latency

From-Point Prefetching

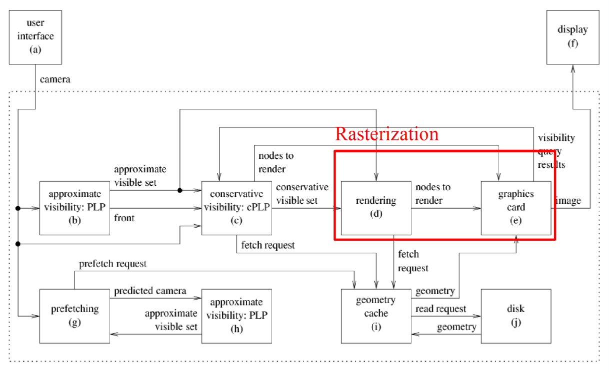
- Improves frame rate by hiding I/O latency
- Uses PLP (approximate visibility algorithm)
    - fine, because prefetching is speculative
- Doesn't need geometry (good for out-of-core)
- Doesn't need graphics pipe (good for PCs)
- Needs less preprocessing than from-region
- Tighter estimate than from-region (less I/O)

The Geometry Cache





The iWalk System



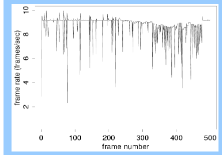
Rasterization

- Pass geometry to the graphics card
  - OpenGL rendering
  - Gouraud shading
- Vertex array per octree node
  - more memory efficient than display lists

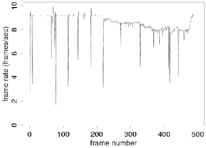
Rendering Results

- Measure frame rates
- Assess image quality
- Evaluate effect of multi-threading and prefetching
- Study the importance of frame-to-frame coherence
- Assess how much better the improved visibility heuristic is

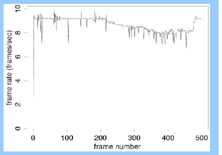
Multi-threading Improves Frame Rates



sequential fetching and rendering

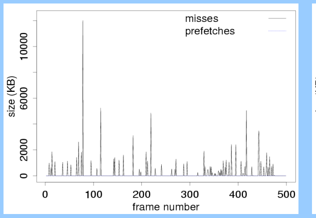


concurrent fetching and rendering

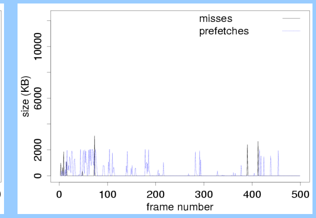


concurrent fetching, rendering, and prefetching

Prefetching Amortizes the Cost of I/O Operations

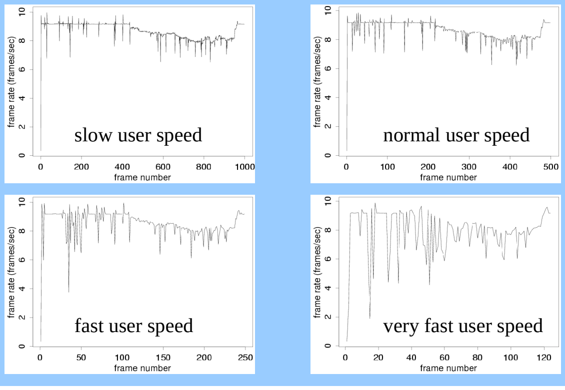


without prefetching



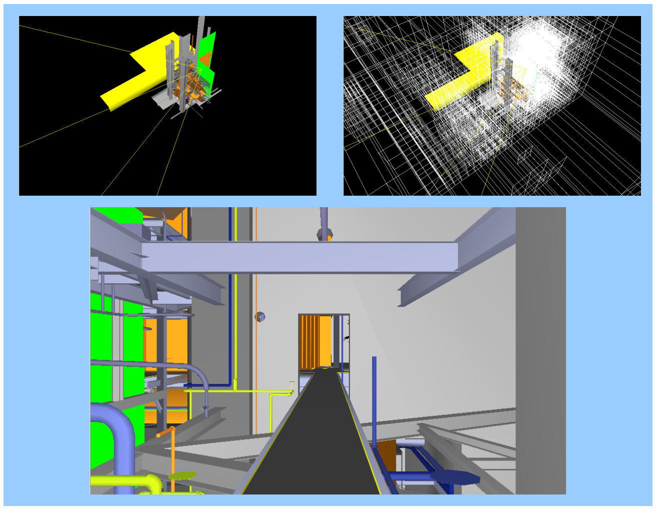
with prefetching

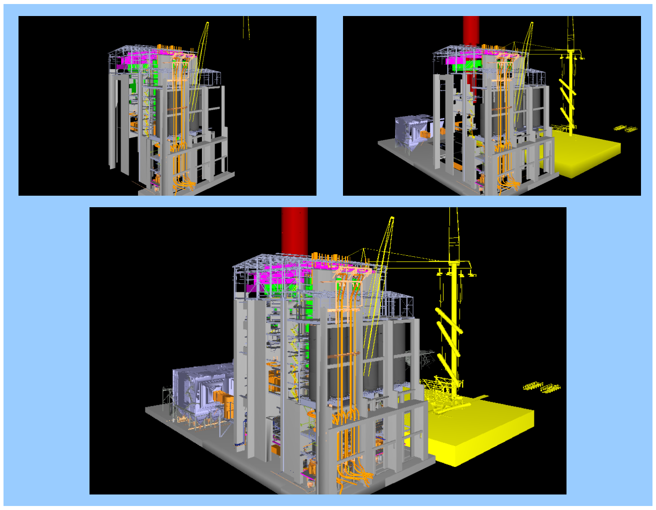
Importance of Frame Coherence



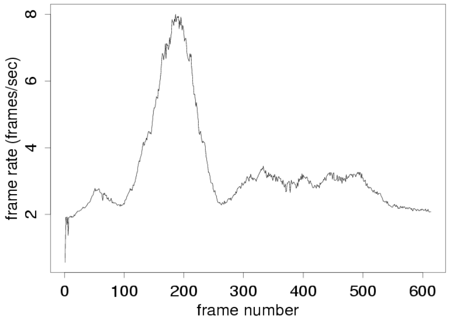
How Much Better is the Improved Visibility Heuristic

- For interior views
  - not much
- For exterior views
  - quite a bit





LLNL Isosurface Rendering Results



Summary of Rendering Results

- We can render a model 20 times larger than main memory at interactive frame rates and acceptable quality on a cheap PC
- Performance is heavily dependent on frame-to-frame-coherence
- Sparse ray tracing helps visibility estimation significantly without much overhead

Out-Of-Core Parallel Rendering

- So far
  - single PC
  - low resolution images (1024x768)
  - interactive frame rates
- Now
  - display wall driven by a cluster of PCs
  - high resolution images (4096x3072)
  - same or faster frame rates

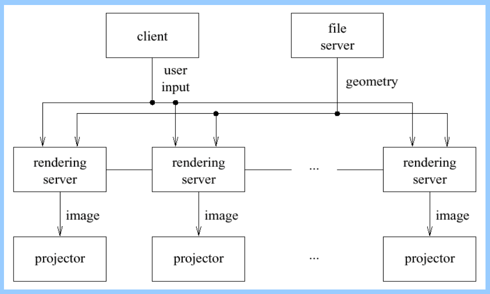
Parallel Rendering

- Sort-first
  - distribute object-space primitives
  - each processor is assigned a screen tile
- Sort-middle
  - distribute image-space primitives
  - geometry processors and rasterizers
- Sort-last
  - distribute pixels
  - rendering and compositing processors

Choosing the Parallelization Strategy

- Why sort-first?
  - each processor runs entire pipeline for a tile
  - exploits frame-to-frame coherence well
- Why *not* sort-middle?
  - needs tight integration between geometry processing and rasterization
- Why *not* sort-last?
  - needs high pixel bandwidth
  - prevents us from using image occlusion queries

The Out-Of-Core Sort-First Parallel Architecture



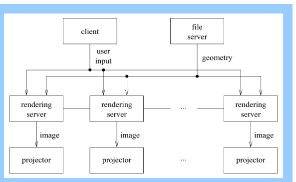
The Out-Of-Core Sort-First Parallel Architecture

- Separate rendering server for each tile
- Client does almost no work, and can be as lightweight as a hand-held computer
- MPI to start and synchronize the servers
- Options: distributed vs. centralized data

UNC Power Plant Tests

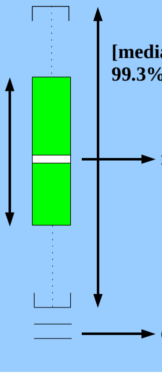
- Pre-recorded 500-frame camera path
- Cluster sizes
  - 1, 2, 4, 8, and 16
- Disk type
  - local and network

Old Cluster



- Rendering servers
  - 900 MHz Athlon, 512 MB of RAM
  - GeForce2, IDE disk
- Client: 700 MHz Pentium III
- File server: 400 GB SCSI disk array
- Network: gigabit Ethernet
- Software: Red Hat Linux 7.2, MPI/Pro 1.6.3

Box Plots



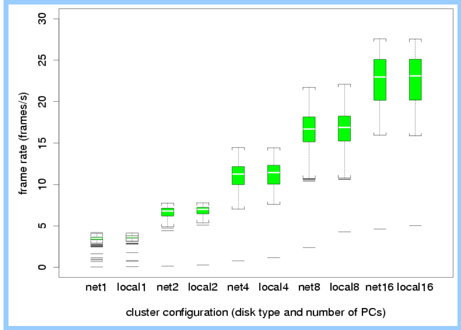
[median – 1.5 IQD, median + 1.5 IQD]: 99.3% of the data (if Gaussian)

interquartile distance (IQD): spread

median: center

outliers

Results for Approximate Visibility

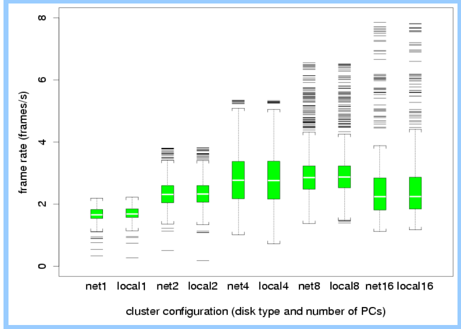


- Median frame rates improve with cluster size
- Disk type makes no difference

Obstacles for Perfect Scalability

- Duplication of effort
  - primitives may overlap multiple tiles
- Communication overhead
  - barrier at the end of each frame
- Load imbalance
  - primitives may cluster into regions

Results for Conservative Visibility Without LODs



- Median frame rates remain almost constant
- Disk type makes no difference
- Additional obstacle: visible geometry increases with resolution

---

Summary of Power Plant Parallel Rendering Results

- 1 PC (1024x768 images)
  - median frame rate: 9.1 frames per second
- 16 PCs (4096x3072 images)
  - median frame rate: 10.8 frames per second
  - cap on frame rate
    - gives prefetching better chance to run
    - reduces frame rate variance

New Cluster

- 8 rendering servers:
  - 2.8 GHz Pentium IV, 512 MB RAM
  - 35 GB SCSI disk
  - NVIDIA Quadro 980 XGL graphics card
- File server
  - same plus 200 GB SCSI disk
- Gigabit Ethernet
- Red Hat Linux 8.0, MPICH 1.2.5

LLNL Isosurface Parallel Rendering Results

- Conservative visibility and LOD
- 8 x 1280 x 1024 (10 megapixels)
- For outside views
  - 3-5 frames per second
- For inside views
  - 8-10 frames per second
- Frame rates using shared disk almost the same as frame rates using local disks

Summary of Parallel Rendering Results

- We can scale the resolution of an application without any loss in performance
- Caching and prefetch exploit coherence well: even with centralized file server, usually limited by rendering

Comparison to Other Parallel Rendering Systems

- Better frame rates than Humphreys02, but we do need to change the source code
- Faster frame rates and higher resolution than Wald01, but lower image quality
- Similar frame rates to Moreland01, plus image occlusion queries

Conclusions

- iWalk system is practical and scalable
- Out-of-core techniques are fast and effective
- PCs are an attractive, cost-effective alternative to high-end machines
- The system can help to bring visualization of large datasets to a broader audience

Research Contributions

- Efficient out-of-core algorithm to build octree
- Extensions of the PLP visibility algorithm
  - ray-tracing based approximate heuristic
  - hardware-assisted conservative extension
- Out-of-core, from-point prefetching algorithm
- Out-of-core sort-first architecture

Future Work

- Support for different types of scenes
  - textures, volumes (working prototype), dynamics
- Efficiency
  - add geometry and appearance quantization
  - eliminate geometry replication
- Analysis
  - develop analytic model for system parameters
  - optimize system parameters automatically

Acknowledgements

- Financial support
  - CNPq (Brazilian research funding agency)
  - Princeton University
  - AT&T Research
  - Oregon Graduate Institute
  - IBM Research
- Datasets
  - UNC Chapel Hill, UC Berkeley, 3rdTech, LLNL, Boeing