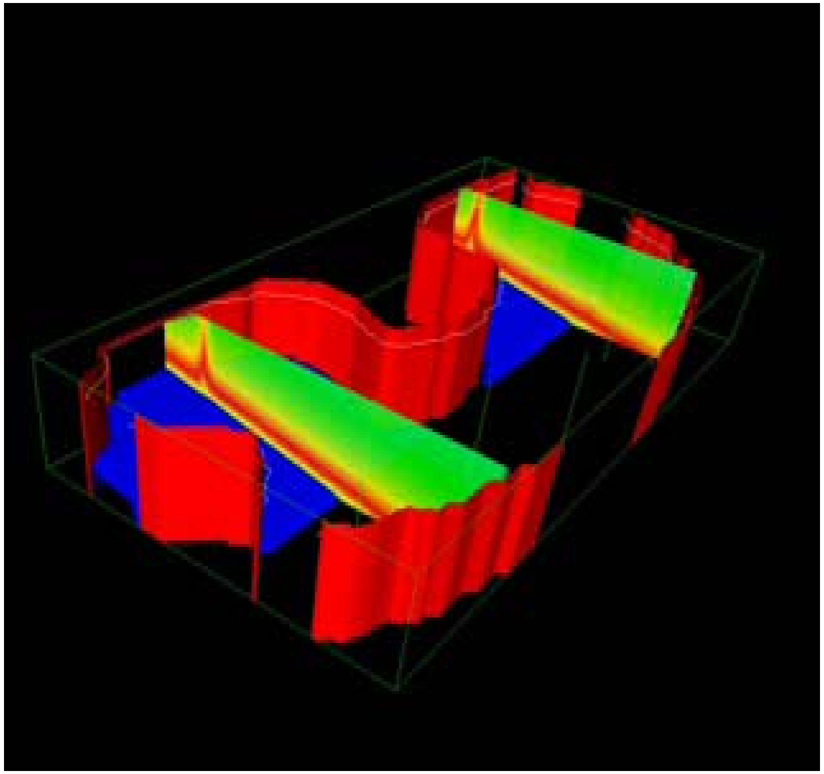
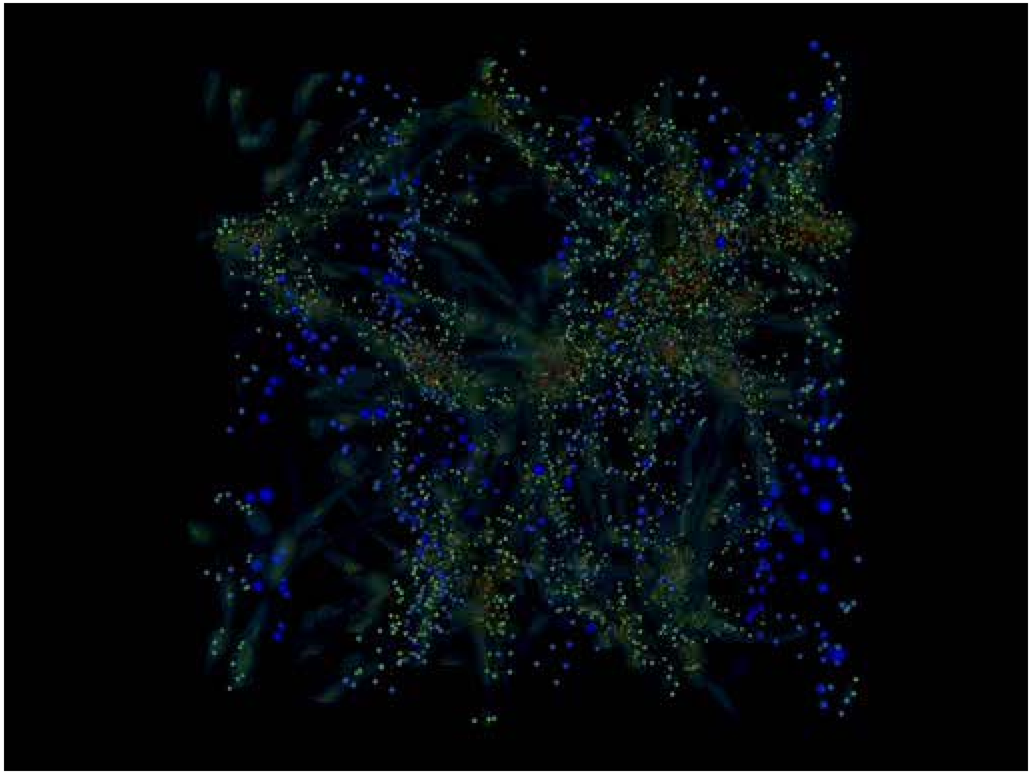
Active Visualization in a Multidisplay Immersive Environment using COTS

Chandrajit Bajaj

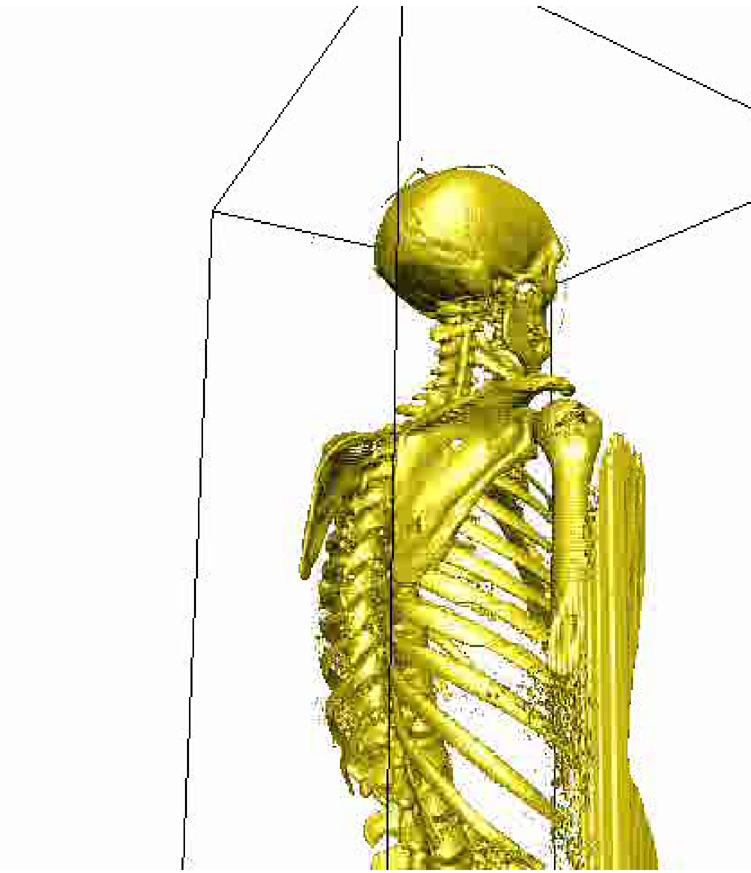
Center for Computational Visualization
Dept. of Computer Sciences & TICAM
University of Texas at Austin
http://www.ticam.utexas.edu/CCV

Exploratory Visualization of Large Simulations





Exploratory Visualization of Large Imaging Data

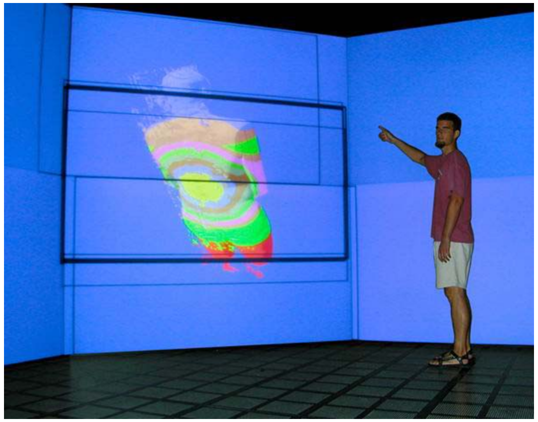


Scalable Visualization

- Visualize increasingly large data meshes from imaging & simulation
- Improve performance by adding computational resources

Visualization Cluster

Isosurface

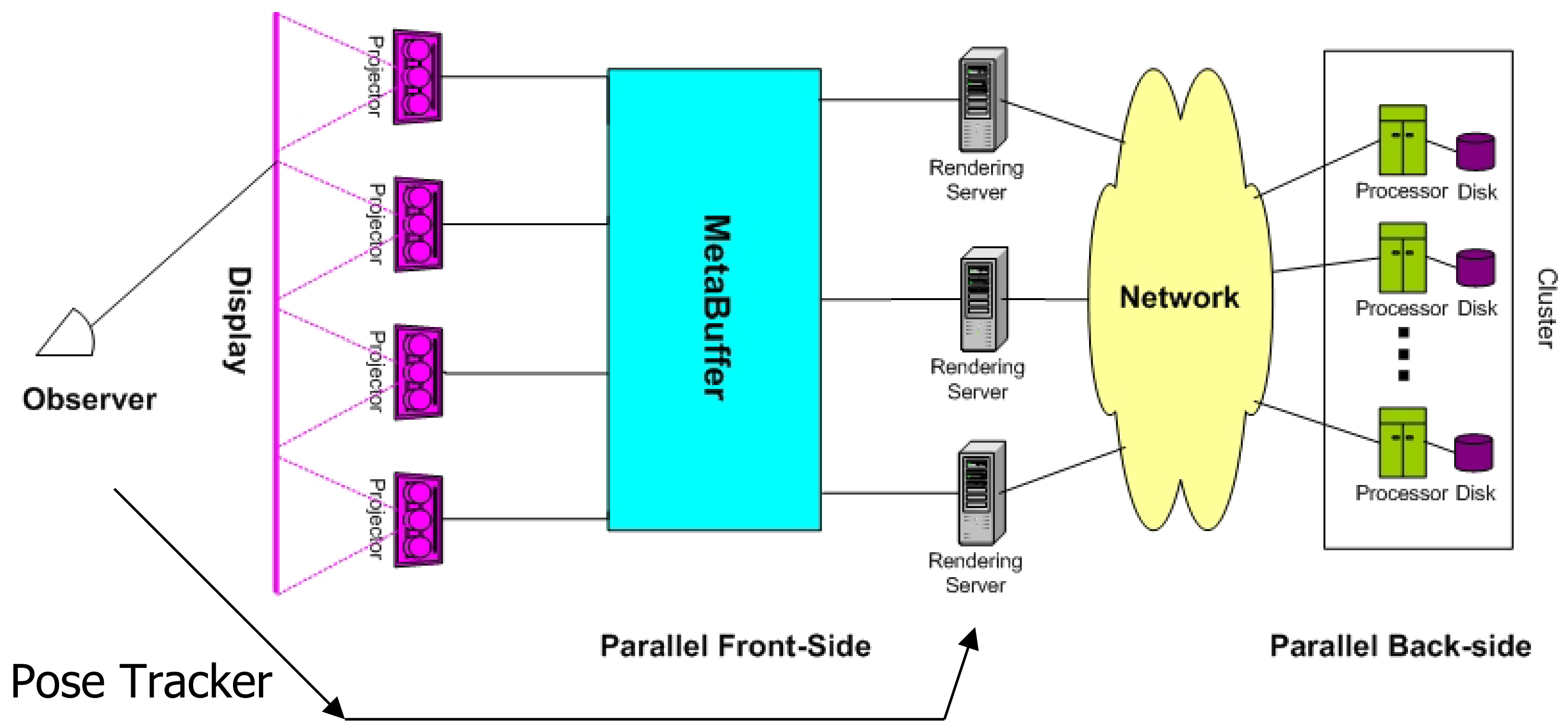


Large display

Objective

- primary costs in scalable visualization:
  - Disk I/O (loading field data from disks)
  - Extraction (geometry)
  - Rendering (geometry)
- A Scalable Visualization Framework
  - Scales with field datasets
  - Scales in computation
  - Scales in disk I/O
  - Scales in rendering

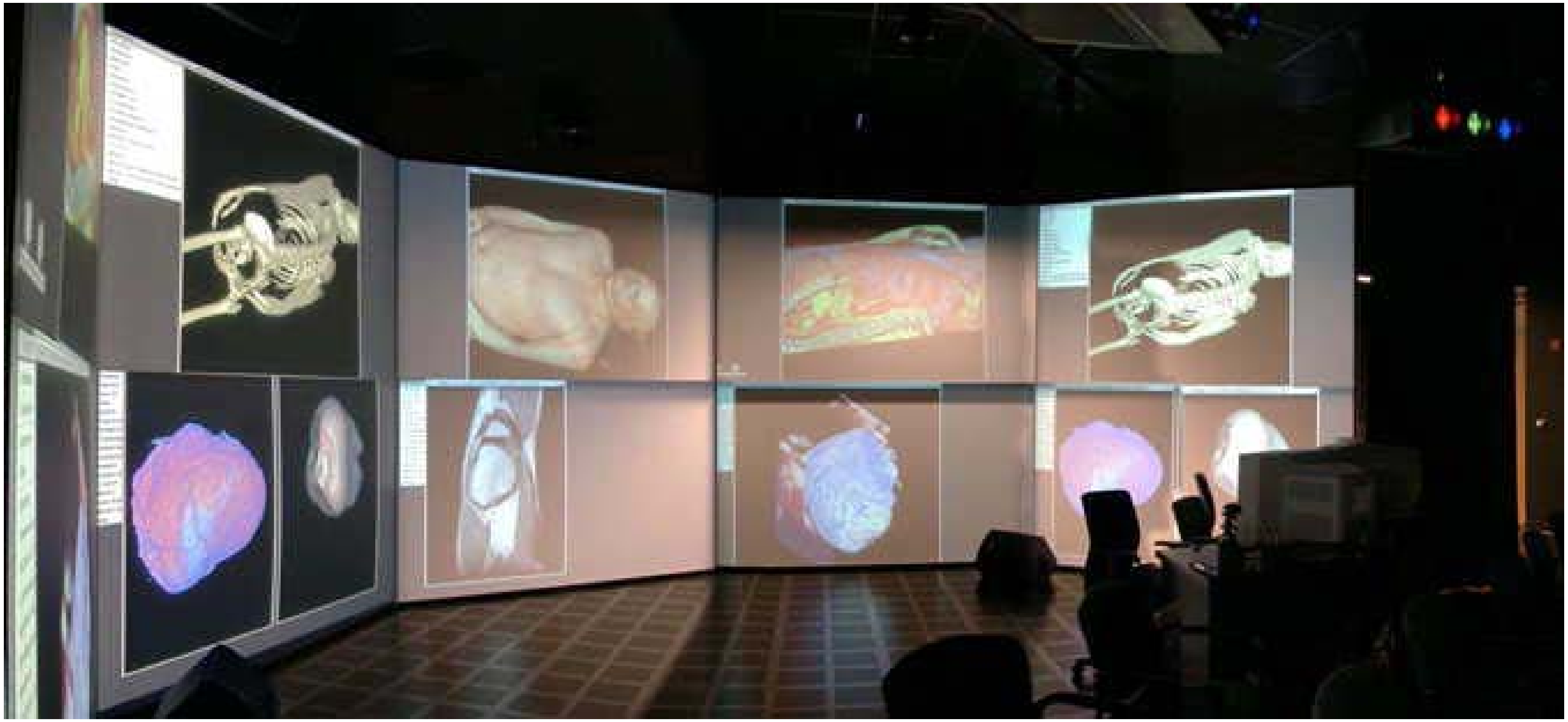
Scalable Closed-Loop Visualization



Cluster

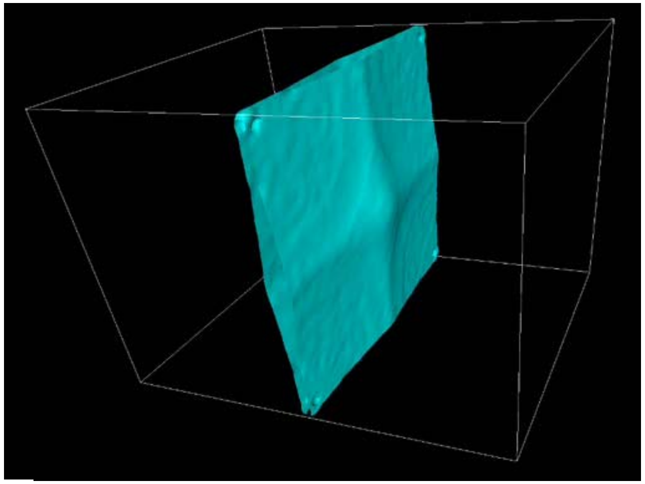
- A PC cluster of 128 Nodes (Compaq SP750)
- PIII 800 MHZ CPU and 256 MB memory
- 100 Mb/s Ethernet (partial nodes with Servernet II and Gigabit Ethernet)
- 32 nodes with GeForce II graphics cards
- 9 GB system disk and 18 GB data disk
- Linux kernel 2.2.19

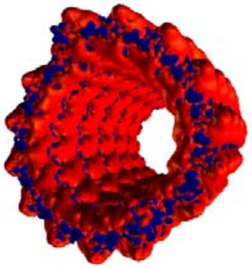
Large Display



Isocontour Visualization

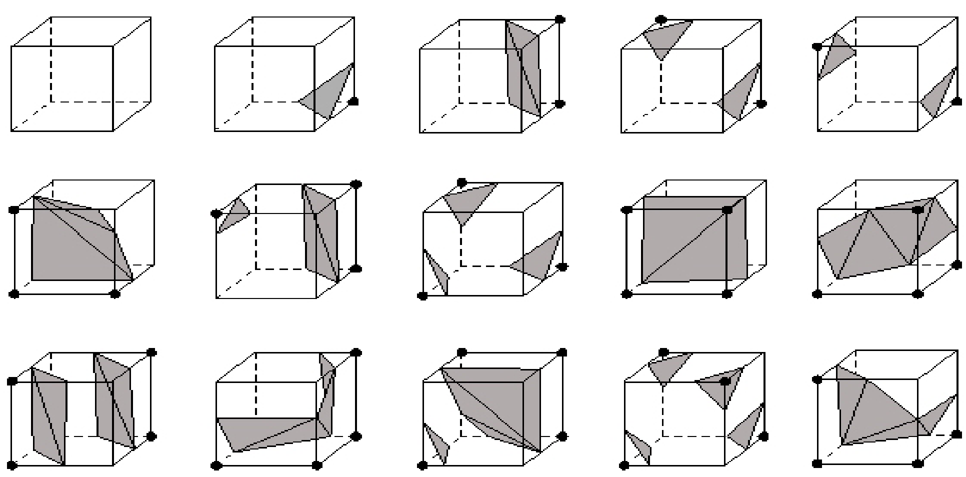
- Input:
  - Scalar Field *F* defined on a mesh
  - Query Isovalue *w*
- Output:
  - Contour $C(w) = \{x \,/\, \mathbf{F}(x) = w\}$





Isocontouring

- Two primary stages in contour extraction from a mesh:
  - Search for intersected cells
  - Contour approximation within an intersected cell



15 distinct cases for triangulating a 3D regular cell

Scalable Isosurface extraction

- Scalable with the number of processors
  - Good load balance and speedup
  - Improve interactivity with more processors
- Scalable I/O
  - Parallel disks and balanced disk I/Os
  - Avoid I/O bottleneck for large datasets
- Scalable with the size of datasets
  - Out-of-core computation
  - Minimum data replication
  - Handle larger datasets when resources are fixed

Computation Model

- N: Total size of the problem
- M: Main memory of a single processor
- P: Number of processors
- D: Number of disks.
- B: Size of a disk block
- L: Latency parameter of the BSP model
- g: Bandwidth parameter of the BSP model

$$N > P*M$$

Computation Time

$$T = \max_{p}\left(T_w + T_c + T_{io}\right)$$

T_w: Local computation time

T_{io} : Disk access time

T_c : Communication time

Isocontouring Algorithms

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">Search Space</td></tr>
<tr><td>Geometric Space</td><td>Range Space</td></tr>
<tr><td rowspan="2">Contour Strategy</td><td>Cell by Cell</td><td>Marching Cubes [Lorenson/Cline]
Octree [Wilhelms/Van Gelder]</td><td>Span Filtering [Gallapher]
Sweeping Simplices [Shen/Johnson]
Kd-Tree [Livant/Shen/Johnson]
LxL Lattice [Shen et al.]
Interval Tree [Cignoni et al.]</td></tr>
<tr><td>Propagation</td><td>Extrema Graph [Itoh/Koyamada]</td><td>Contour Tree [Van Kreveld]
Seed Set [Bajaj/Pascucci/Schikore]</td></tr>
</table>

External Search Data Structure

- Meta-block tree *(Kanellakis '93)*
- External Interval Tree *(Arge and Vitter '96)*
- Binary Blocked Interval Tree *(Chiang and Silva '98)*

Optimal disk space $O\left(\frac{N}{B}\right)$

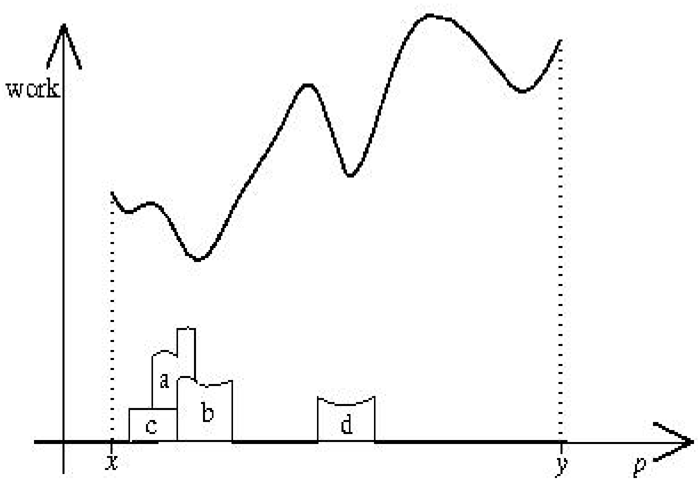
Optimal query I/O operations $O\left(\log_B N + \frac{T}{B}\right)$

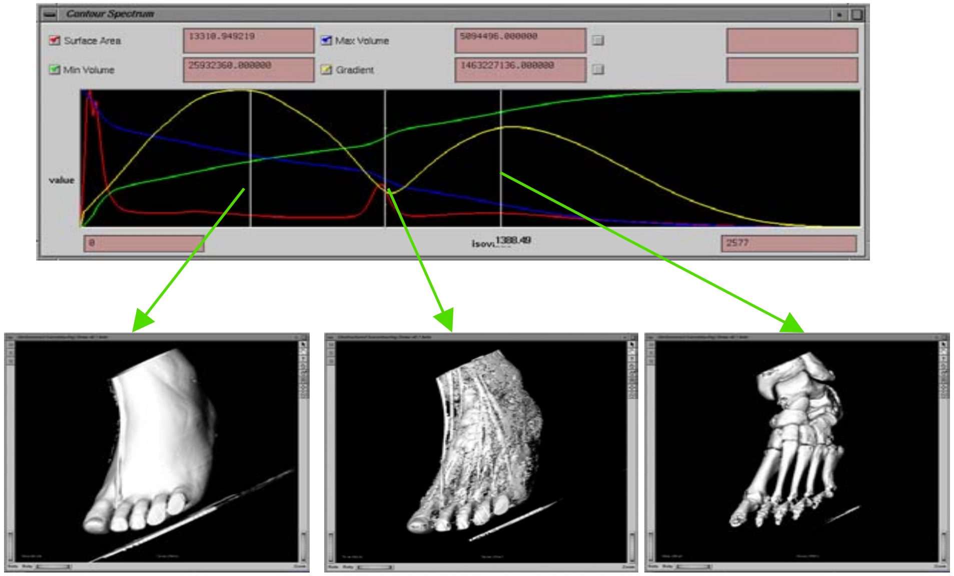
Parallel Isocontouring

- SIMD Hansen/Hinker '92
- Parallel (Cluster) Ellsiepen '95
- Parallel LxL lattice Shen et al. '96
- Parallel ray tracing Parker et al. '98
- Range Partition Bajaj et al. 99

Contour Spectrum

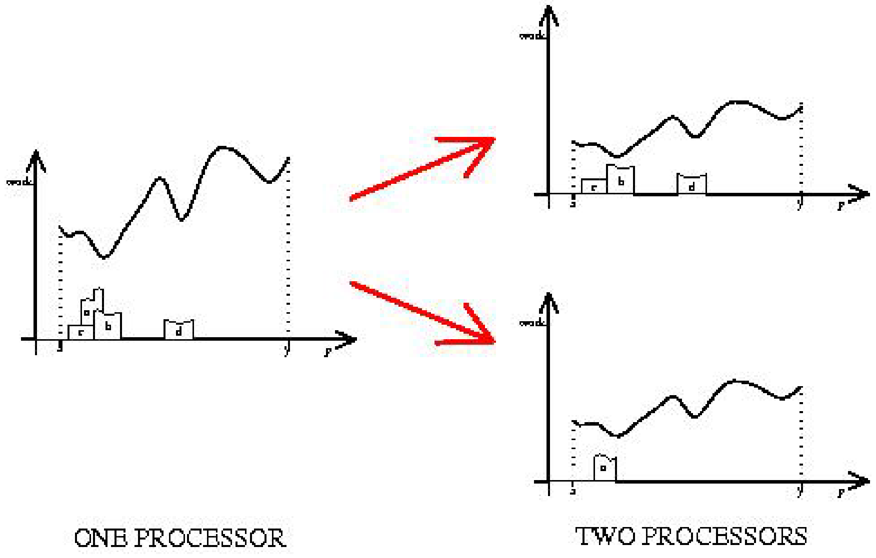
- Spectrum of a data set can represent the work load for different isovalues p





The overall work load diagram for each isovalue is the sum of the diagrams of atomic units

Ideal Data Partition



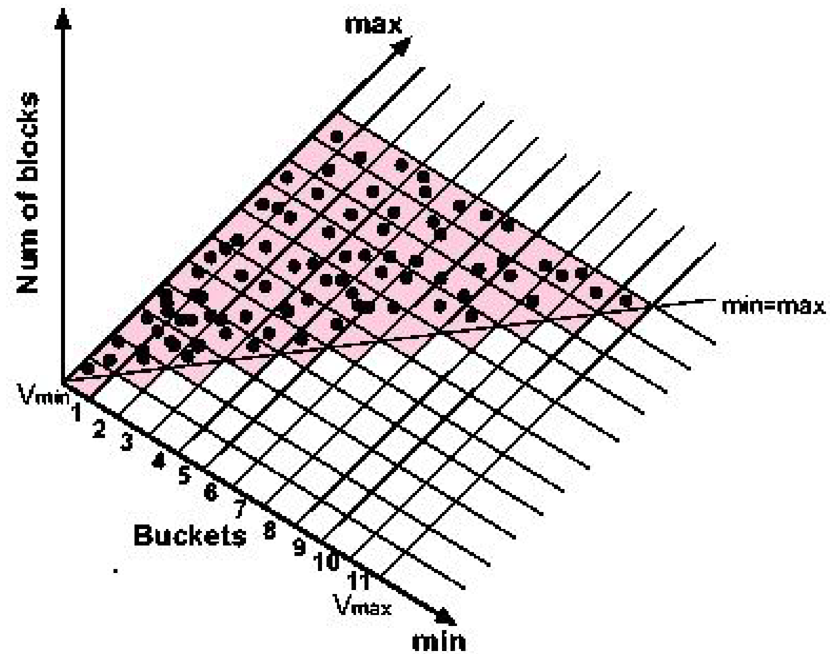
- Ideal data partition for load balanced parallel computations (two processor case)

Static Data Partitioning

- Why Static partitioning?
  - Communication is slow
  - Dynamic data assignment requires run-time redistribution or data replication
  - It gives good load balance for massive datasets
- How?
  - Deterministic algorithm
  - Randomized algorithm

Deterministic Algorithm

- 1. Partition volume into blocks of the same order as disk blocks
- 2. Partition range space into an nxn lattice
- 3. Sort blocks in each lattice element by their range sizes
- 4. Assign the sorted blocks in a round-robin fashion



Randomized Algorithm

- 1. Partition volume into blocks of the same order as disk blocks
- 2. Assign blocks randomly to processors

Well balanced when the number of blocks is large

Randomized Algorithm

Theorem (Raghavan 88) Let $a_1, \cdots, a_n$ be real numbers in $(0,1]$. Let $x_1, \cdots, x_n$ be independent Bernoulli trials with $\mathrm{E}(x_j) = \rho_j$. Let $\psi_\beta = \sum_{j=1}^{n} a_j x_j$. If $\mathrm{E}(\psi_\beta) > 0$, then for any $\nu > 0$

$$\Pr(\psi_\beta > (1+\nu)E(\psi_\beta)) < \left(\frac{e^\nu}{(1+\nu)^{(1+\nu)}} \right)^{E(\psi_\beta)}$$

It shows that with high probability no processor has much larger that the average work load if there are many blocks.

P. Raghavan *Journal of Computer and System Sciences*, 37:130-143, 1988

Scalable Parallel & Out-of-core Isocontouring

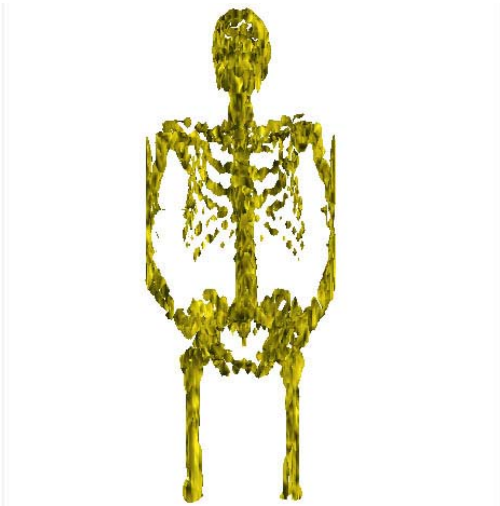
(Sketch of preprocessing Algorithm)

- Assume input to be slabs distributed among D disks
- Rearrange data into blocks of size θ(B)
- Assign data statically onto the processors and disks
  - Good load balance
  - Minimum data replication
- Communicate blocks to their destined disks
- Build external interval tree for blocks on each disk
  - Load only relevant data blocks
  - Minimize disk access

Parallel and Out-of-core Isocontour Querying

- Each processor runs independently
  - Searches its external interval tree to find its active blocks
  - Loads its active blocks and extract isosurfaces
  - Renders the extracted isosurfaces with its local graphics board
  - Final Image is composited by the Metabuffer

Multi-resolution Isosurfaces



24,084 triangles

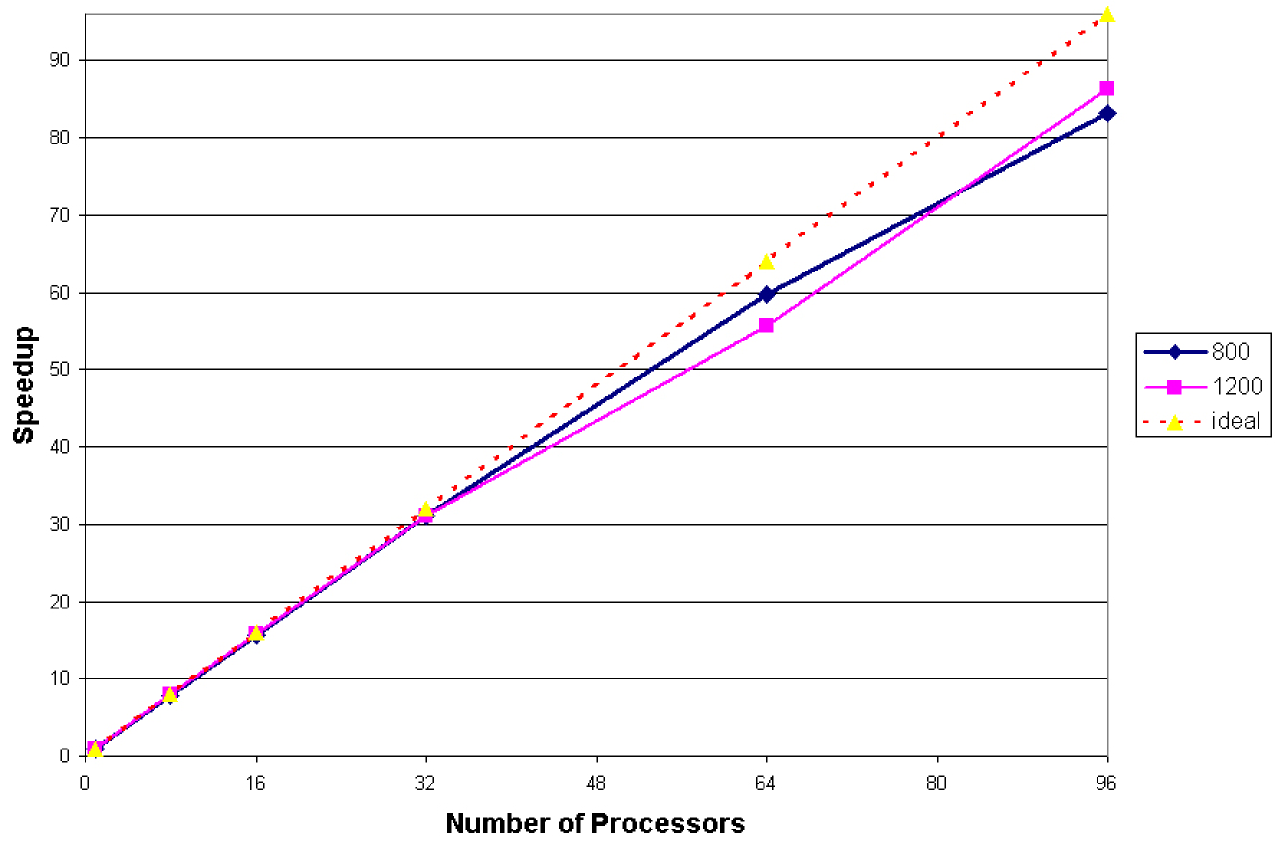
651,234 triangles

6,442,810 triangles

Test Datasets

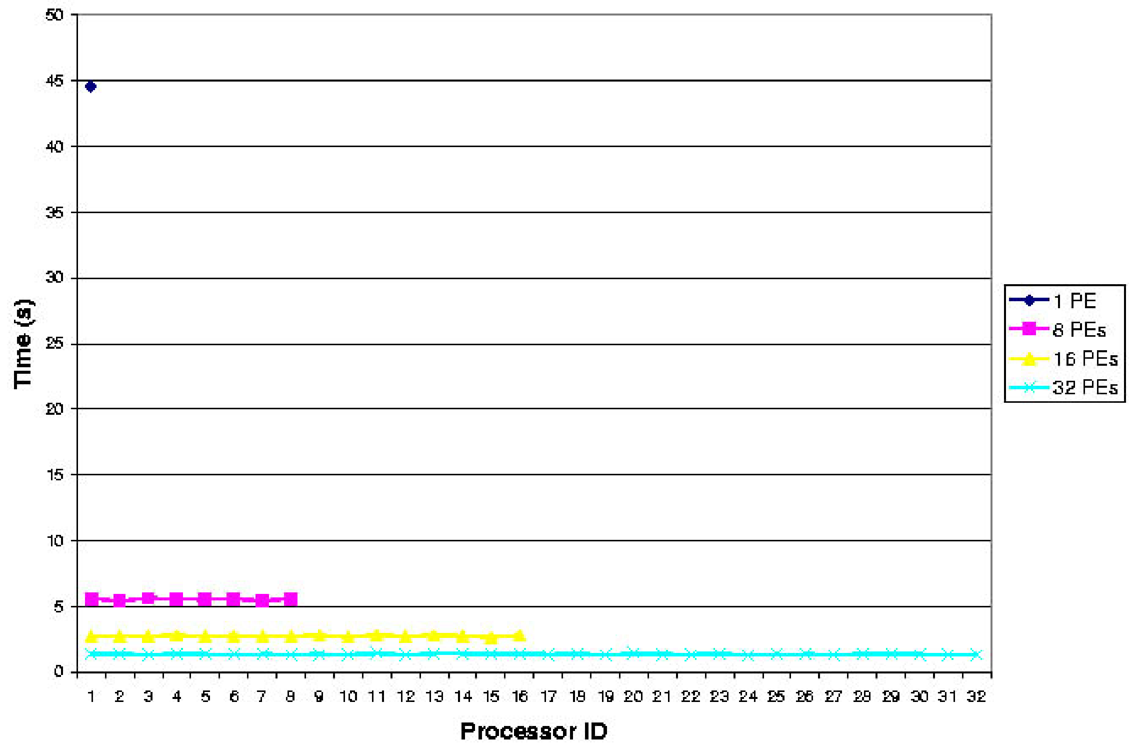
Name	Dimension	Size
Male MRI	*512x512x1252*	*656 MB*
Male Cryosection	*1800x1000x1878*	*6.6 GB*
Female Cryosection	*1600x1000x5186*	*16.5 GB*

Speedup



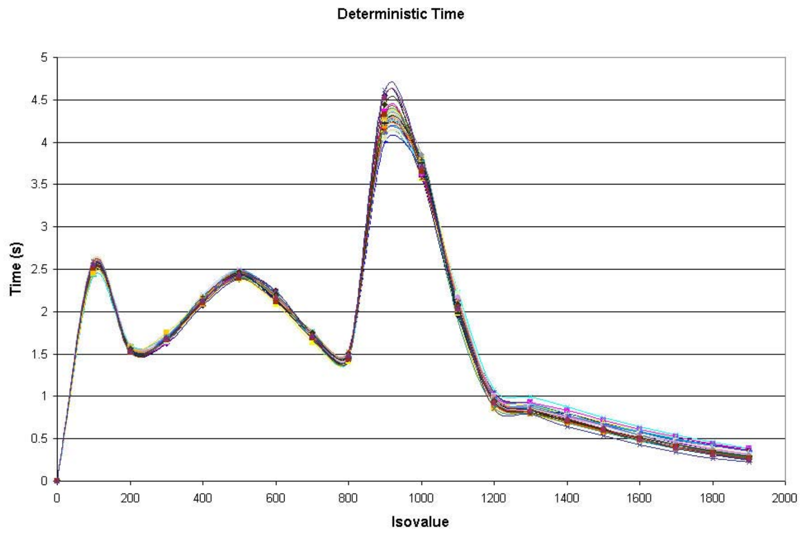
Visible Male MRI Dataset with random data distribution

Extraction and Rendering Time

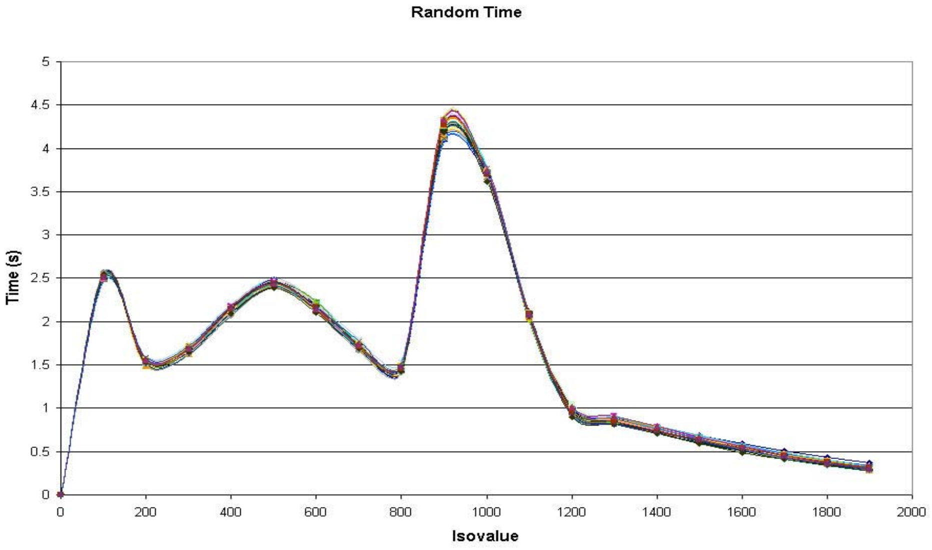


Visible Male MRI Dataset (isovalue = 800)

Workload Histograms



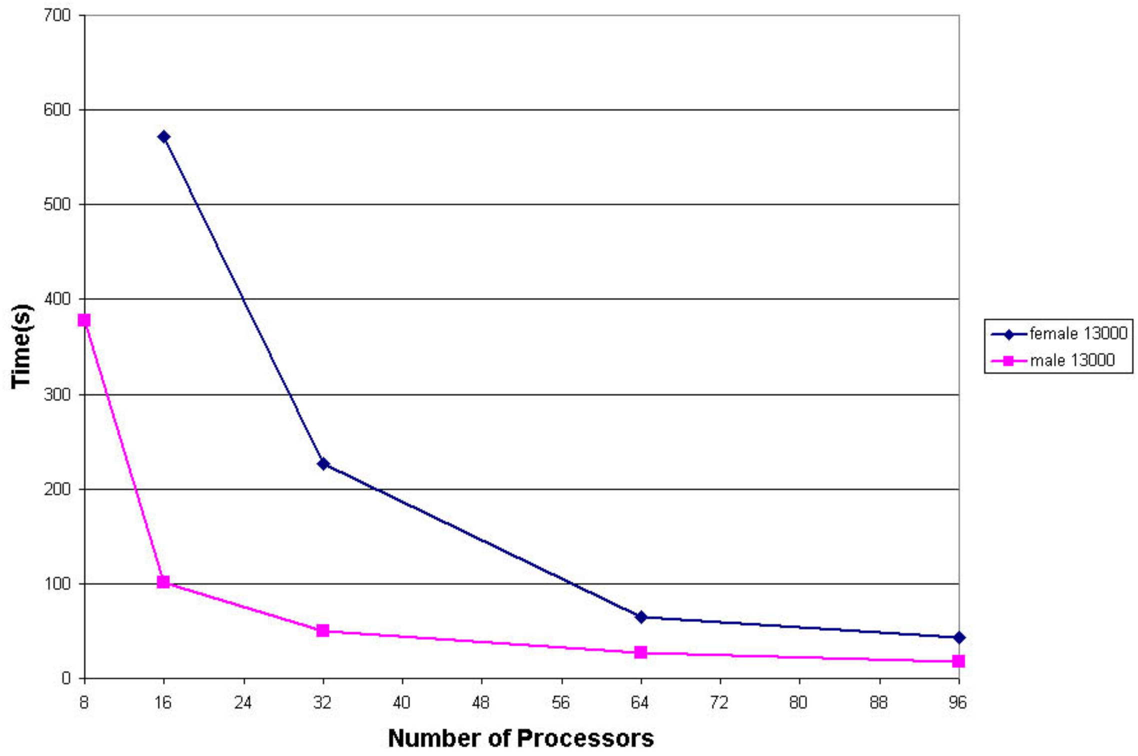
Deterministic Greedy Algorithm



Randomized Algorithm

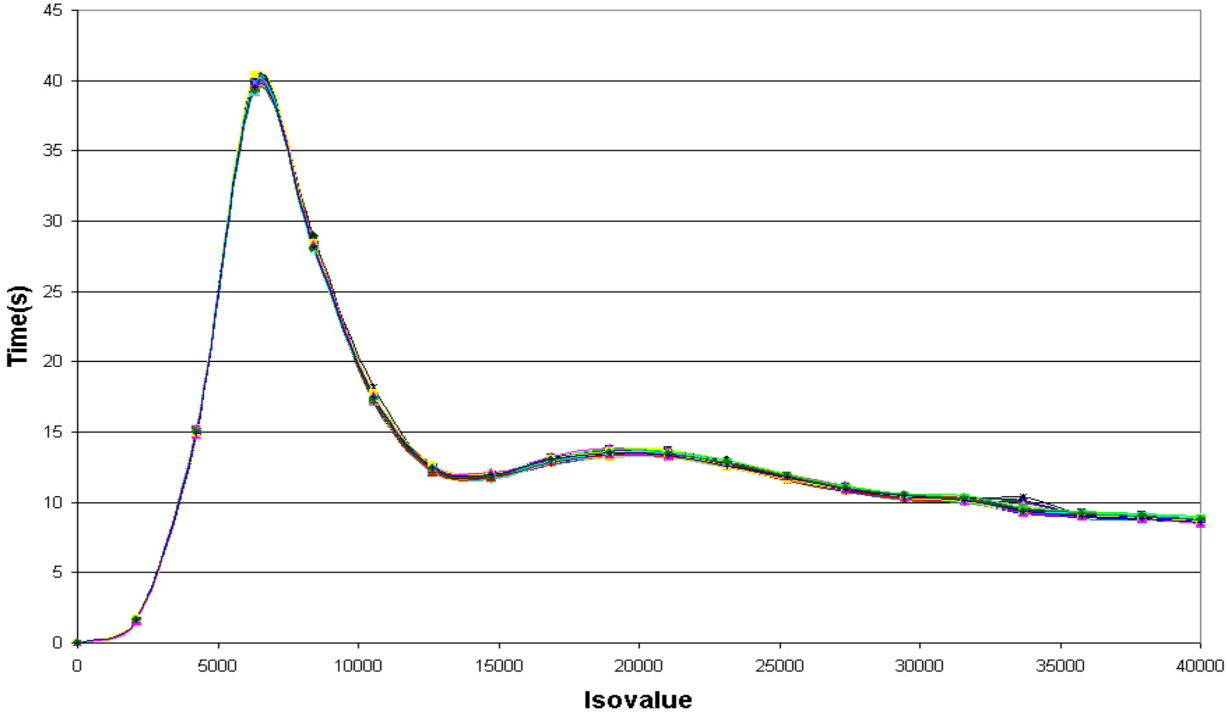
Male MRI Dataset with 32 processors

Extraction Time



Male and Female Cryosection Datasets

Workload Histogram for Male Cryosection Dataset



Male Cryosection Dataset with 96 processors

View Dependent Isocontouring

- Why?
  - Many polygons are invisible
  - Reduce extraction and rendering time
- Conditions
  - No pre-existing polygons for generating occlusion maps
  - Data Blocks are distributed among multiple processors
  - Conservative visibility culling (No holes)

View-dependent Rendering

- Visibility Culling
  - Object space culling
    - Interactive walkthrough *(Teller and Sequin '91)*
    - Occlusion BSP Tree *(Naylor '92)*
    - Prioritized-layer projection *(Klosowski and Silva '99)*
  - Image Space Culling
    - Hierarchical Z-Buffer *(greene et al. '93)*
    - Hierarchical Tiling *(greene '96)*
    - Hierarchical Occlusion Map *(Zhang '97)*
    - Lazy Occlusion Grid *(Hey '01)*
    - Randomized Z-Buffer *(Wand '01)*

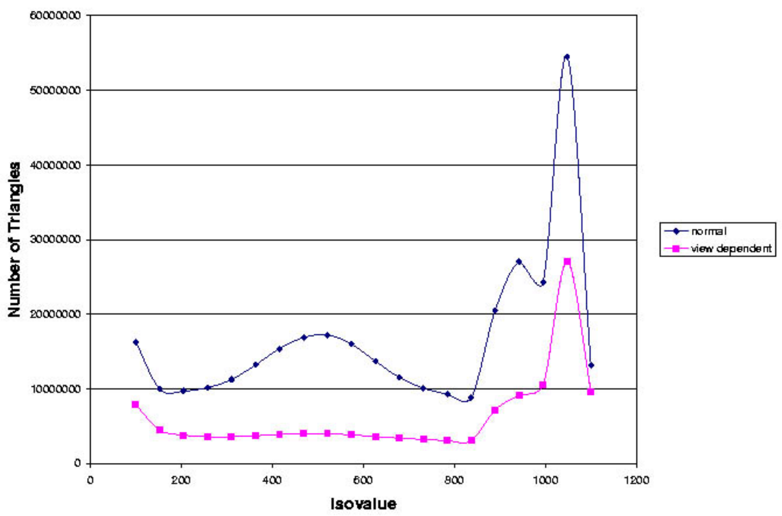
View-dependent Isocontouring

- View-dependent Isocontouring
  - Octree front-to-back traversal *(Livnat and Hansen '98)*
  - Parallel ray-tracing *(Parker et al. '98)*
  - Ray-casting & Propagation *(Liu et al. '00)*
  - Parallel Multi-pass *(Gao and Shen '01)*
  - Parallel Single-pass *(Zhang and Bajaj '02)*

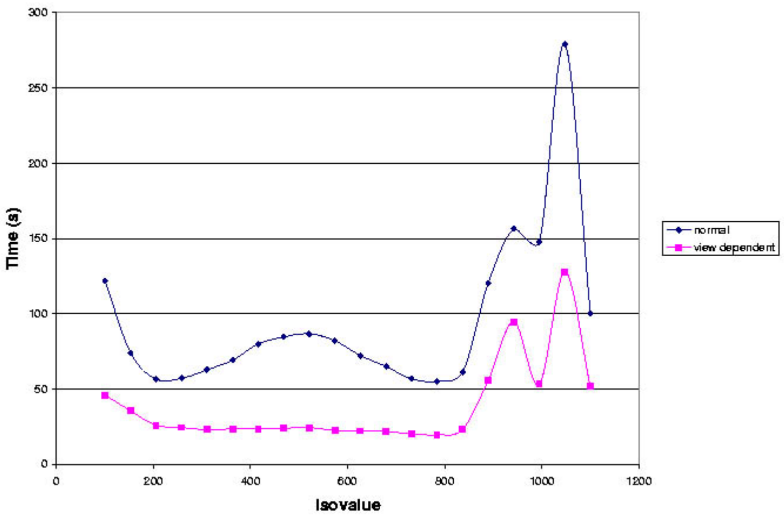
Algorithm outline

- Occluder Selection
  - Find initial occluding blocks by raycasting
  - Build occlusion map by extracting and rendering isosurfaces in the occluding blocks
- Visibility Culling
  - Cull the remaining blocks with the occlusion map

Results



# of triangles extracted

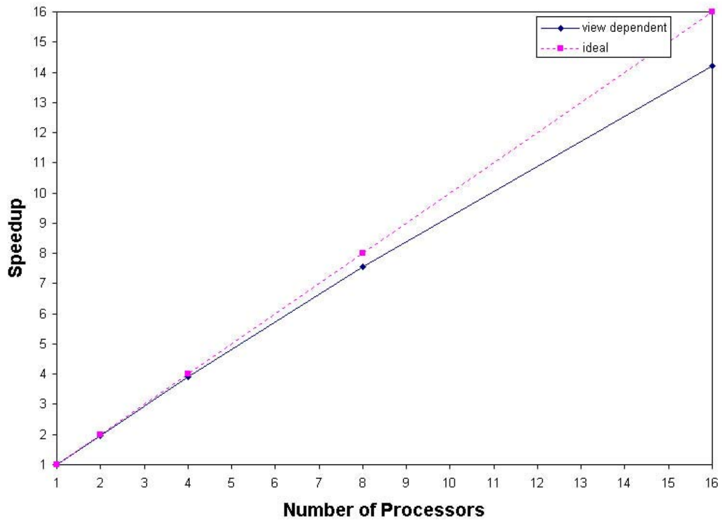


Extraction and rendering time

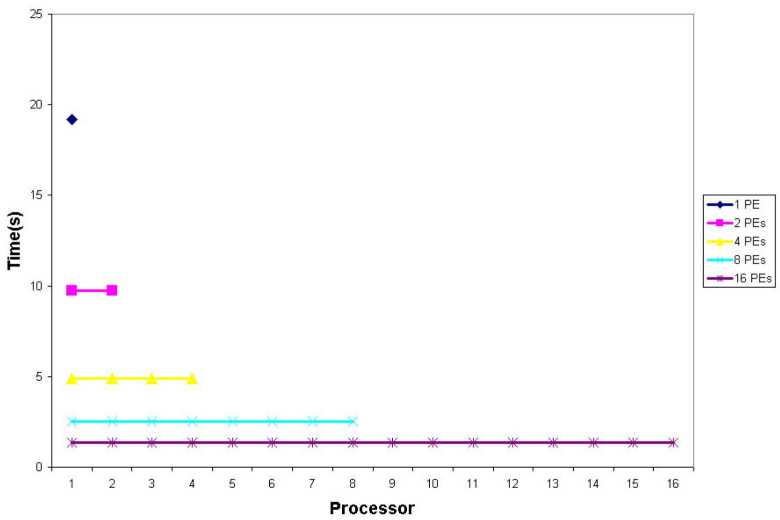
Parallelization

- Parallelize occluder selection
  - Each processor shoots a subset of rays
  - Occluding blocks are the union among processors
  - Block ranges are replicated on each processor
- Parallelize occlusion map construction
  - Each processor extracts and renders a subset of occluding blocks that reside on its local disk
  - Occlusion maps of individual processors are merged
- Parallelize Visibility Culling
  - Each processor queries its own external interval tree and tests its local blocks
  - Each processor extracts and renders visible blocks on its local disk

Results



Speedup



Extraction and rendering time

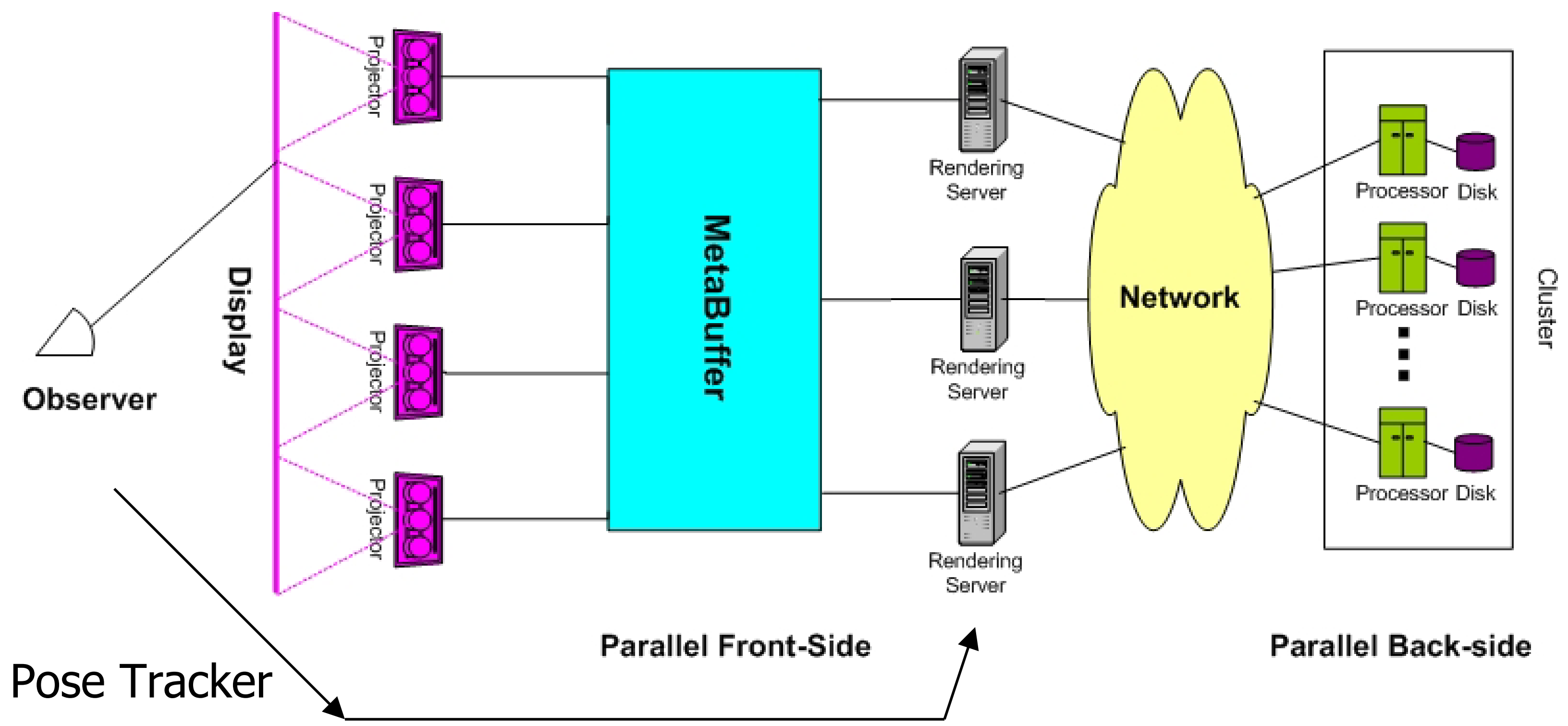
Good Features

- Conservative (no hole)
- Single Pass
- Easily Parallelizable
- Well Load Balanced
- Out-of-core

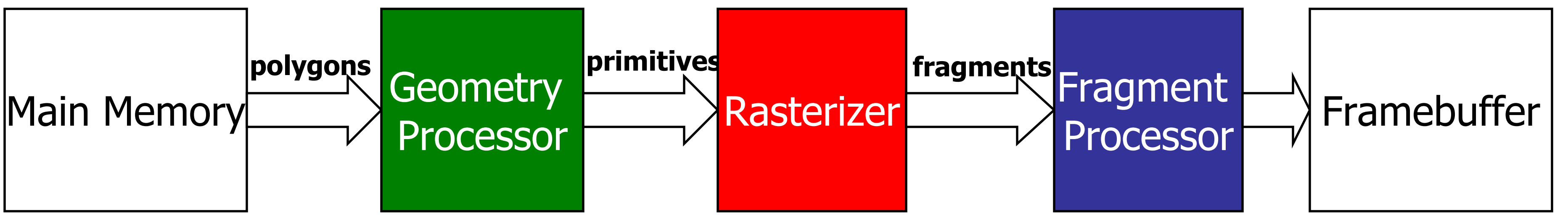
Parallel Rendering

- Fast display of large isosurfaces
- Based upon the Metabuffer architecture
  - Parallel renders mapped to tiled displays
  - Load balance among rendering processes
  - Many possible configurations

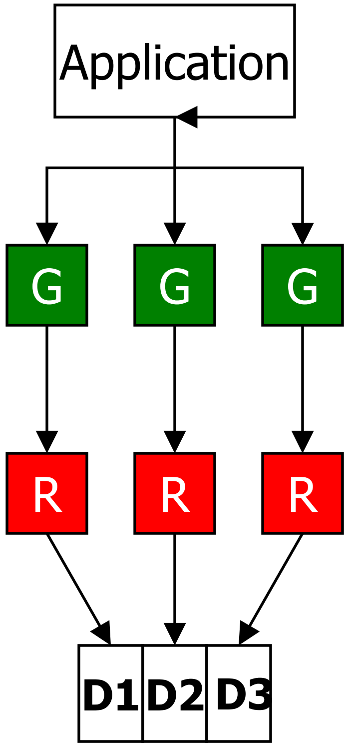
Scalable Closed-Loop Visualization



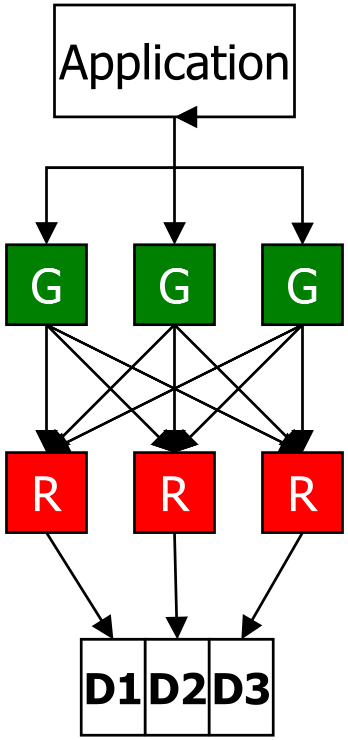
Graphics Pipeline



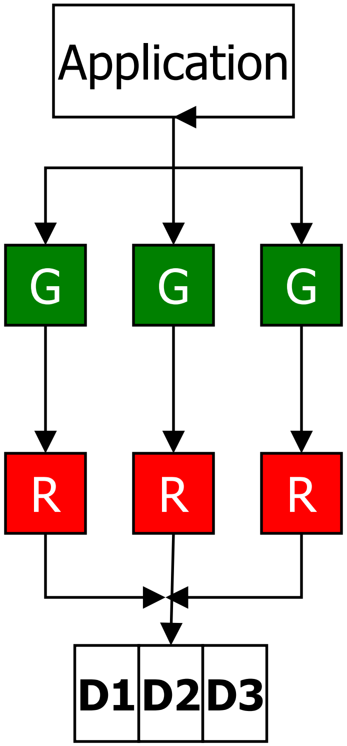
Parallel Rendering



Sort First



Sort Middle



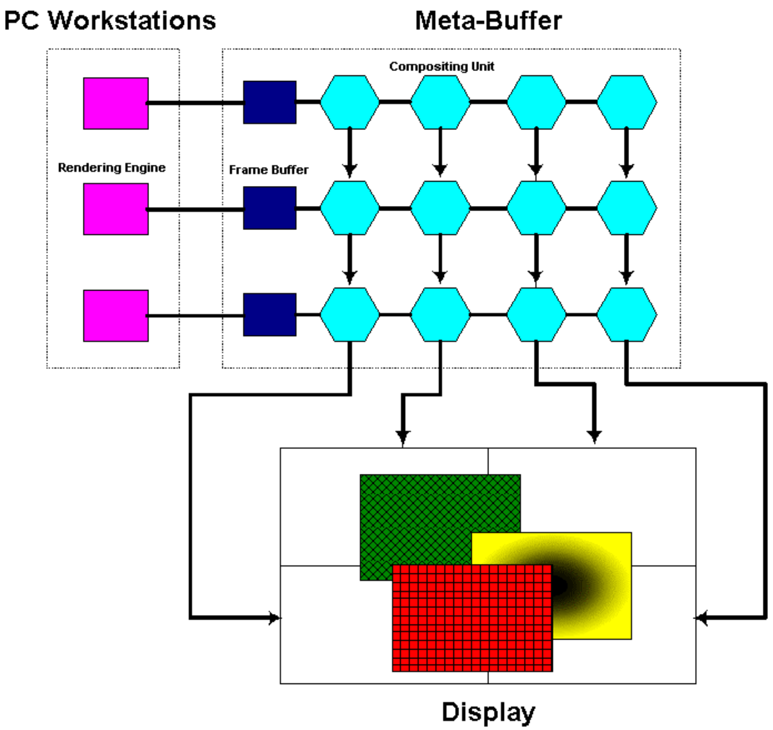
Sort Last

Scalable Parallel Rendering

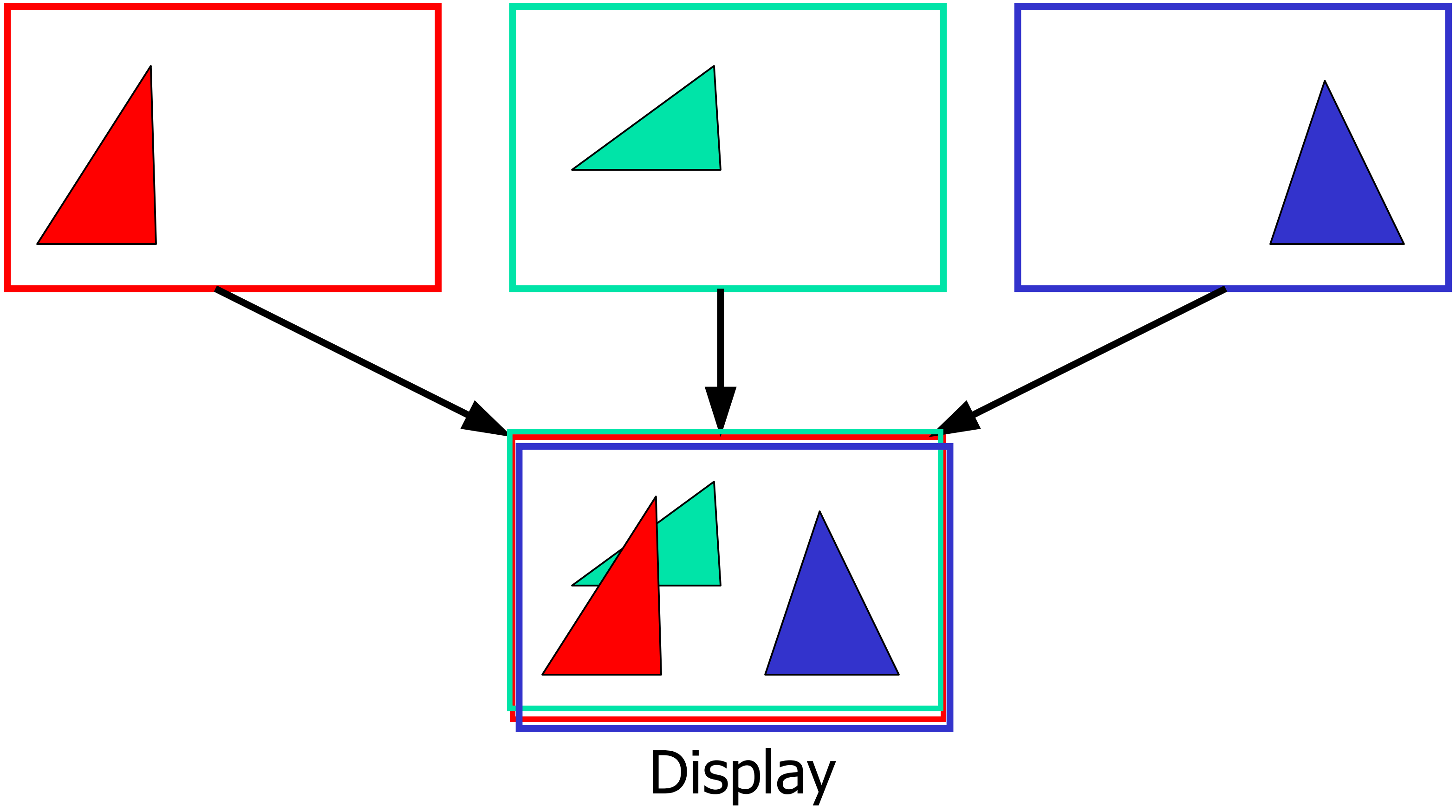
- Scalable Display Wall (Princeton)
  - Myrinet & sort-first
- WireGL (Stanford)
- Sepia (Compaq)
  - ServerNet II & custom compositing
- Meta-Buffer (UT)
- Lighting 2 (Stanford)

Metabuffer Features

- Independently scalable number of renders and display tiles
- The viewport of a render can locate anywhere in the display space
- Viewports can overlap
- Viewports can be different size (multi-resolution)



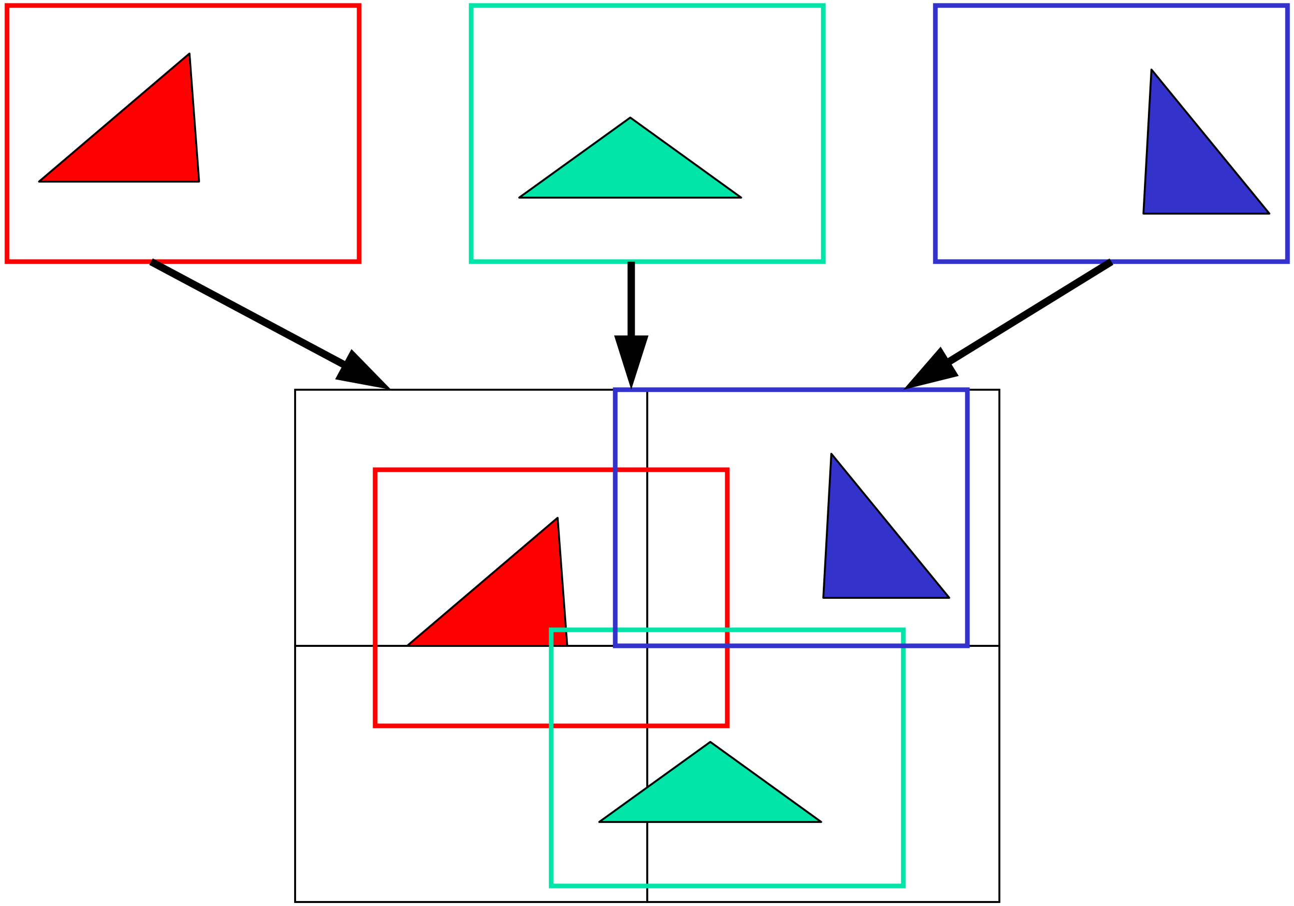
Configuration I



Configuration I

- Each Renderer has the same viewport
  - Polygons can be assigned to any renderer
  - Display has the same resolution as a rendering process
- Load balance for isosurface rendering
  - Each processor generates similar number of triangles
  - No need to redistribute triangles
  - Efficiently use memory as cache for change of viewpoint

Configuration II



Configuration II

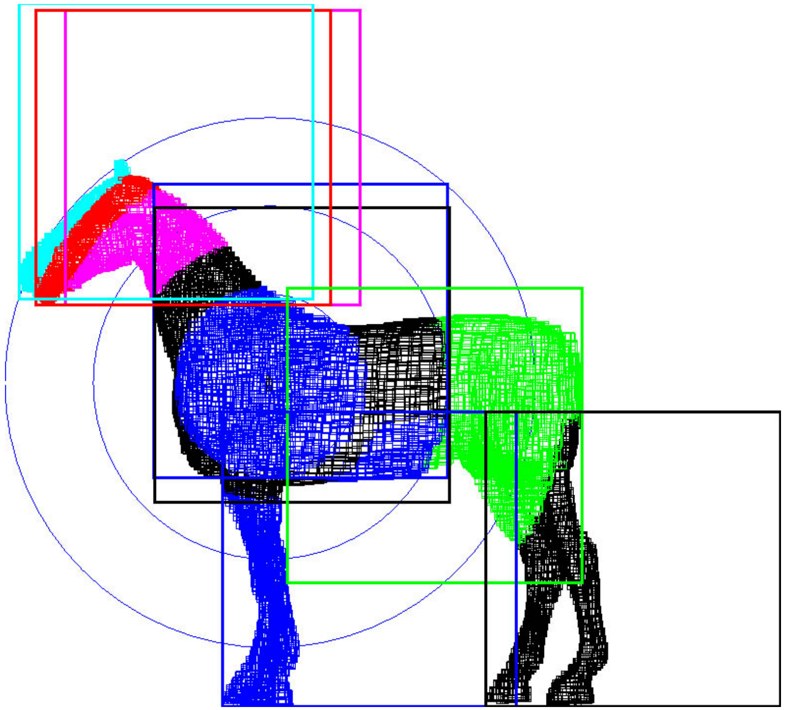
- Each renderer has a viewport with the size of a tile
  - Faster rendering and higher resolution on large display
  - Independent number of renderers and tiles
  - Combination of sort-first and sort-last
- Load Balance
  - Polygons cannot be assigned arbitrarily
  - Viewports are positioned with constraints
  - Load balance among the viewports
  - Different viewport locations for different view parameters

Viewport Positioning Problem

- Conditions
  - m rendering server to cover n tiles (m > n)
  - Each tile has the resolution $w \times h$
  - Each server renders C triangles/sec
  - T triangles in the scene
- Constraints
  - The viewport of each server has the same resolution $w \times h$
  - servers only render triangles in their viewports
  - Every triangle is covered by the union of viewports and rendered by at least one server
- Best time: T/(m*C); worst time T/((m-n+1)*C)
- NP-hard. Have to use approximation method

Greedy algorithm

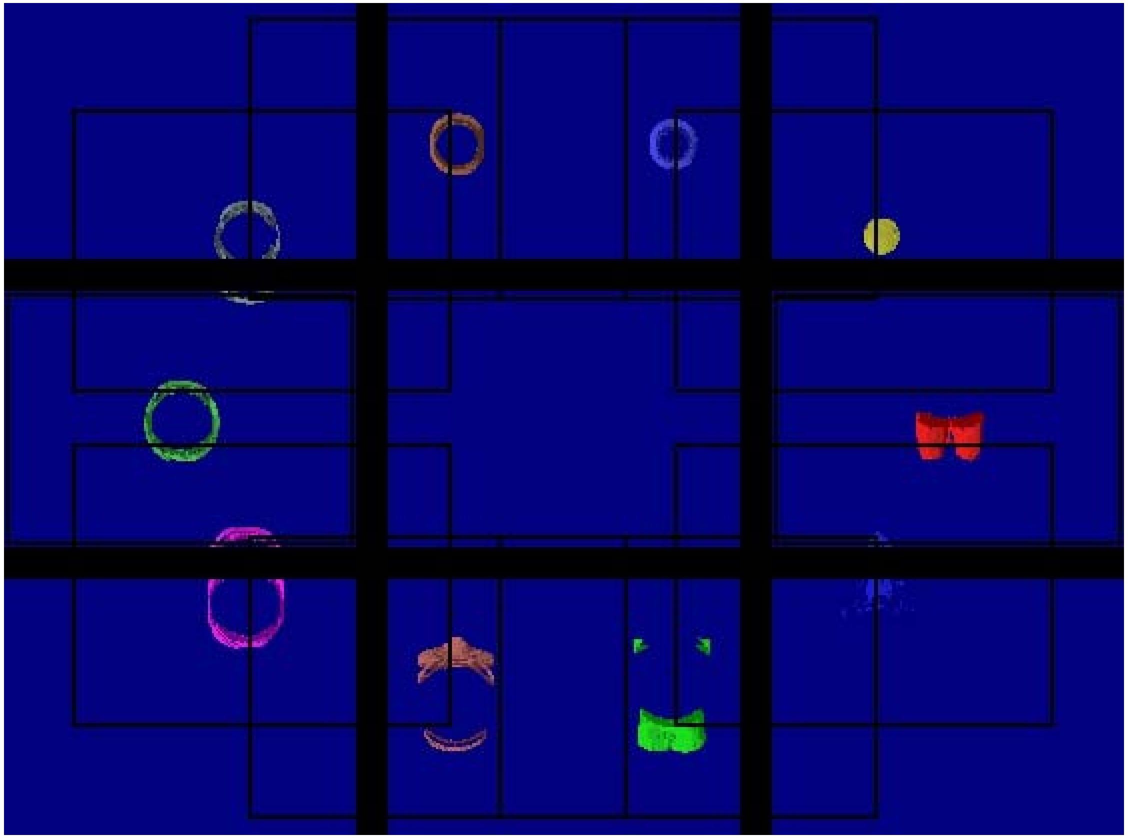
- Find the center of mass of all triangles.
- Sort triangles by the distance to the center of mass
- Each triangle is assigned to a viewport in the order of decreasing distance
  - Create a new viewport if no viewport can cover the triangle
  - If multiple viewports are applicable, chose the one with least mobility
  - Close a viewport if its triangle count exceeds a threshold
- Iterate the viewports to move triangles from over-loaded ones to under-loaded ones



Progressive Image Composition

- It is slow to recompute viewport positions and redistribute polygons when view point changes
- Change the resolution of viewports for time-critical rendering
  - The Metabuffer supports multi-resolution
  - Initially polygons are well-balancedly distributed
  - When the user navigates, viewports are enlarged to encompass its assigned polygons. Thus those renderers still renders at the same rate but with lower resolution
  - When the user pauses at some viewpoint, polygons are reshuffled to reduce viewport sizes.

Movie

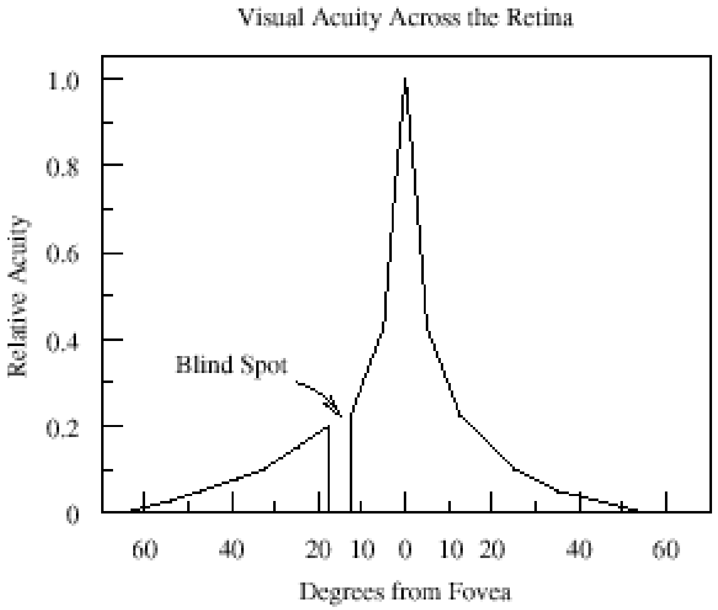


Multiresolution Multi-Tiled Displays: Human Vision

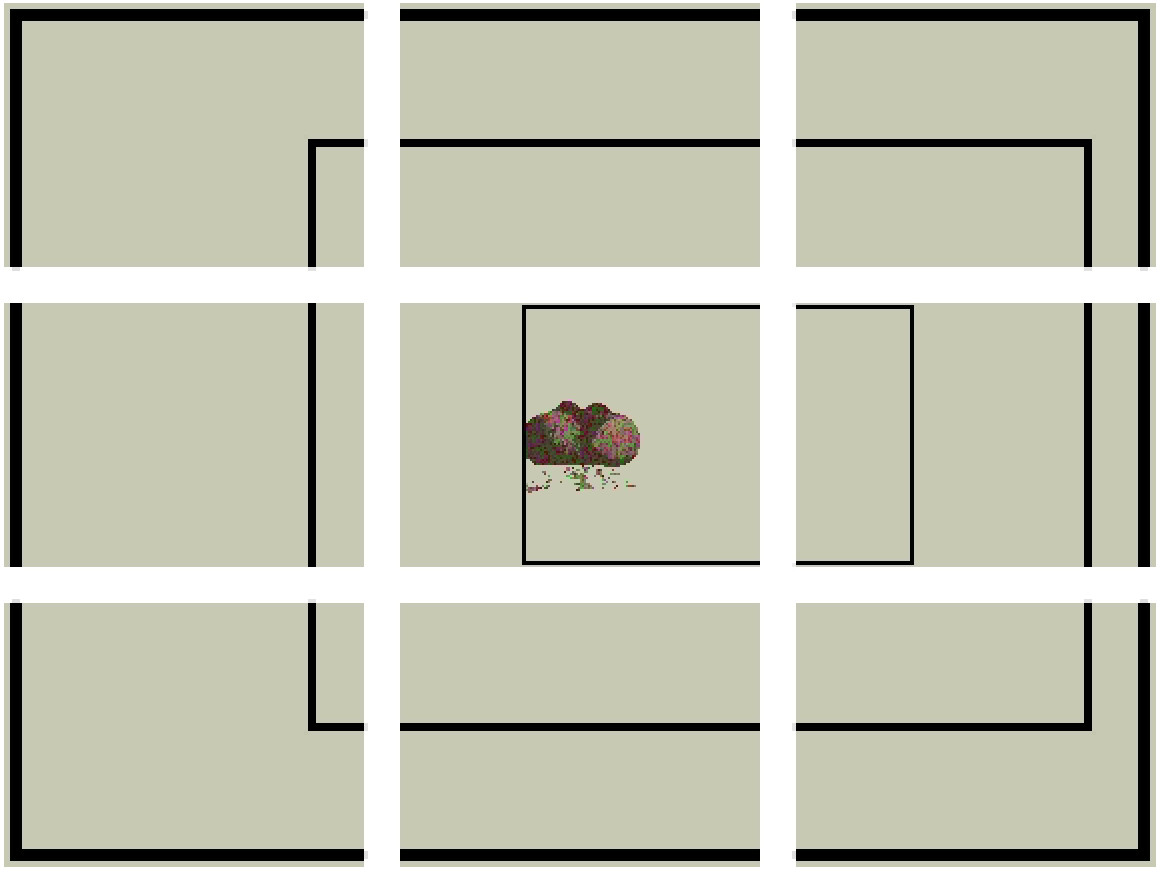
- Peripheral vision
  - Not sensitive to detail
- Huge multitiled displays
  - Only small percentage viewed
- Gaze of users
  - Concentrate rendering resources
- Periphery
  - Rendered in low resolution

Visual acuity

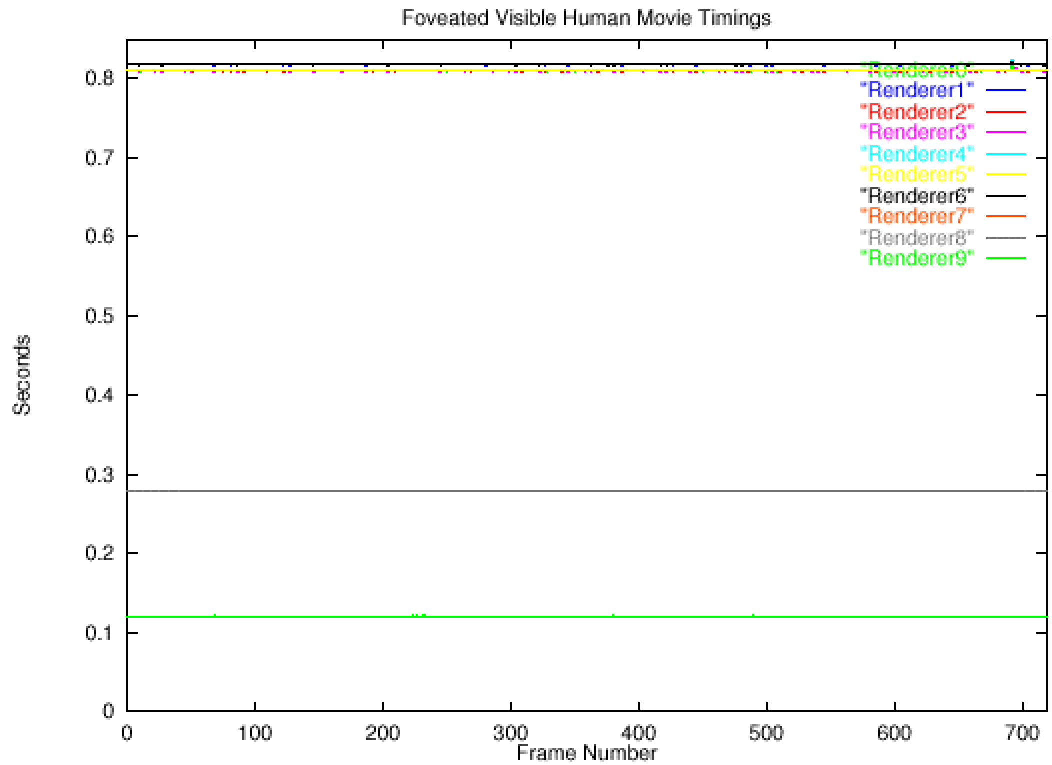
- Continuous
  - Dynamic assignment
  - LOD and resolution
  - Generalized ROI
    - Frequency
    - Distance
    - History
- Discrete
  - Points on graph
  - Static assignment



Active Visualiztion: Male VH (9,128,798)



Male Timings



Conclusion

- A end-to-end scalable parallel framework
- Parallel Multi-resolution Isocontour extraction
  - Load balanced and completely out-of-core
  - Minimum data replication
  - View-dependent isocontour extraction
- Parallel Multi-resolution rendering
  - Load balance for different configurations
  - Progressive image composition for time-critical rendering
  - Foveated resolution display

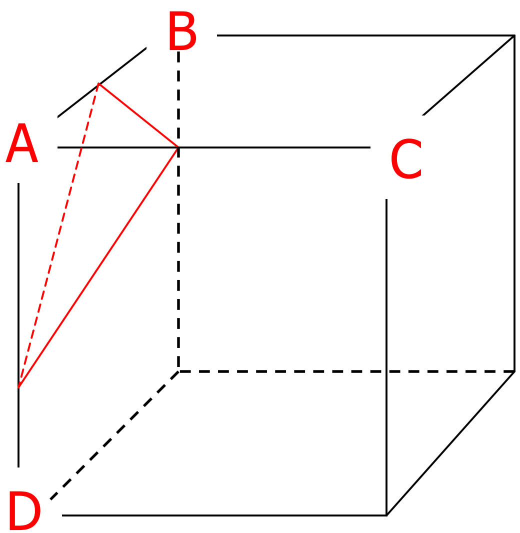
Compression

- Output from large dataset is also large
  - Store isosurfaces in compressed format to save storage space
  - Use compression to save communication between computational servers and rendering clients
- Post-extraction surface compression is usually expensive
  - Extract isosurfaces in compressed format
  - allow incremental decompression and rendering

Surface Compression

- Turan ’84
- Deering ’95
- Chow ’97
- Taubin/Rossignac ’96
- Touma/Gotsman ’98
- Bajaj/Pascucci/Zhuang ‘99

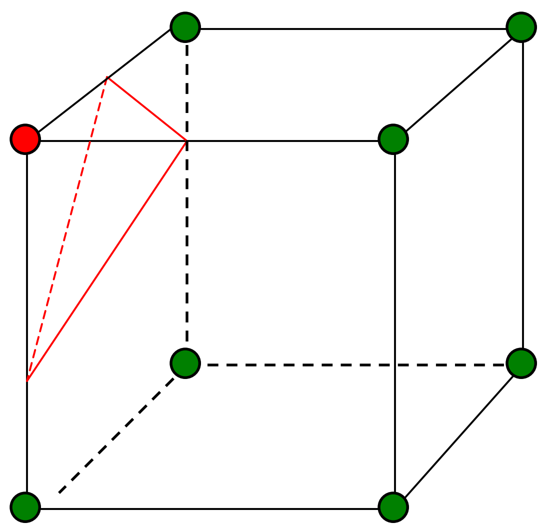
Edge Index



To reconstruct the red triangle, one only needs to know function values at vertex A, B, C, D and indices of edge AB, AC and AD

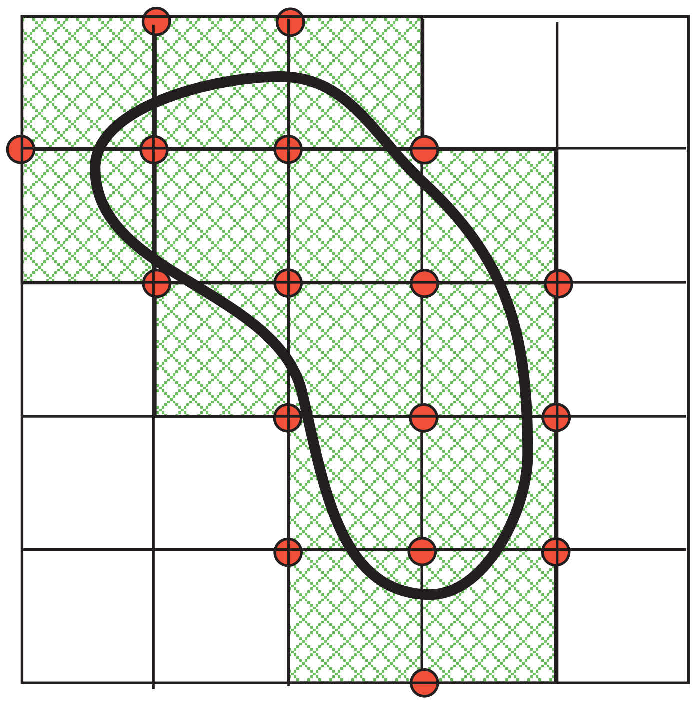
Cell Configuration

- The configuration of a cell can be derived from the function values and indices of its relevant vertices



- > isovalue
- < isovalue

2D Example

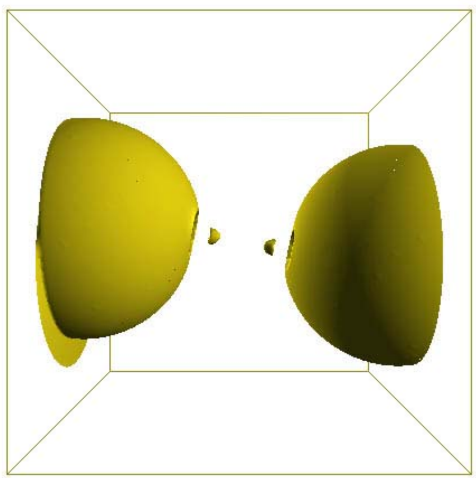


Red Vertices: Relevant Vertices
Green Cells: Valid Cells

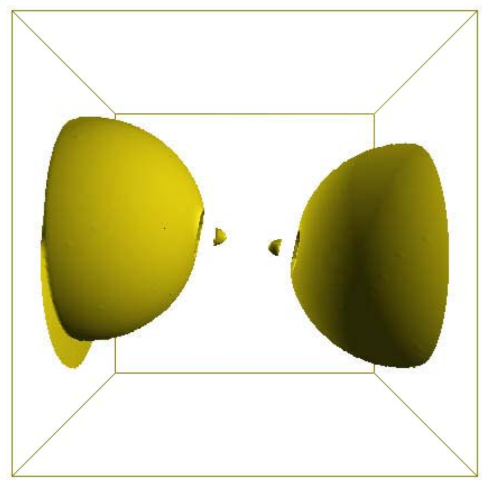
Advantages

- **Compressed Output**: Isosurface is extracted directly in compressed format
- **Minimal Memory requirement**: Only two slices are needed in memory at any time
- **Compressed Input**: Each slice may be stored in compressed image format (jpeg)
- **Incremental Transmission**: It can be transmitted and decompressed incrementally

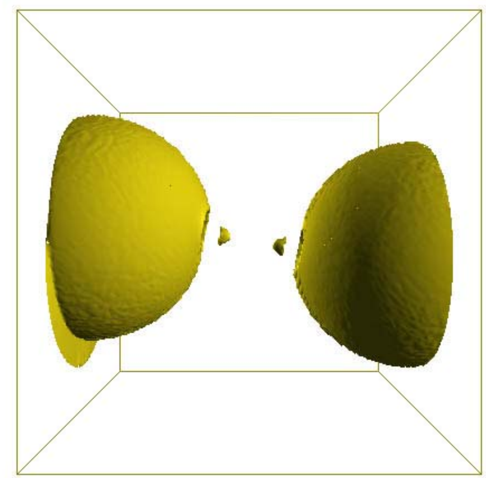
Guaranteed Right Topology



14 bits/vertex

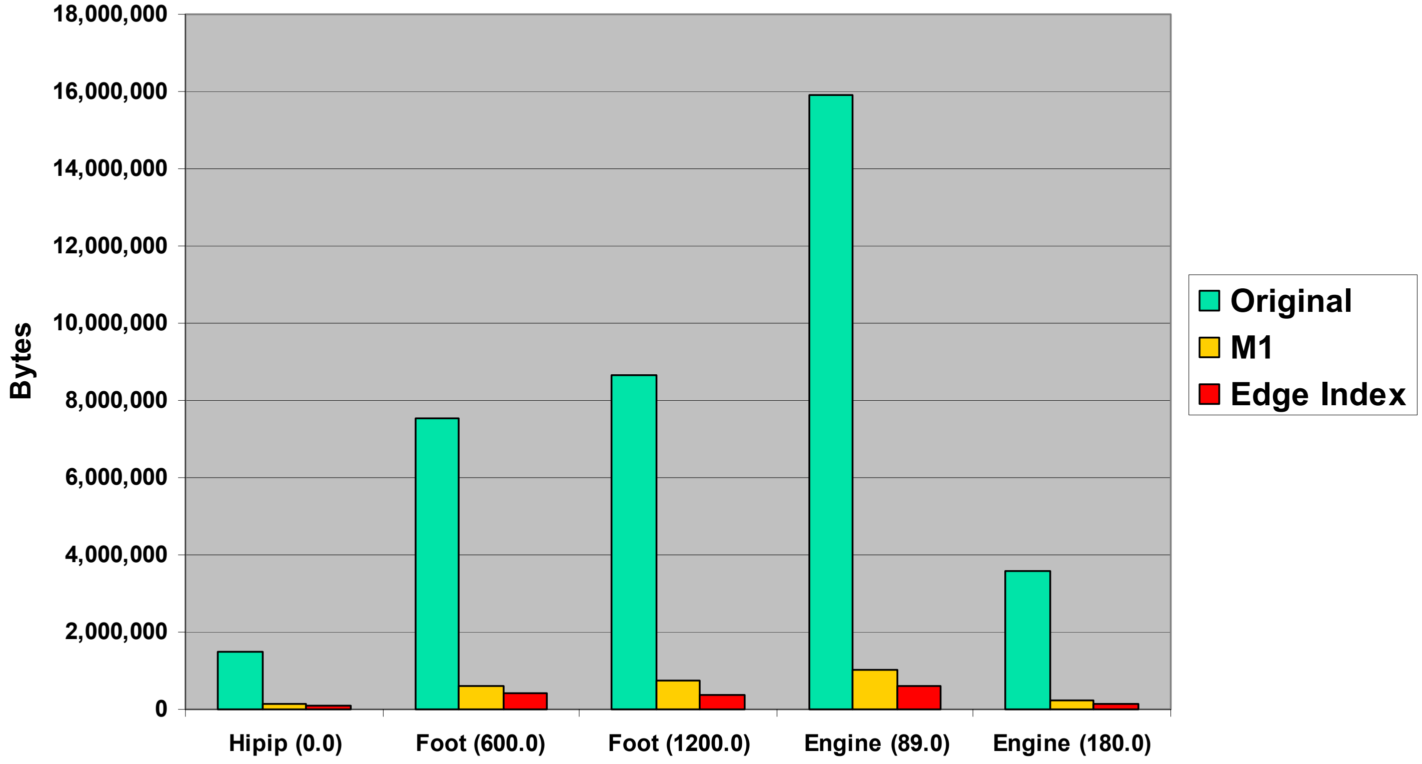


10 bits/vertex



6 bits/vertex

Compression Results

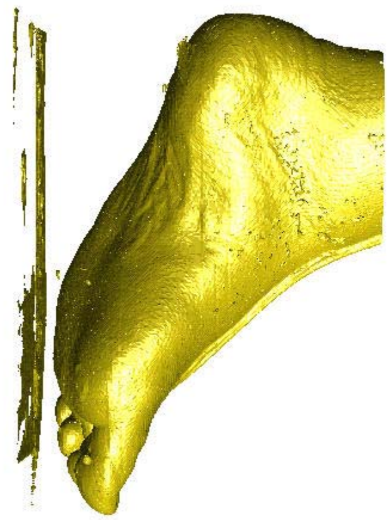


M1 alg. is in Bajaj/Pascucci/Zhuang '99 and uses 8 bits/vertex

Compression Results

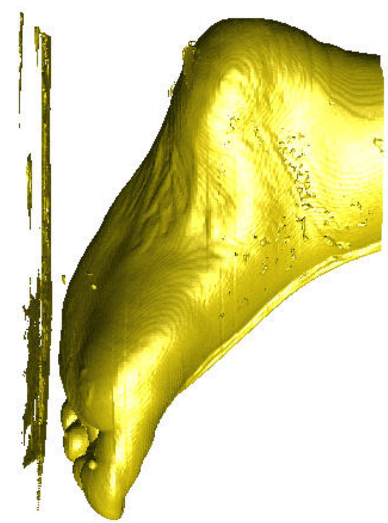
Original

7,536,227 bytes



M1 Algorithm

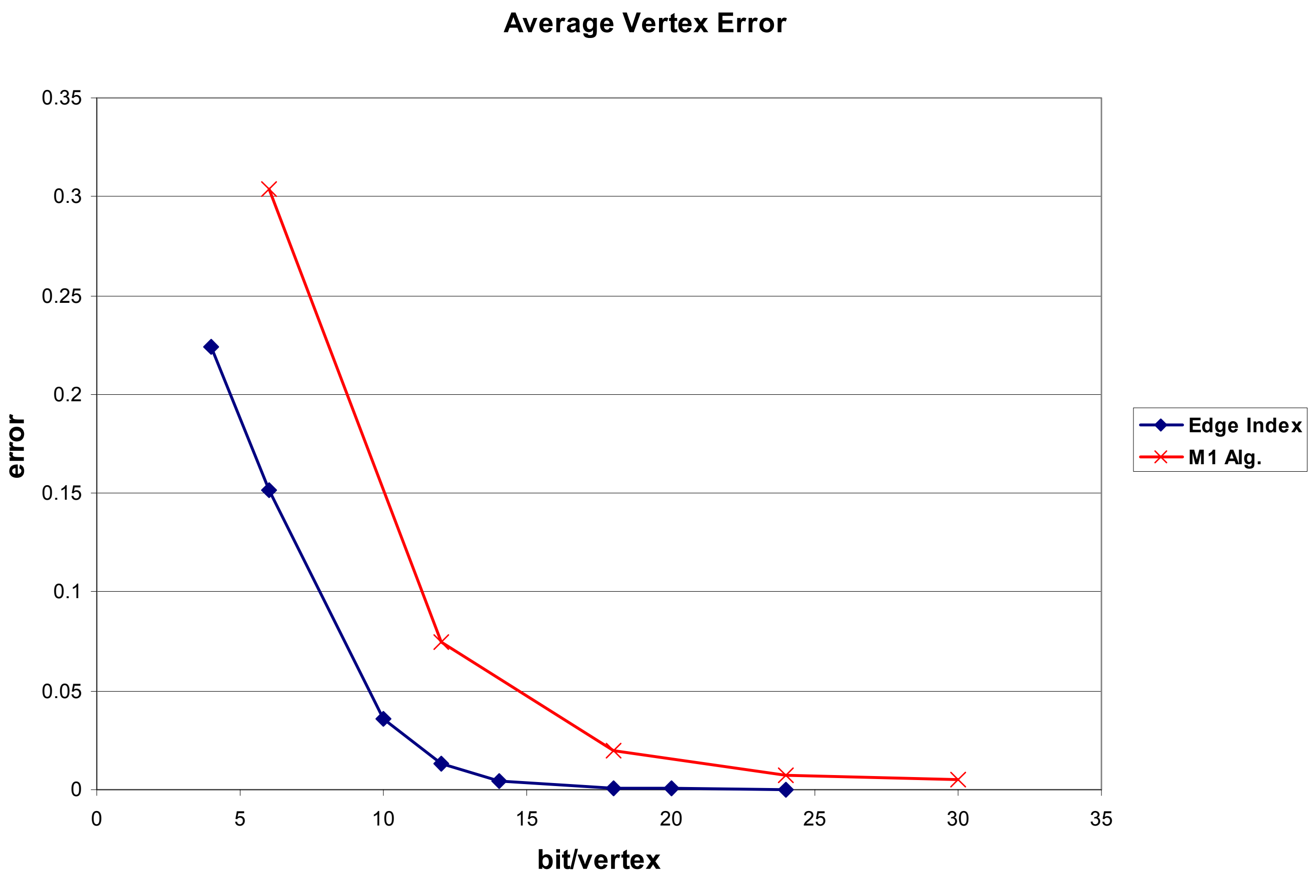
587,344 Bytes



Edge Index

407,658 Bytes

Average Error



Blackhole data (isovalue = 1.23)