CSCS

Swiss National Supercomputing Centre

Parallel visualization applications delivered to remote users

Jean M. Favre

Data Management, Analysis and Visualization Group

Motivation

- Provide a visualization service to remote users with requirements for large data exploration
- Integrate the offering with the current computing solutions at our Center
- Gives users the same resource allocation schemes they are used to.

Outline

- Resource allocation
  - Graphics server
  - Data Access
- Remote and parallel visualization
  - Remote Access
  - Multi-user sessions
  - Multi-node sessions
  - Data distribution
- Distributed Rendering
  - VTK Composite Renderer
  - Equalizer Graphics
  - HP Compositing API
- Conclusion

The HPC systems at CSCS

Cray XT3 1664 dual-core Opterons

Cray XT4 500 dual-core Opterons

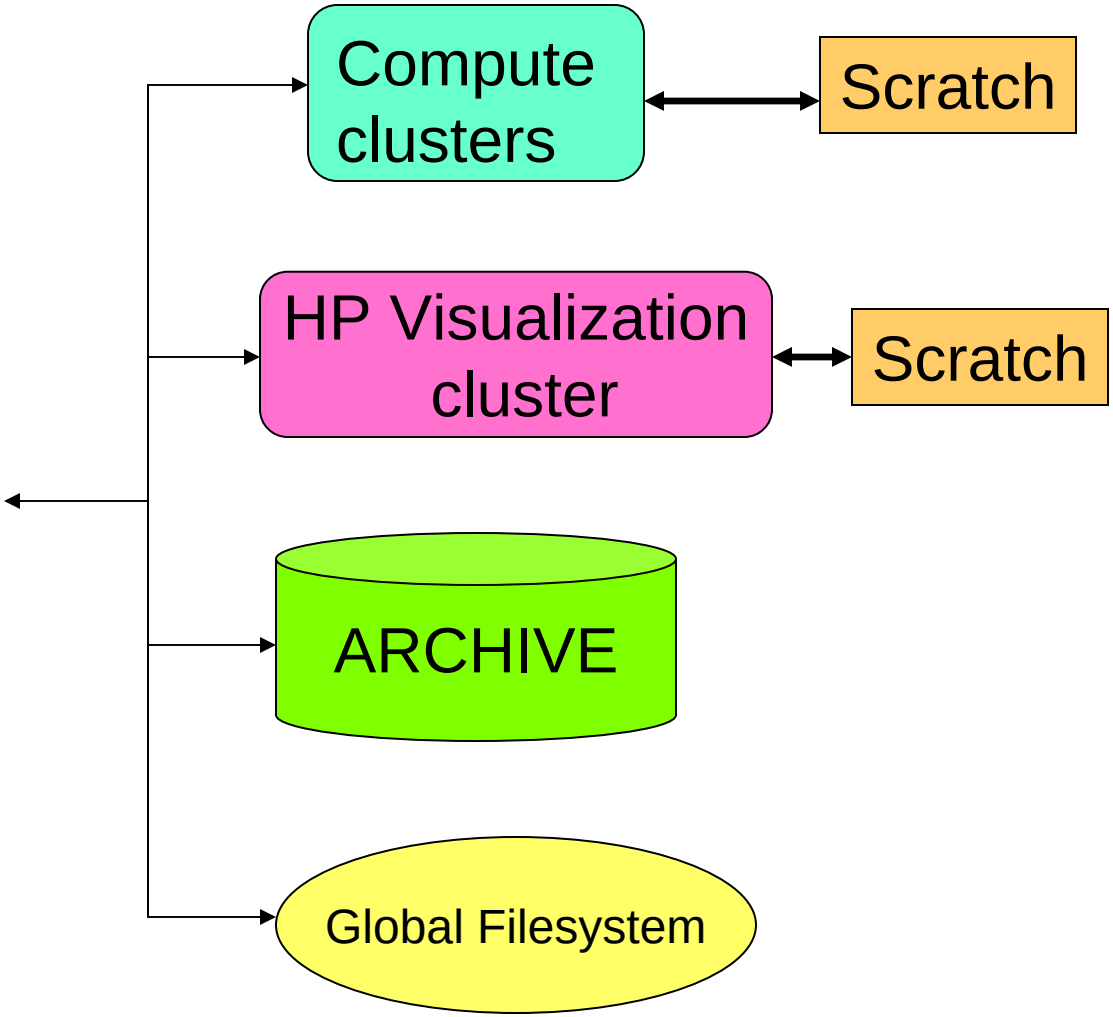
Sun (400 Opterons, TIER-2 in the LCG)

IBM 748 Power 5

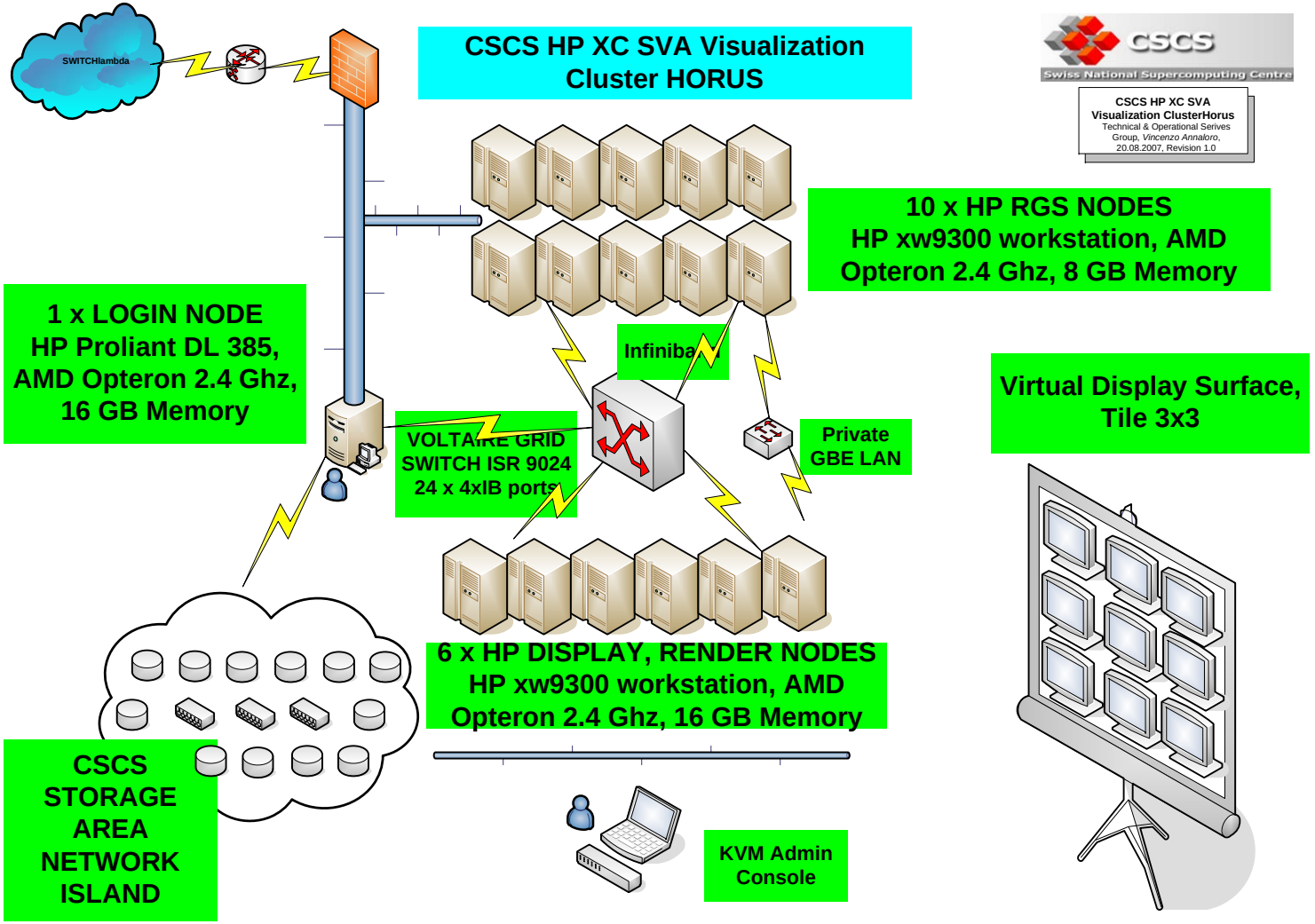
HP Cluster HORUS (32 Opterons)

Global Parallel File System

- GPFS file system
- IB 10 Gb main interconnect between IO and client nodes
- 100TB today and up to 500TB in the next year
- Test performance we reached as single jobs 740MB/s and up to 1.3 GB/s in parallel jobs
- Based on the DDN 9950



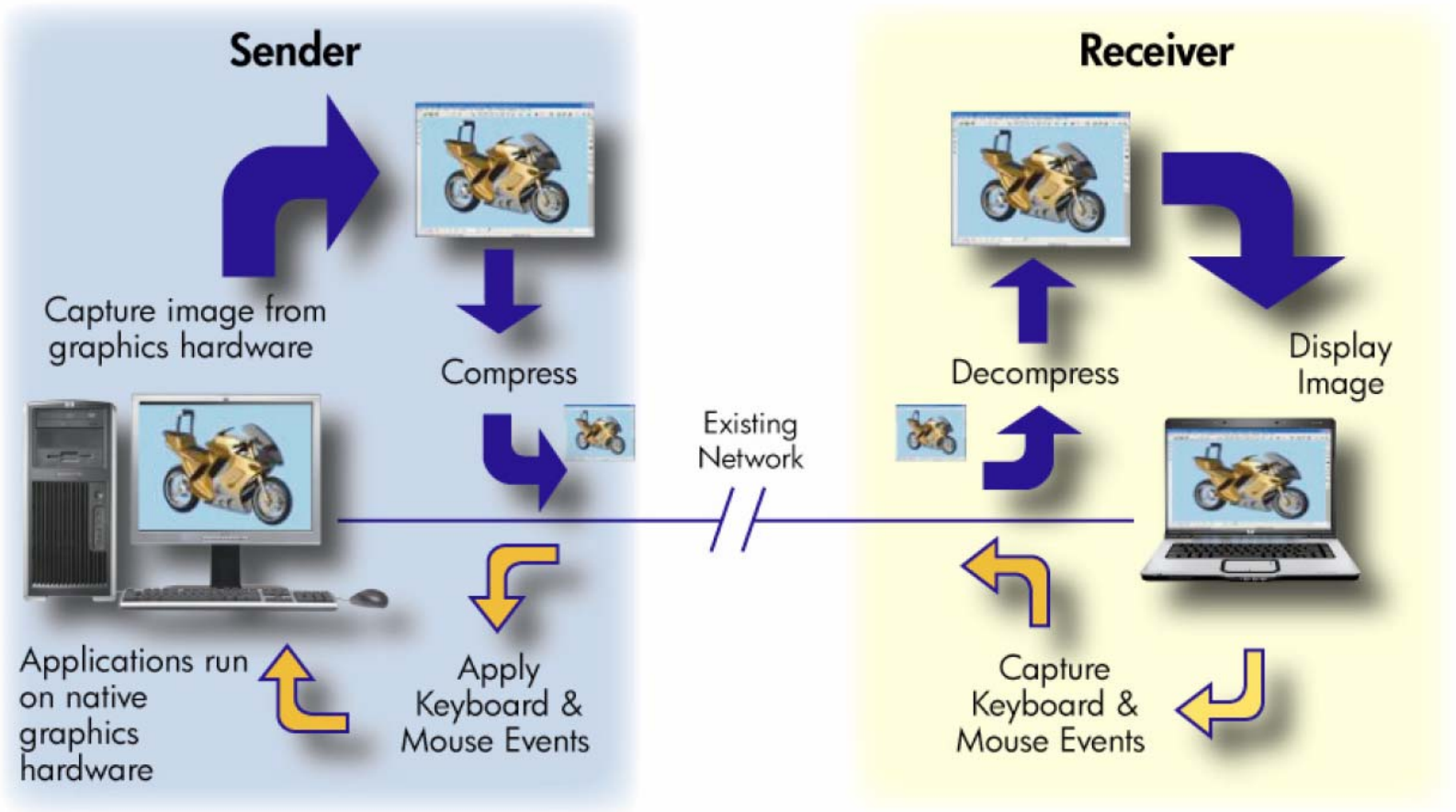
The HP SVA Visualization cluster



HP Remote Graphics Software (RGS)

- An innovative software that enables real-time remote access to workstation desktops and share it over a standard network.
- Remote visualization enables:
  - Remote access/demo
  - Remote team review
  - Remote user application support/training
  - Centralization and consolidation of desktop workstations

1 sender + 1 (or more) receivers



RGS transfers only final image data (pixels) over the network. No part of original data is sent.

Technical advantages of HP RGS

- Utilizes 3D hardware rendering capability of sender system and does not burden CPU
- Rendering and image capture are tightly linked and optimized while maintaining complete application transparency (no modification necessary with application to remotely use)
- Image compression technology applies different compression algorithm to maintain good balance between performance, image quality, and compression ratio.

Multiple scenarios of use

The HP Scalable Visualization Array is managed by SLURM

- Interactive single-node remote visualization
- Interactive broadcast mode (teacher + students)
- ……..(shared controls)
- Multi-node rendering + remote image display

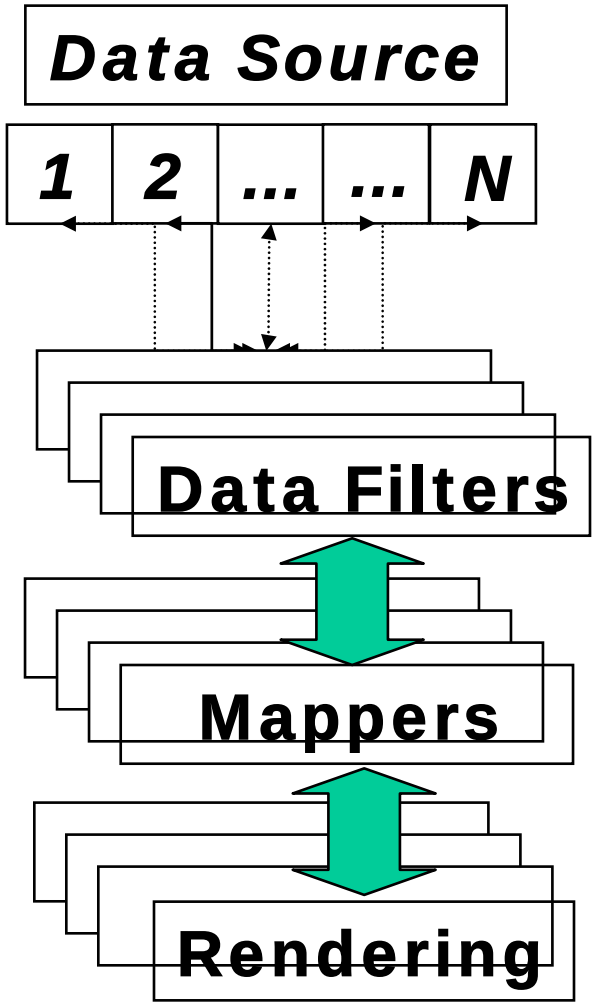
- Multi-node client-server applications with ParaView, with client outside of HP cluster
- 3x3 tiled display at CSCS

Users come in on a first-come first-serve basis

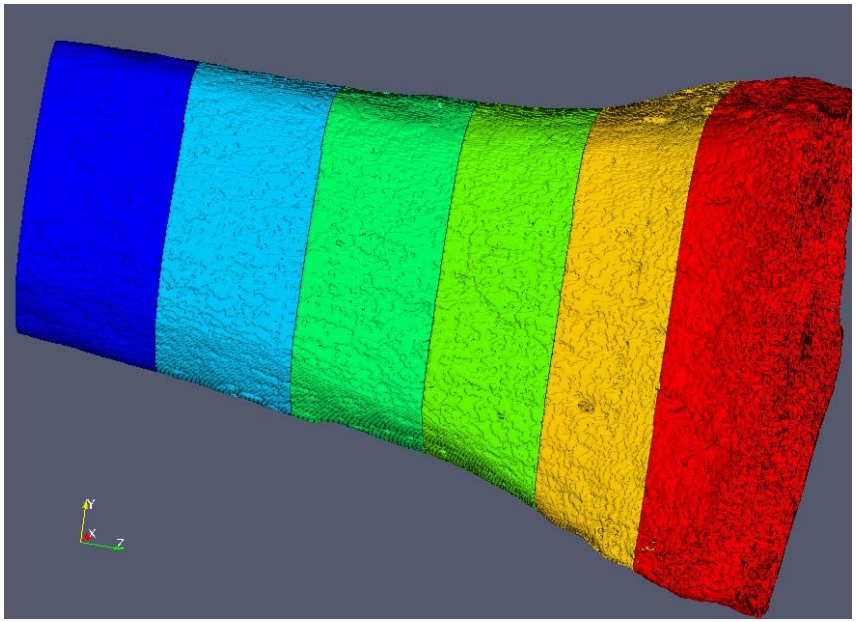
- The X servers are killed and restarted between each use
- The X server can be started with 1 or 2 DISPLAYs
- Nodes are moved to a compute queue during *night-time*

Parallelism in Scientific Visualization

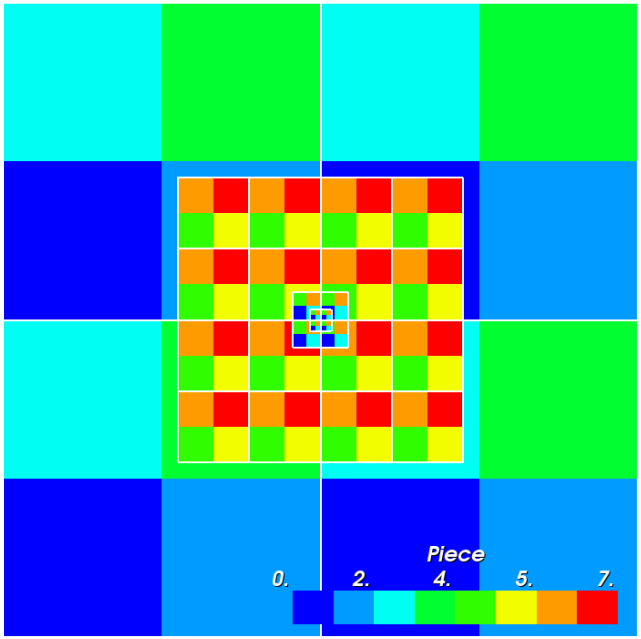
- Data parallelism most everywhere
  - Data I/O
  - Processing
  - Rendering
- Time-parallelism (new)
  - See IEEE Vis'2007 "*Time dependent processing in a parallel pipeline architecture*", J. Biddiscombe et al.



Data partitioning & load balancing examples

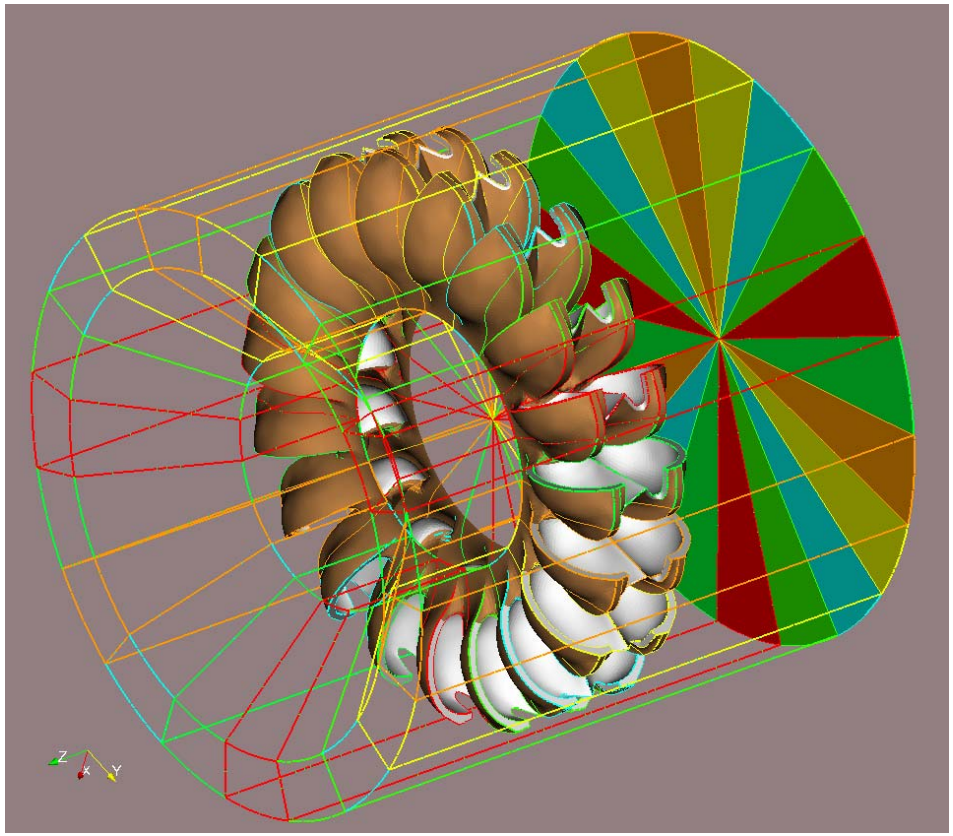


The mesh is split in multiple pieces of size(# of cells / # cpus)

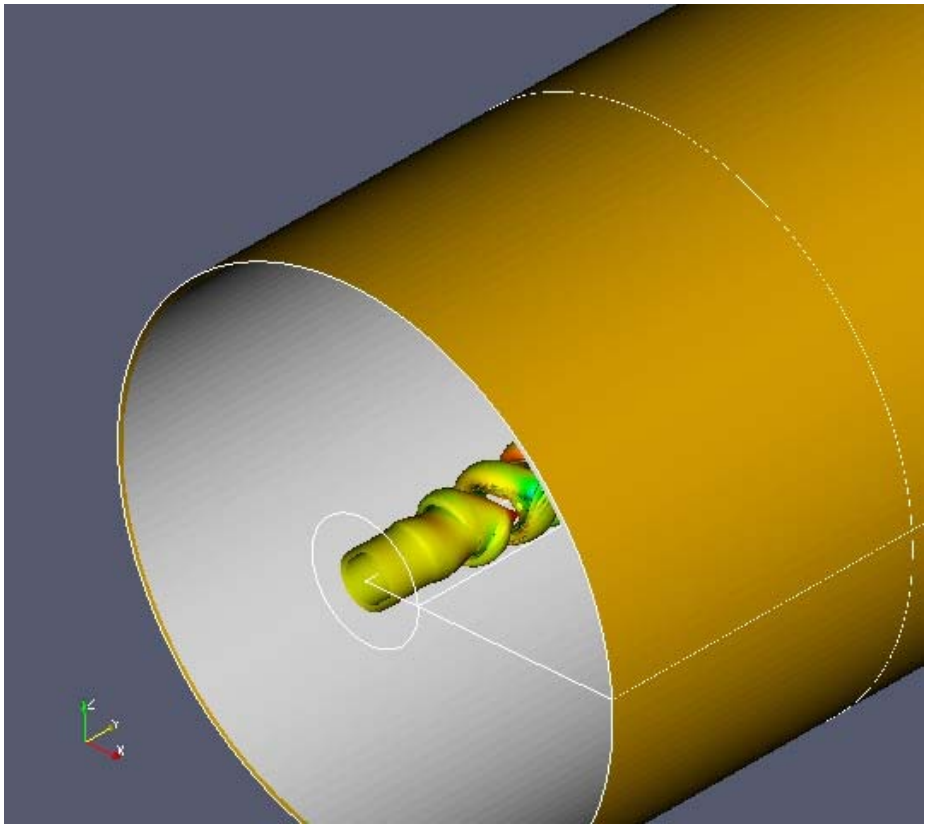


The AMR mesh refinement is naturally mapped to different cpus

Data partitioning & load balancing examples

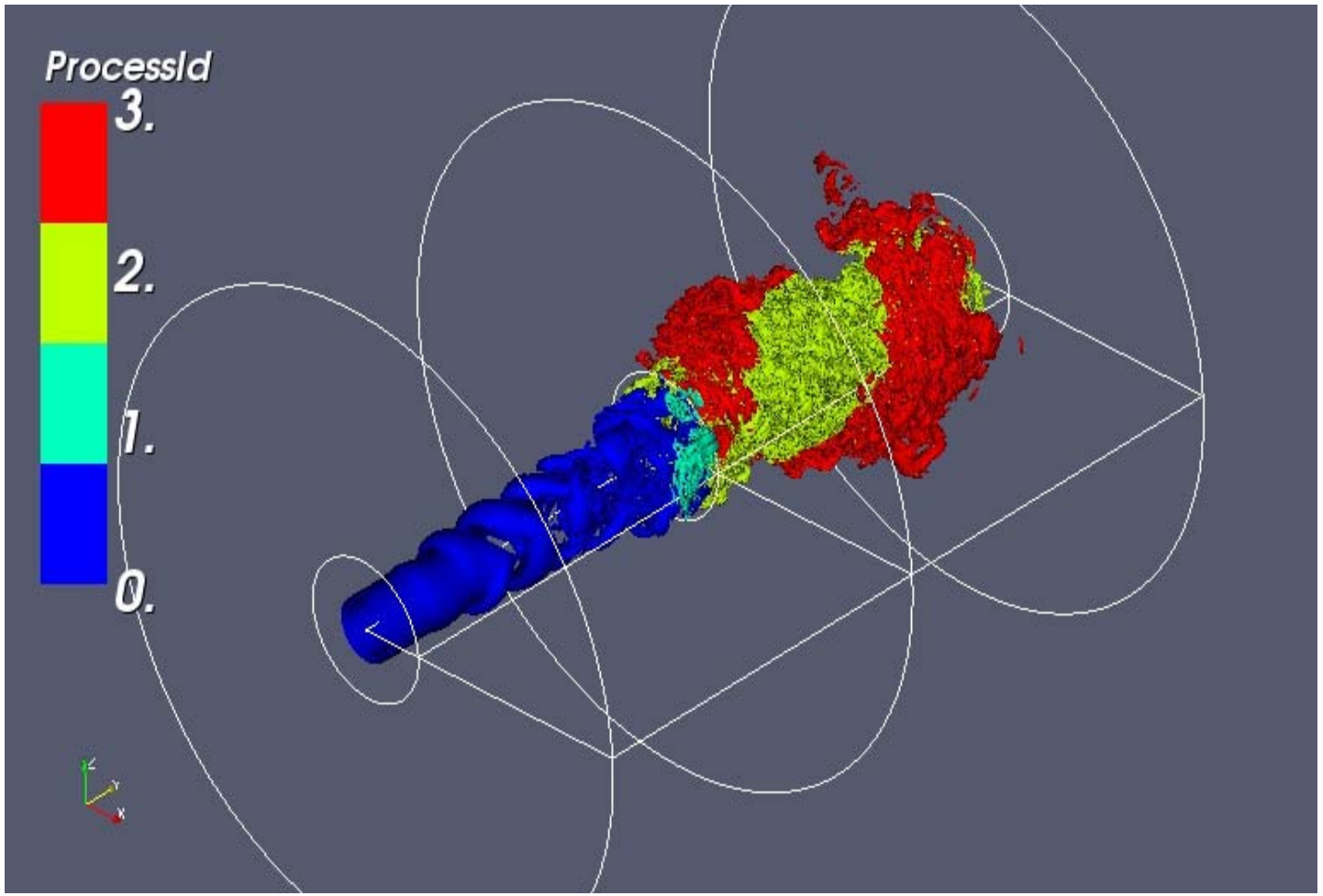


The mesh partitioning from the solver is re-used (ANSYS CFX)

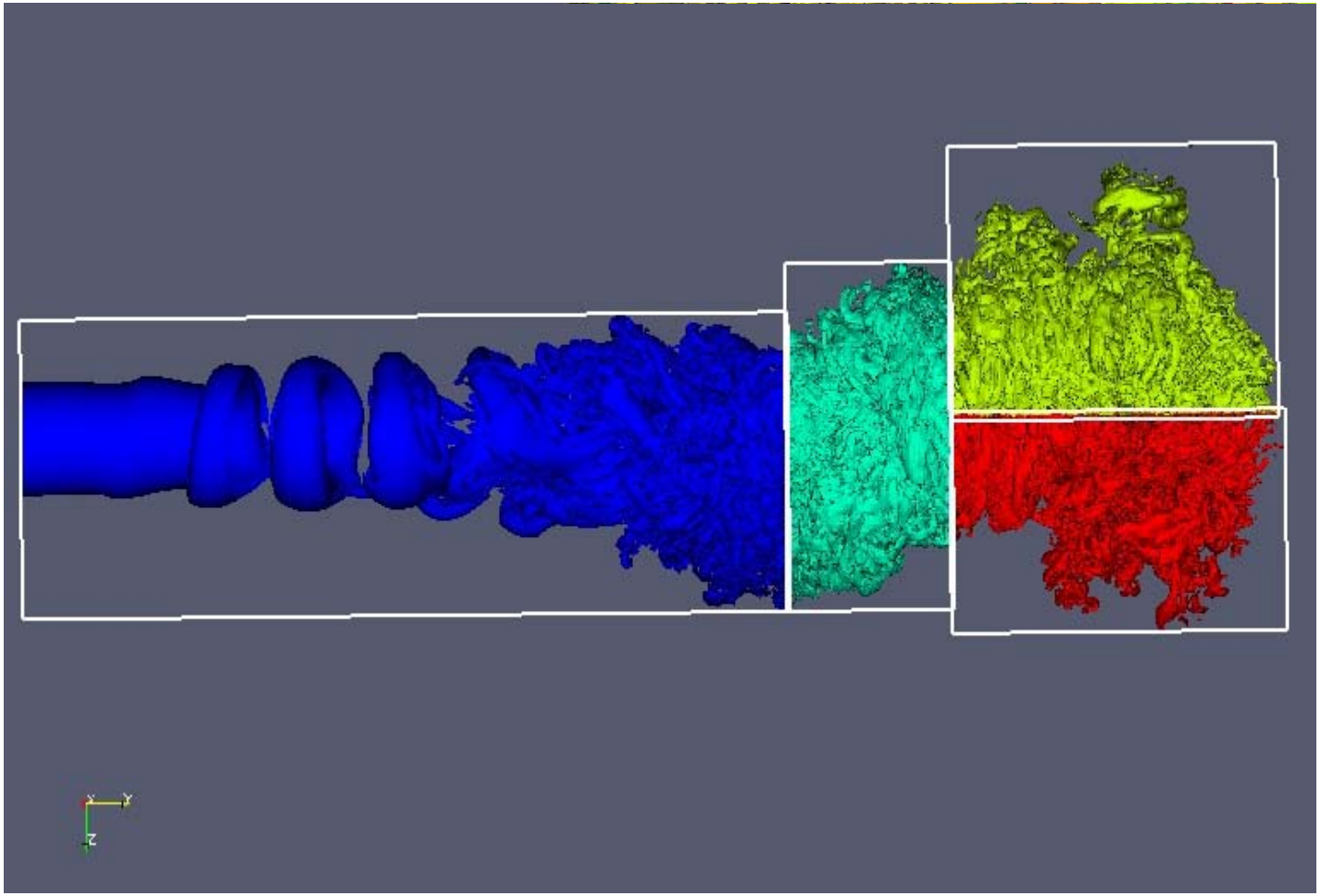


Cylindrical coordinates will require a custom-made mapping

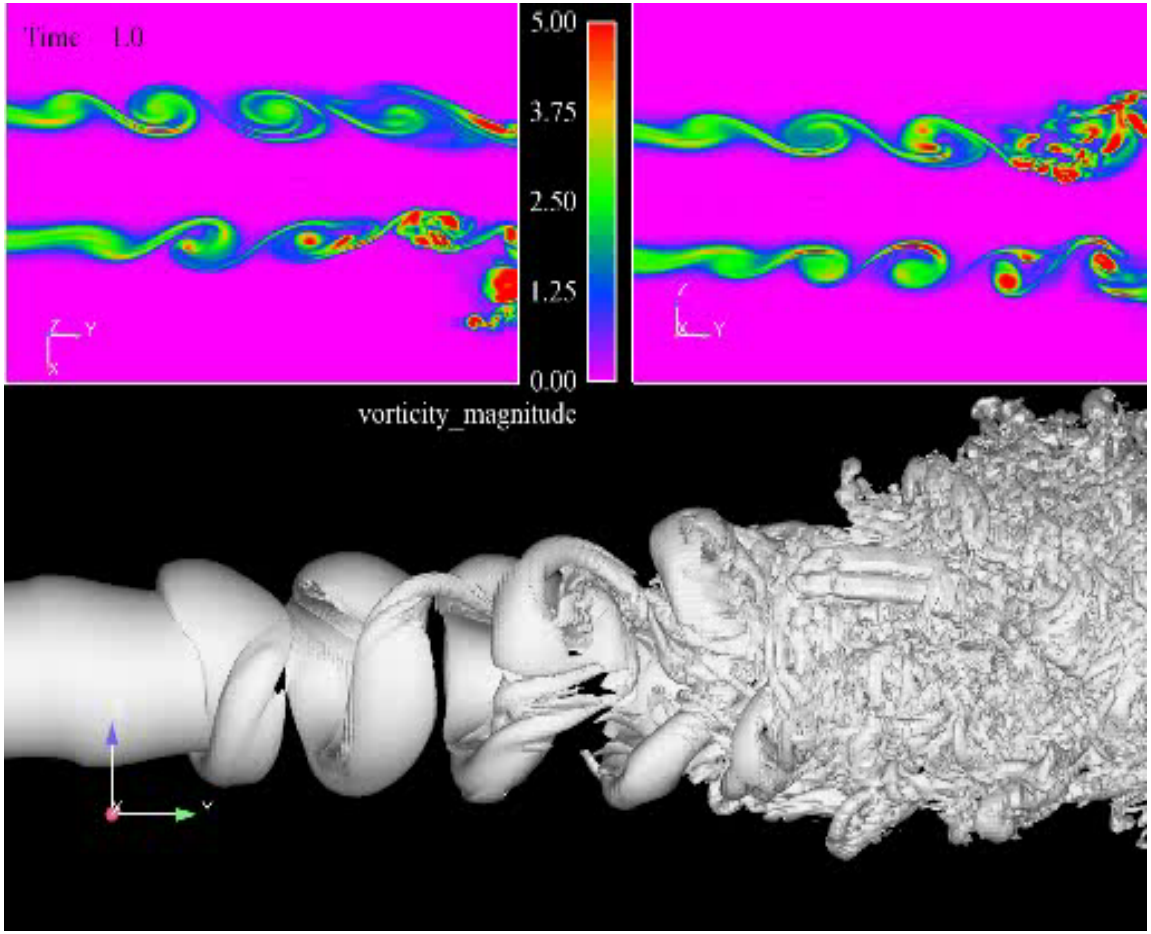
Default Load Balancing



3D tree load balancing



Swirling Flow Animation



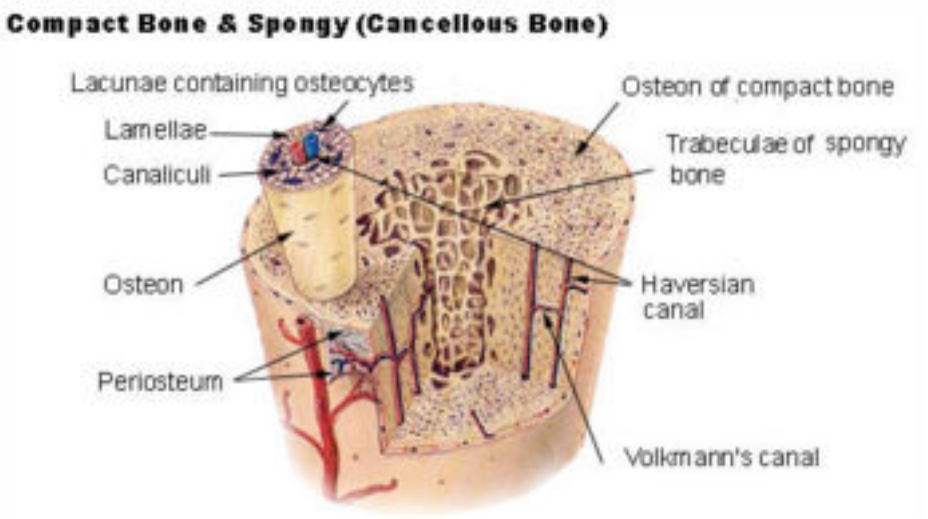
Once the data load balancing is optimised, the user does his own movies from his remote desk.

Practical Examples

- Applications
  - Bio-medical
  - Astro-physics
  - Geo-physics

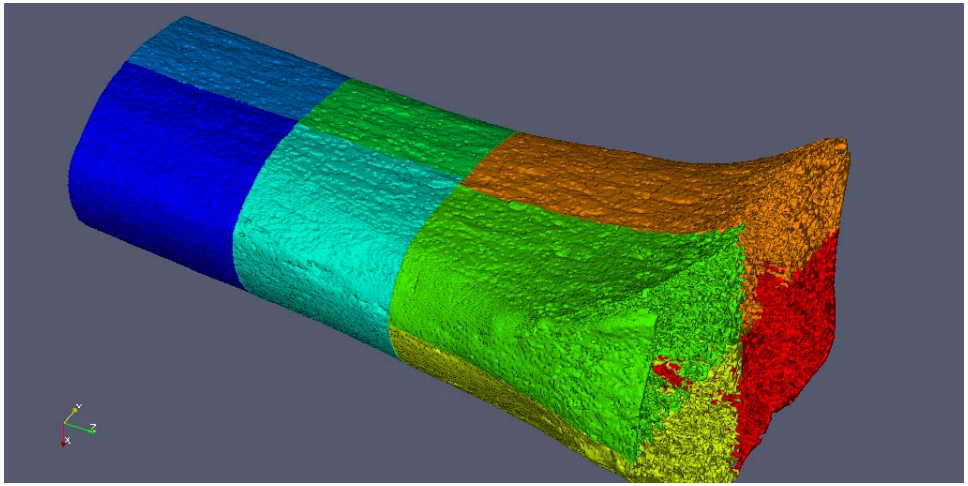
Finite Element data visualization

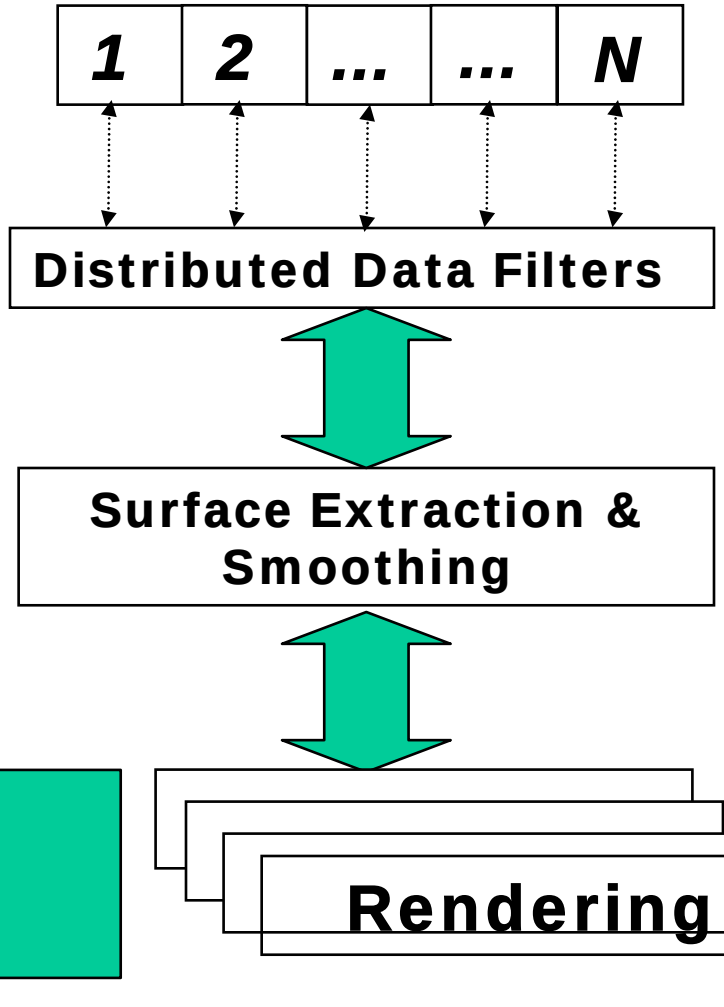
- Simulation of bone loading involves very large finite element meshes obtained from CT-scans of bones.
- Trabecular bone is a type of osseous tissue with a low density and strength but very high surface area, that fills the inner cavity of long bones.
- http://parfe.sourceforge.net/



- ParaView support include
  - Distributed and Parallel I/O
  - Ghost-cells at the interfaces
  - Interactive rendering of large polygonal meshes

The overall Paraview visualization pipeline

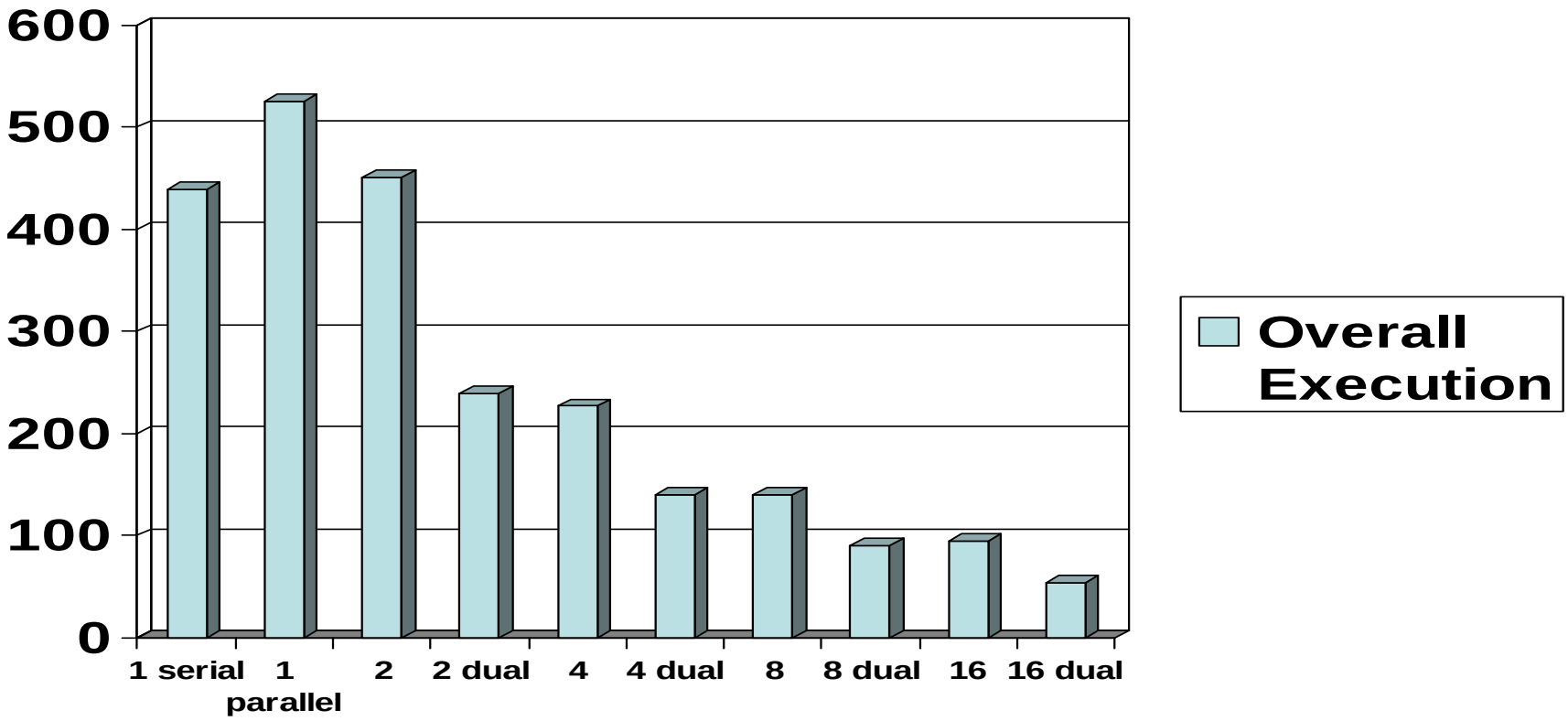




Three benchmarks

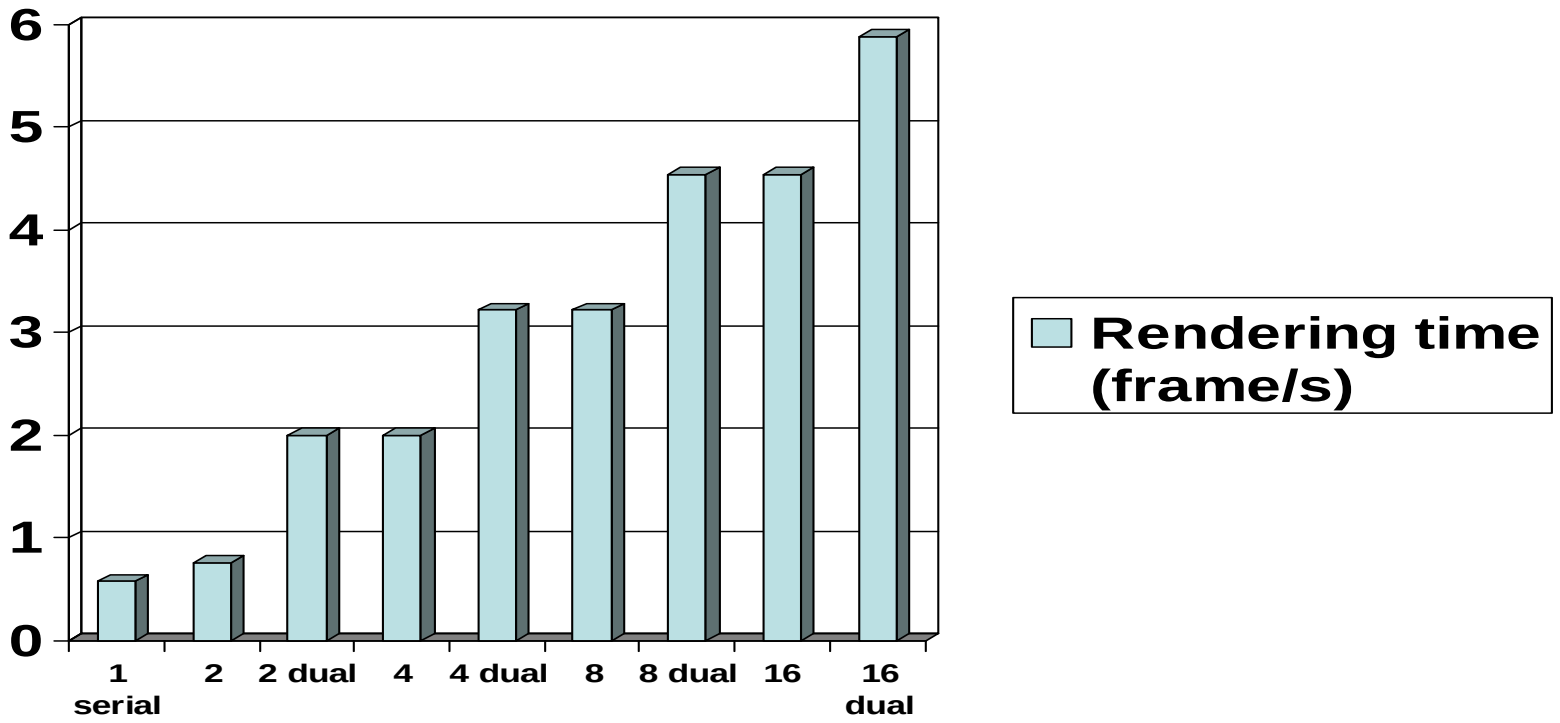
- Adding more processing/rendering power for a given dataset
  - Surface is 3.8 million pseudo-colored quadrilaterals
- Increasing the size of the dataset
- The largest dataset available

Benchmark 1 (execution of the full pipeline)



- The left-most column (serial) does not include a data repartitioning phase
- All „dual“ columns use dual cpu/dual gpu per node on our HP SVA cluster.

Parallel Rendering Times



- Immediate mode rendering in 1024x768 pixel buffer

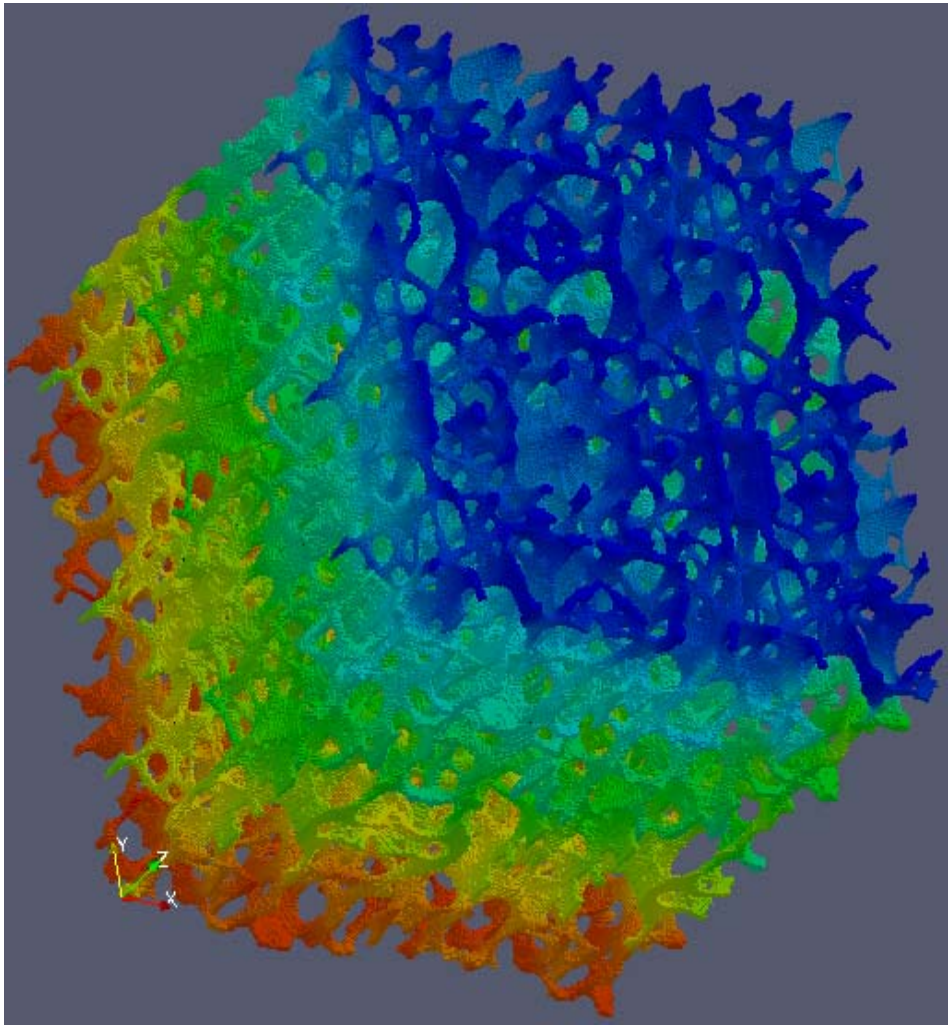
Benchmark 2

Surface of increasing size:

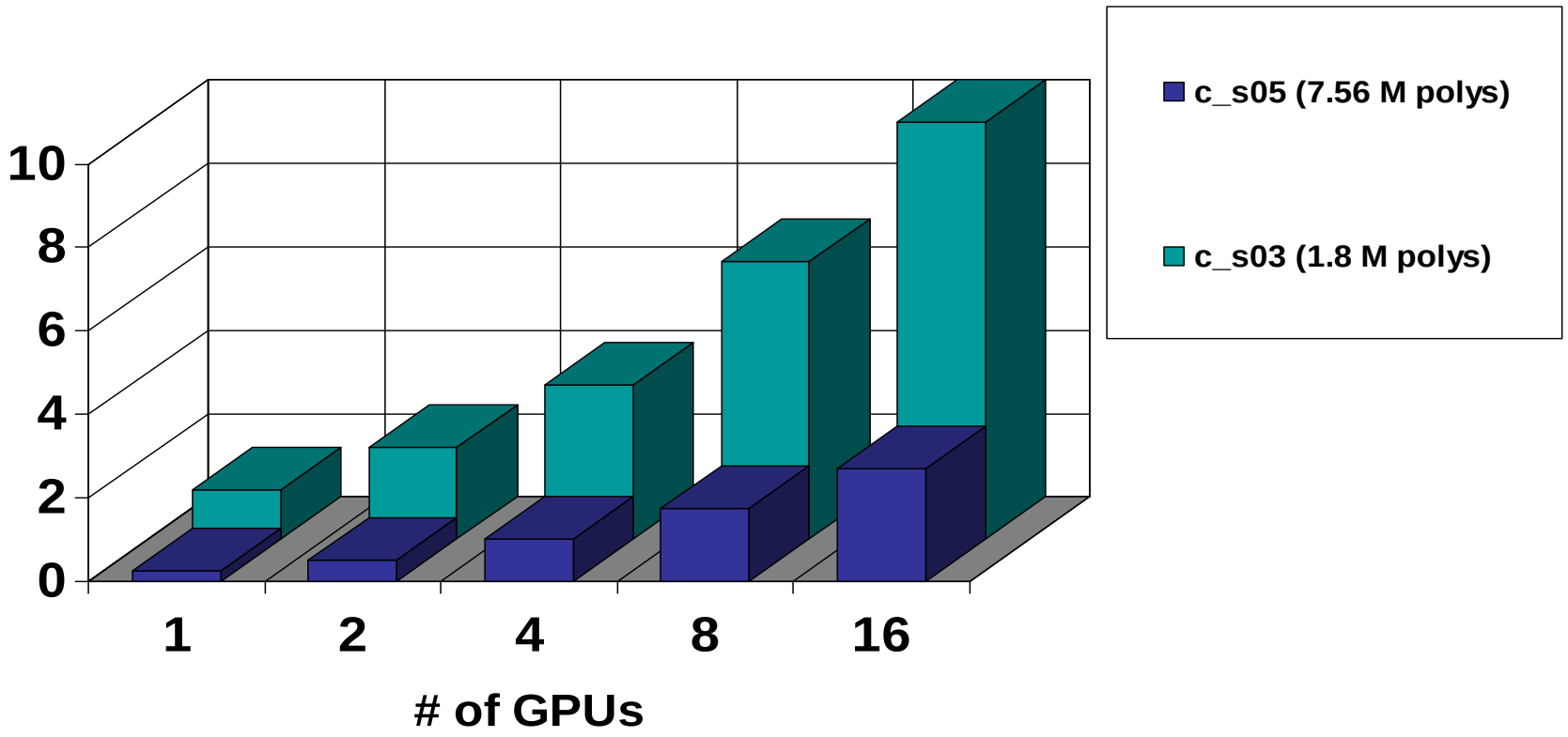
- 1.8 M polygons
- 7.5 M polygons
- 22.9 M polygons
- 46 M polygons

- 1, 2, 4, 8, 16 graphics cards

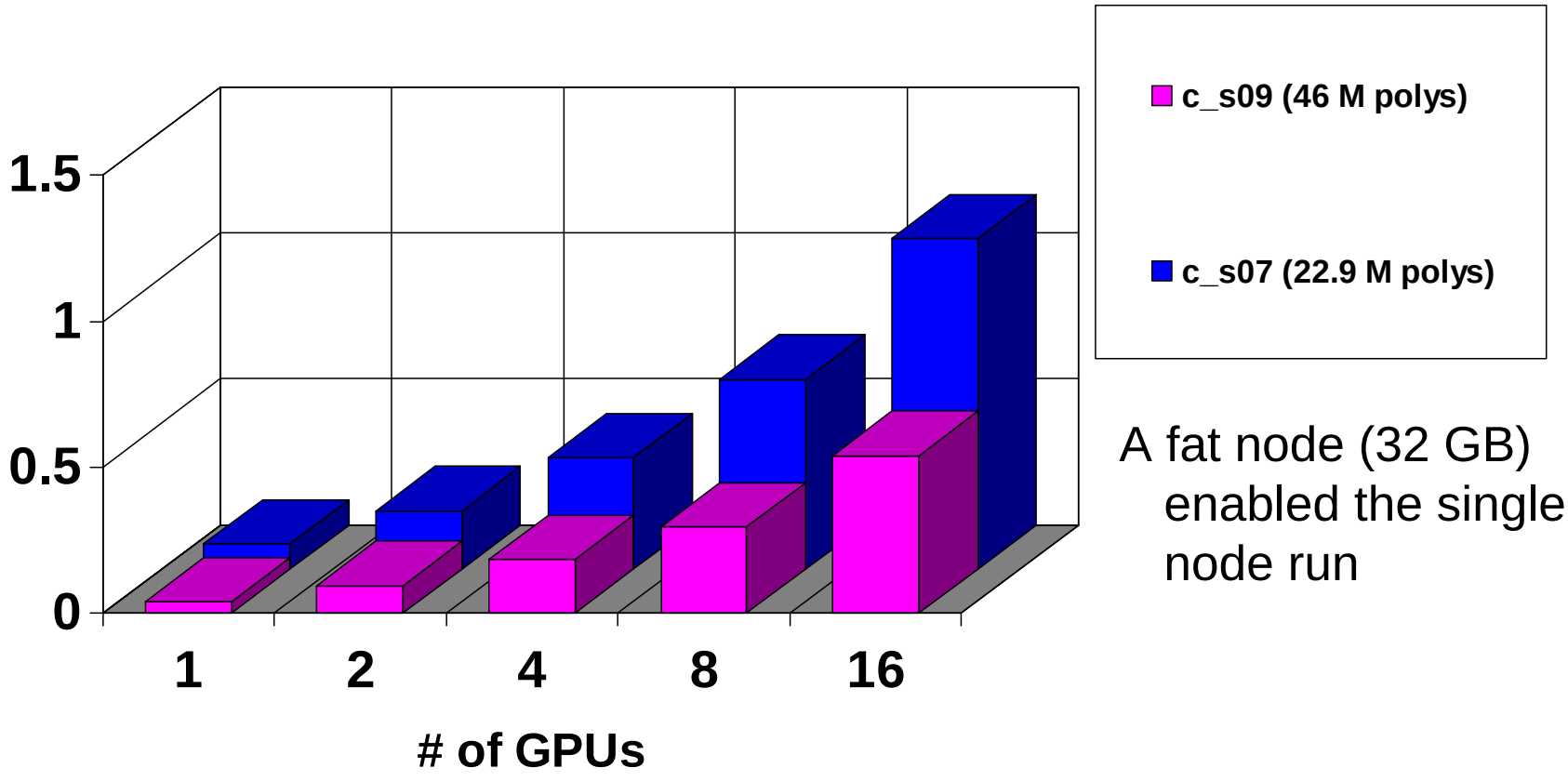
- sort-last rendering, without pixel sub-sampling and without Squirt compression
- Immediate mode rendering
- Ghost-cells were not provided



Rendering times (frames/s)



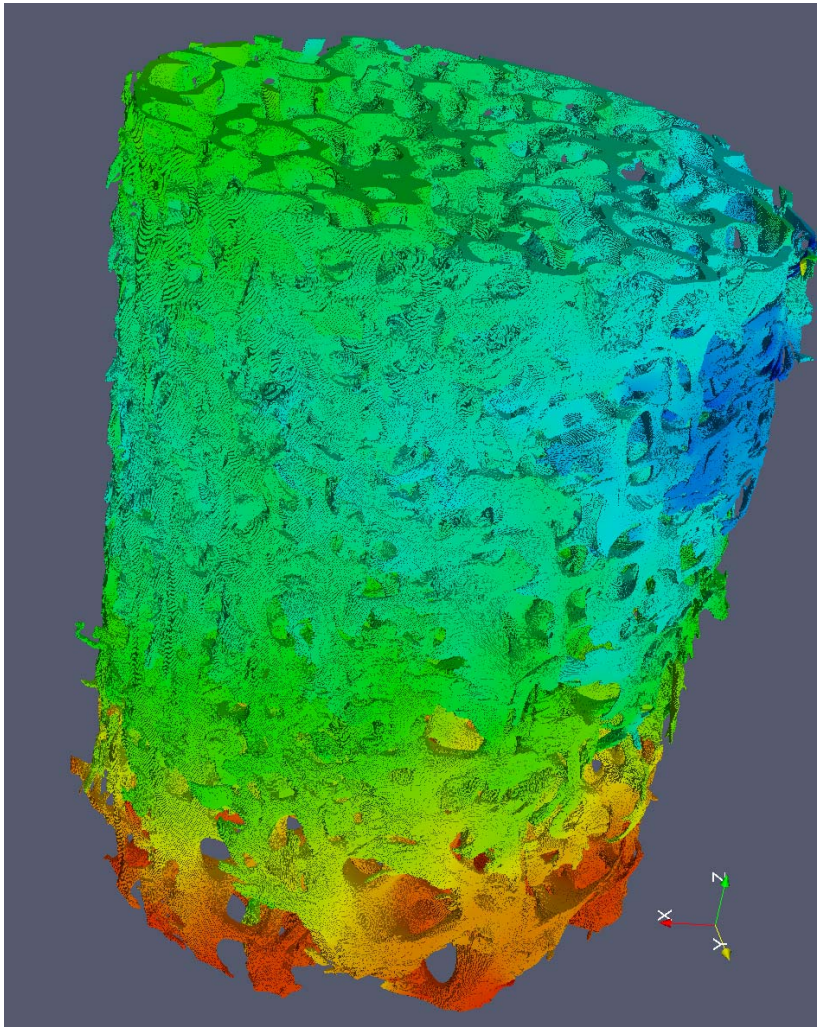
Rendering times (frame/s)



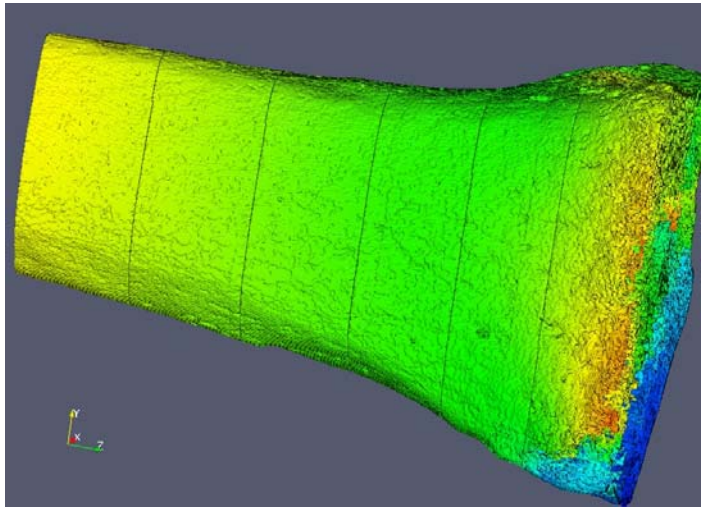
A fat node (32 GB) enabled the single-node run

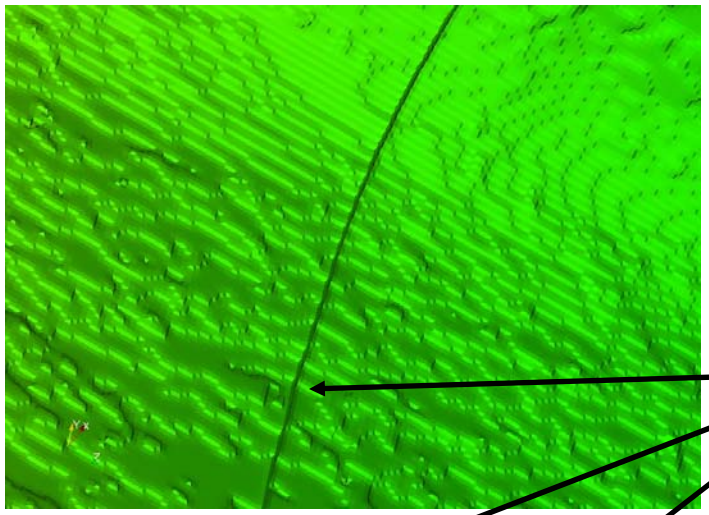
Benchmark 3

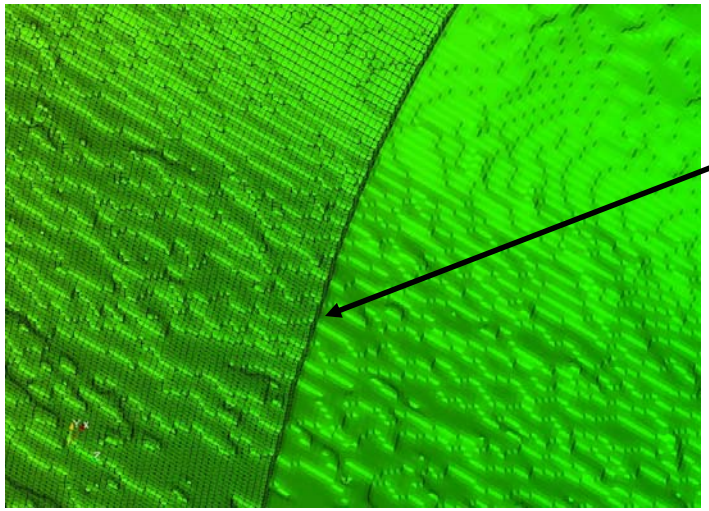
- 409,387,412 hexahedra and 447,294,209 nodes with nodal displacements distributed among 16 CPUs
- re-indexing of node numbers
- *on-the-fly* 64-bit to 32-bit conversion of HDF5 data
- *interactive* visualization

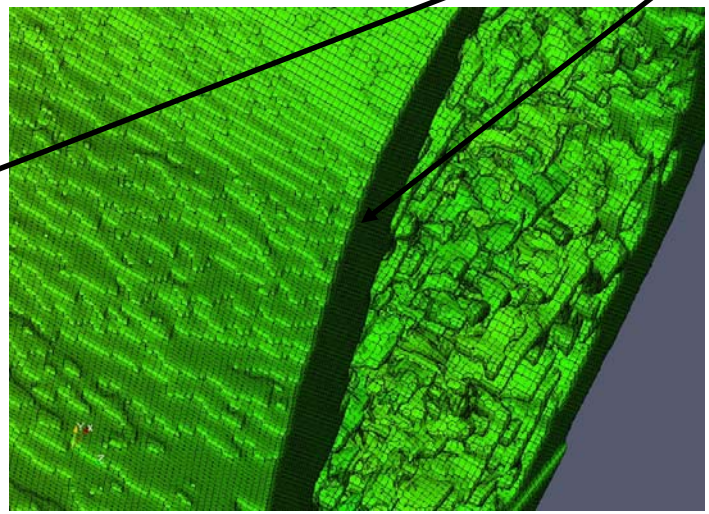


Artifacts with the simple-minded decomposition









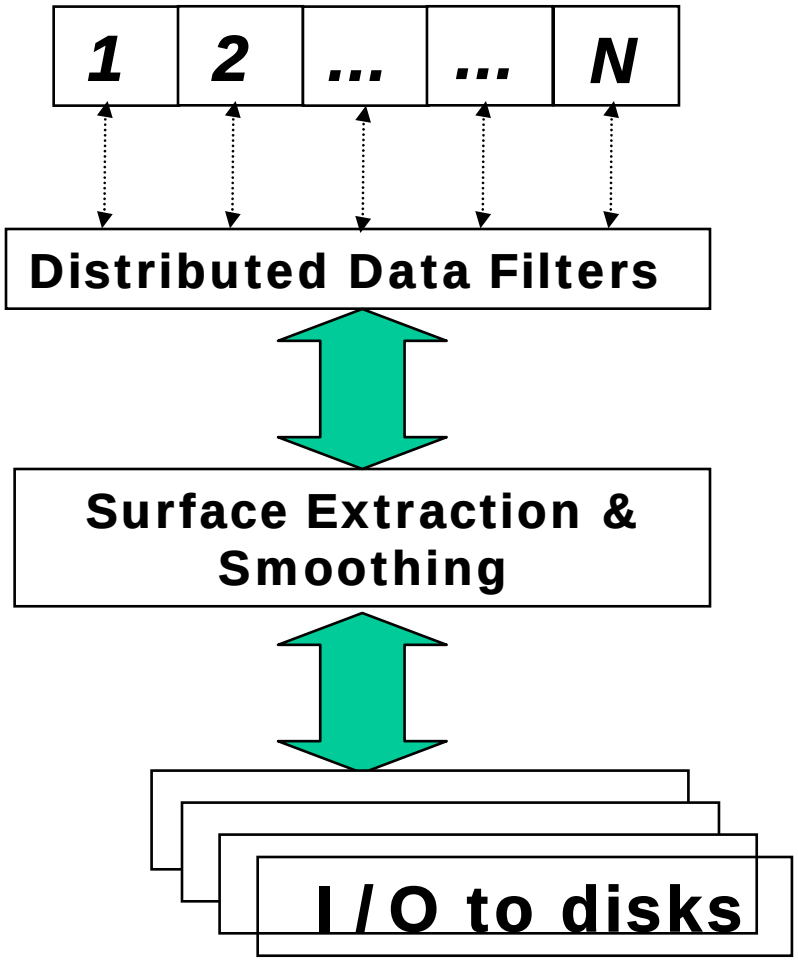
Problem:
Visual artifacts are caused by a non-physical boundary

It can be resolved by creating ghost-cells overlapping the boundaries

ParaView's parallel Kd-Tree partitioning will accomplish that.

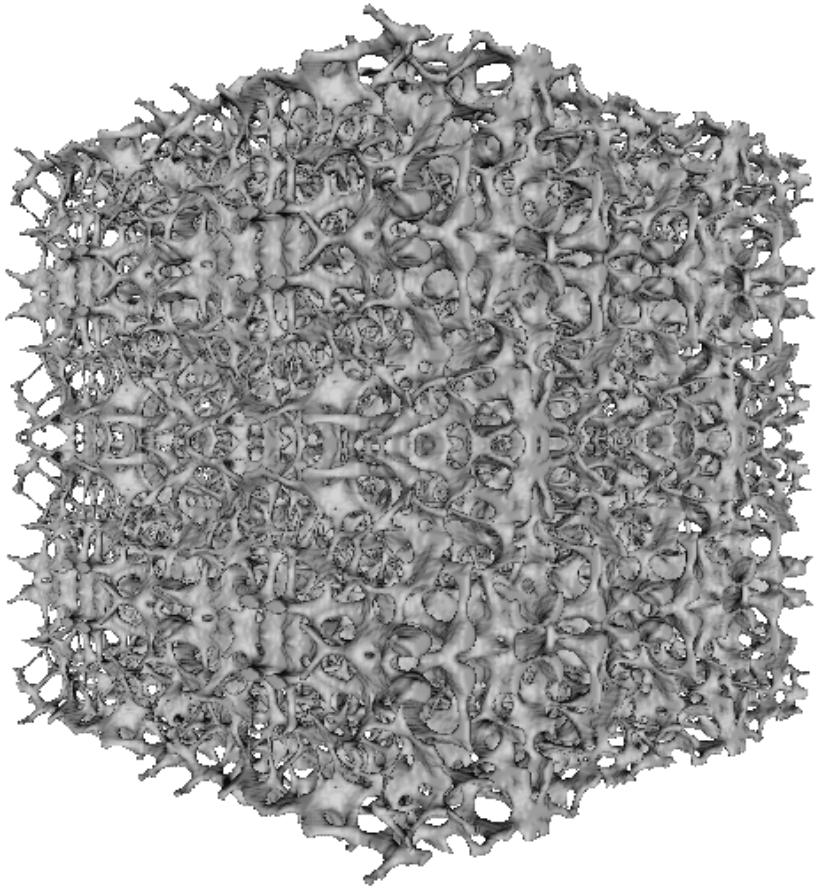
Offload the pre-processing

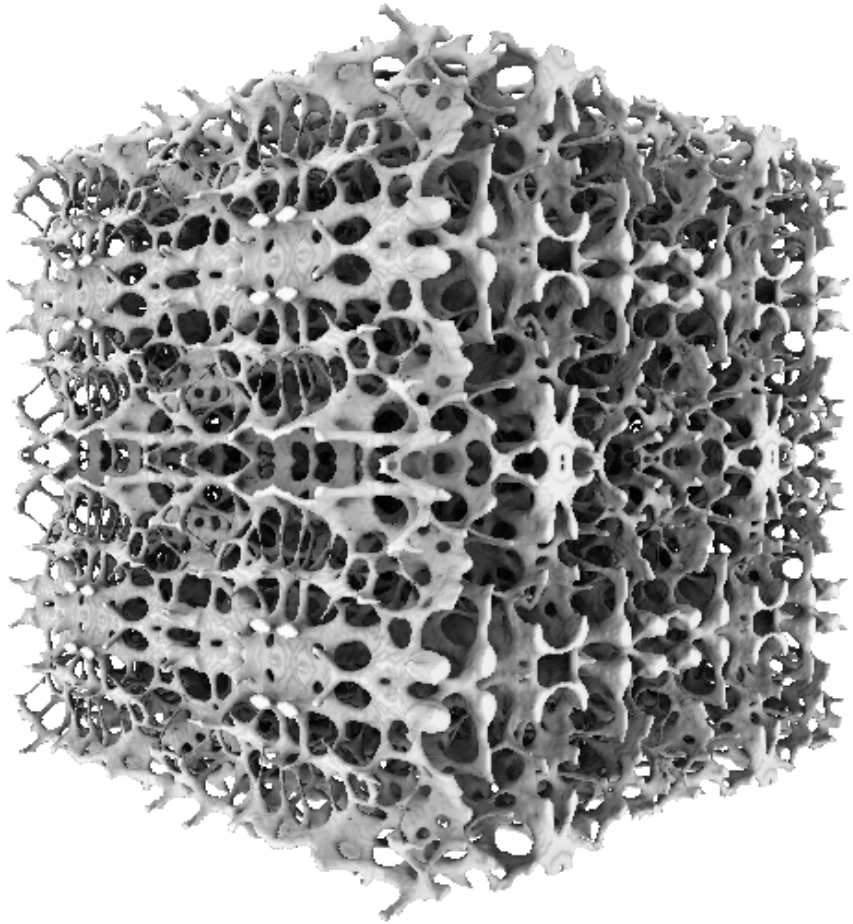
- Run the distributed data filter (D3) on a large cluster without graphics.
- VTK can write the data with ghost-cells in a partitioned manner.
- Pre-computing the data partitioning can be done on N processors (generating N pieces), and read back on M graphics engines.
  - N < M (a refined data partitioning can be done)
  - N > M (each proc handles several pieces)



Ambient Occlusion

Bone surface: ~8M triangles/4M vertices; 256 samples; 512x512 depth maps.
AO pre-computation time: 240s on one node; speed scales linearly with the number of nodes.





Ambient Occlusion Algorithm

For each vertex in a 3D model compute a visibility value in the range [0.0,1.0]:

1. Compute bounding sphere enclosing 3D model
2. Compute a number of random points on the sphere surface
3. Initialize vertex visibility array to zero
4. Move the view point to each point and render the scene pointing the camera towards the sphere center; at each rendering:
    1. retrieve depth map
    2. Compare depth component of each transformed vertex to value in depth map; if vertex[z] < depth_buffer[z] increment vertex visibility counter
5. Divide visibility values by maximum visibility value

Ambient Occlusion

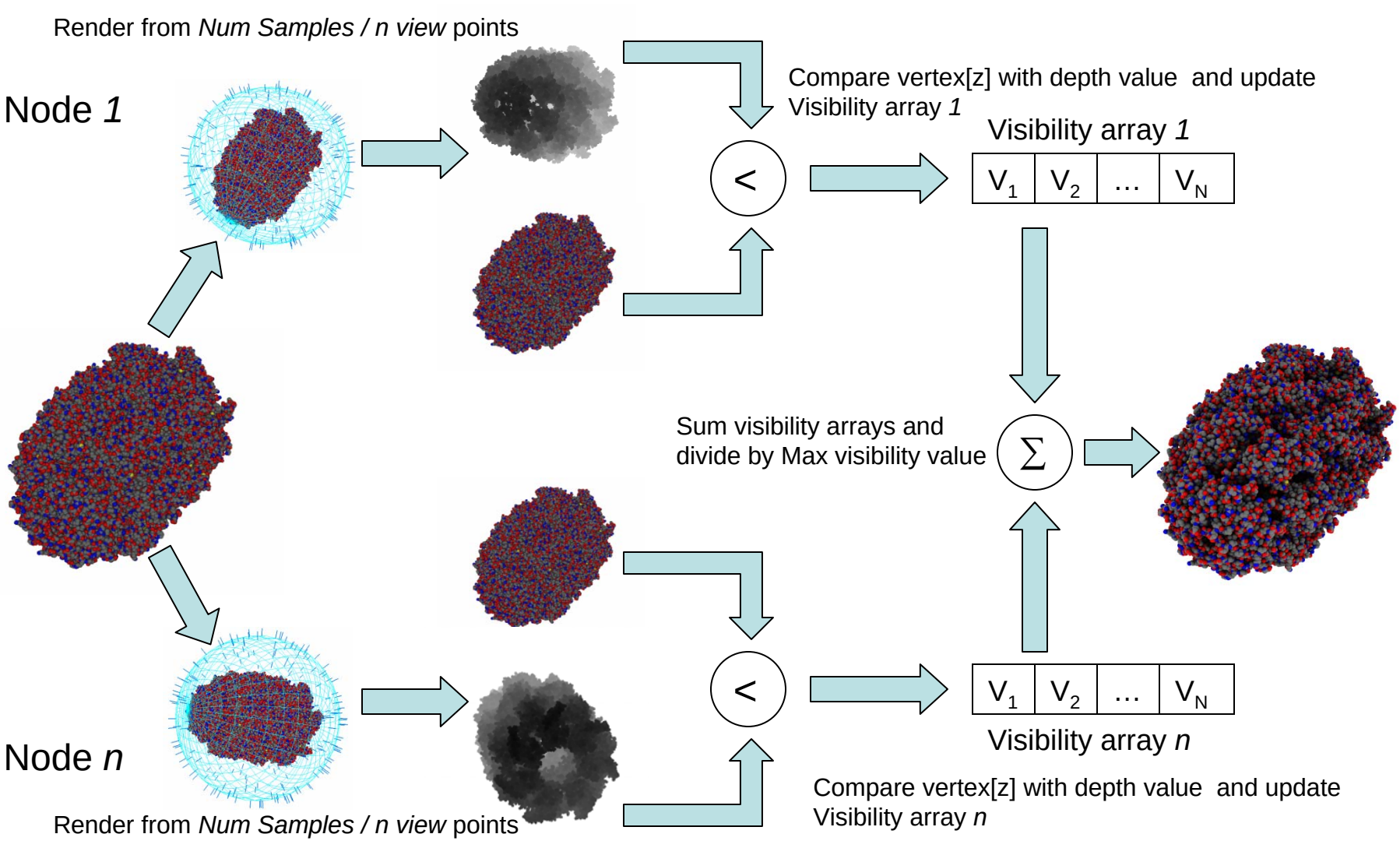
- Use the per-vertex visibility value during shading e.g.:
  - Vertex color = Material Color * Vertex Visibility
- Visibility values can be stored as per-vertex 1D texture coordinates associated with a 1D luminance texture.

  Texture coordinate = visibility value:

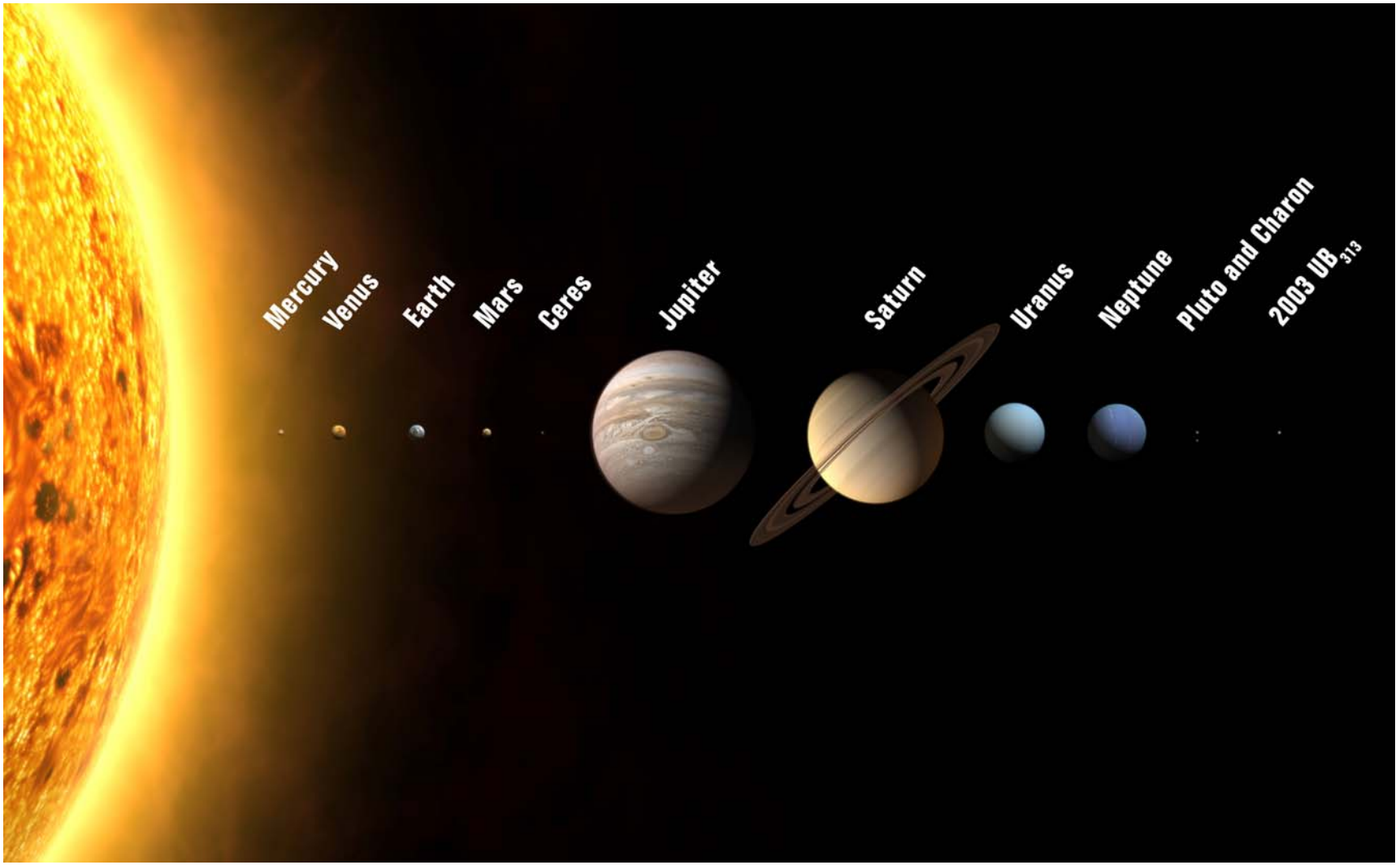
0 1

- Algorithm can be parallelized:
  - Send whole 3D model to different nodes
  - Each node performs Num. samples / Num. nodes *rendering/ depth-comparison/visibility-update* loops and updates a separate vertex visibility array
  - The visibility arrays from each node are summed together and divided by maximum visibility value

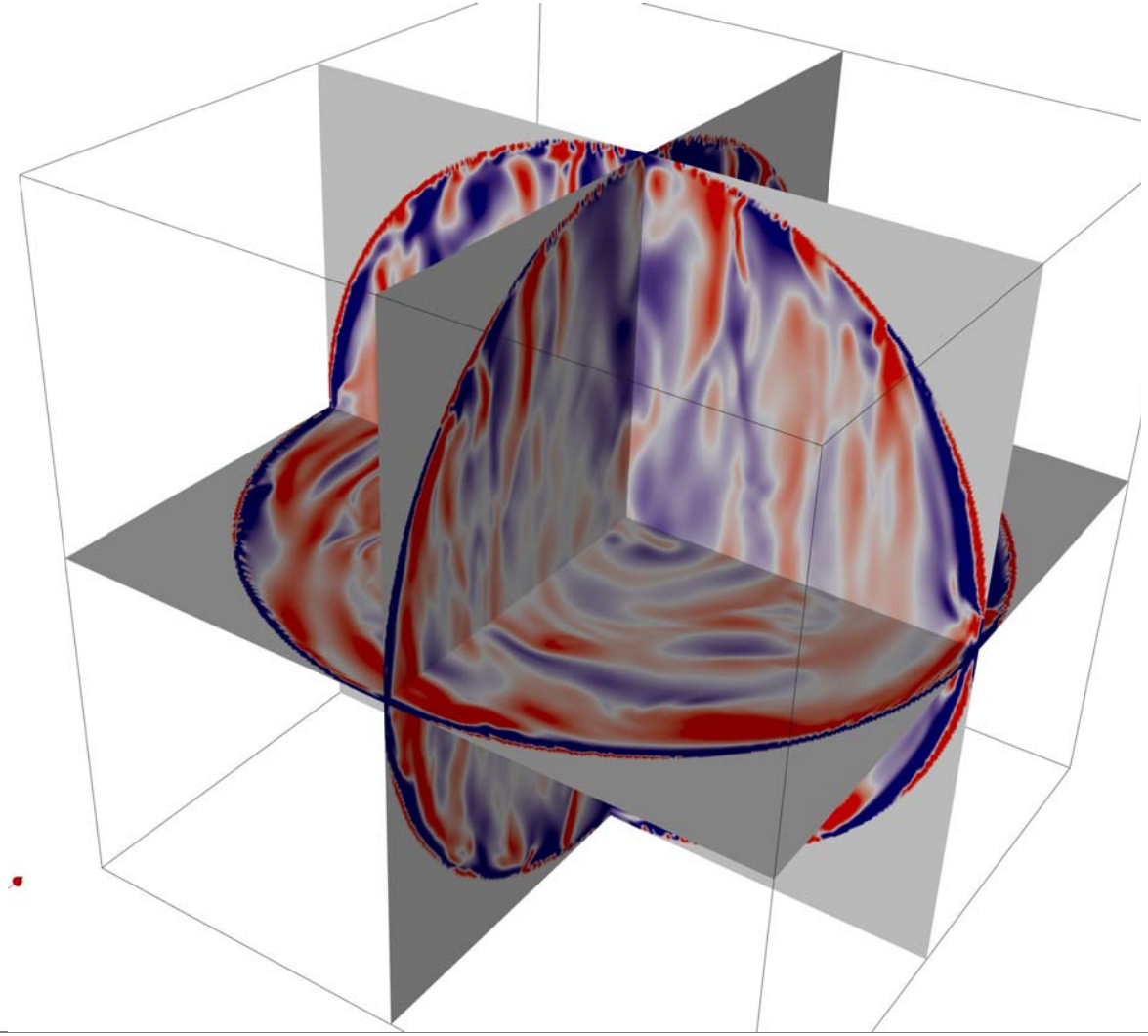
Ambient Occlusion



Geo-physics data visualization

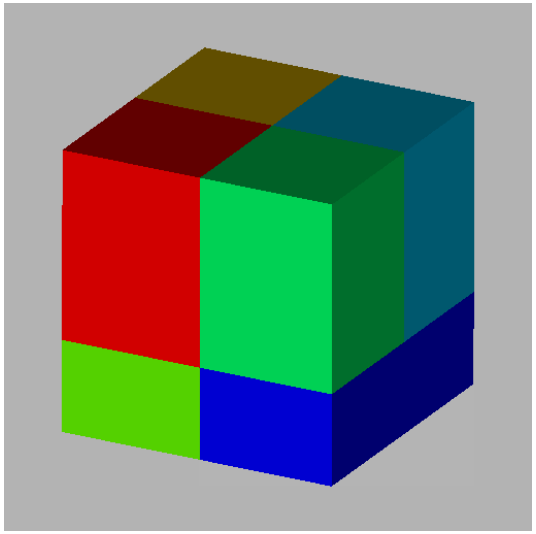


Large structured grids (400^3) and (800^3)

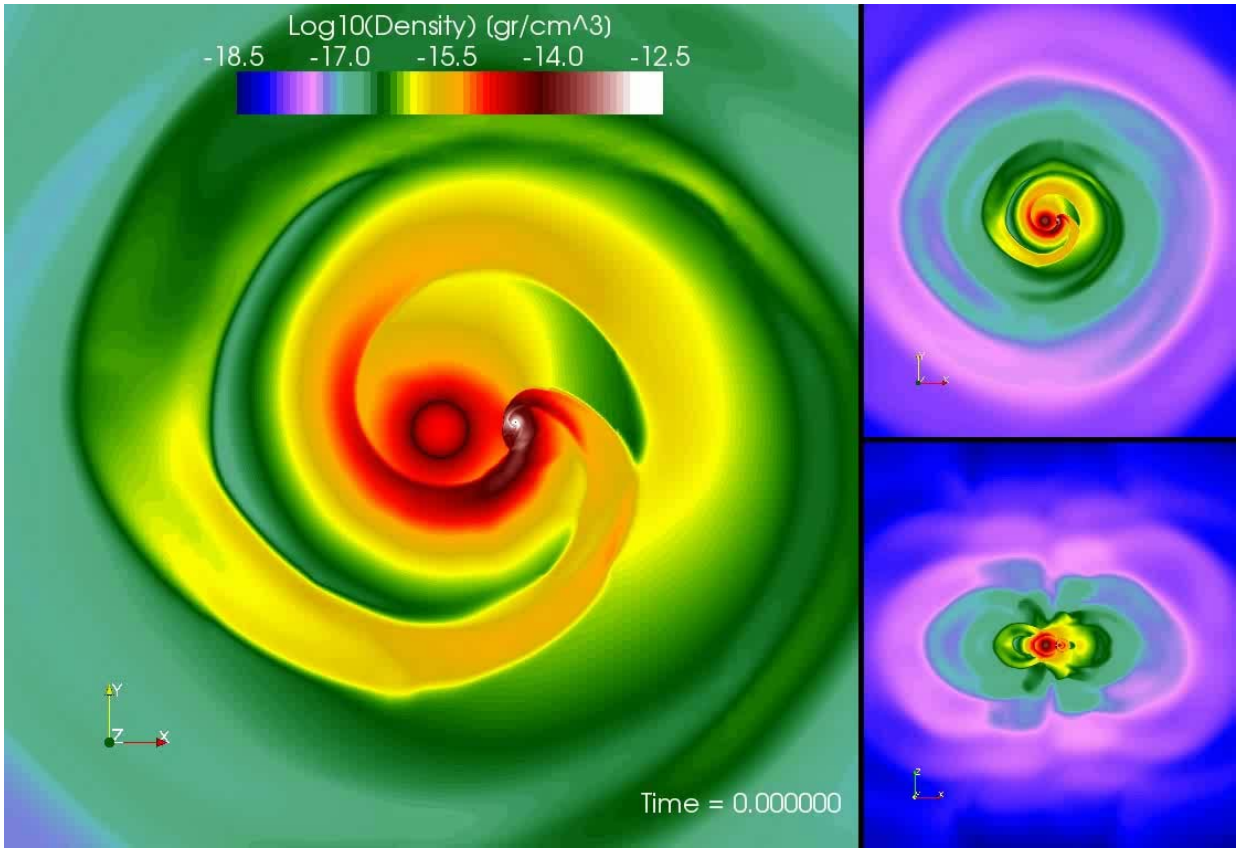


The pipeline decides by itself how to split the volume

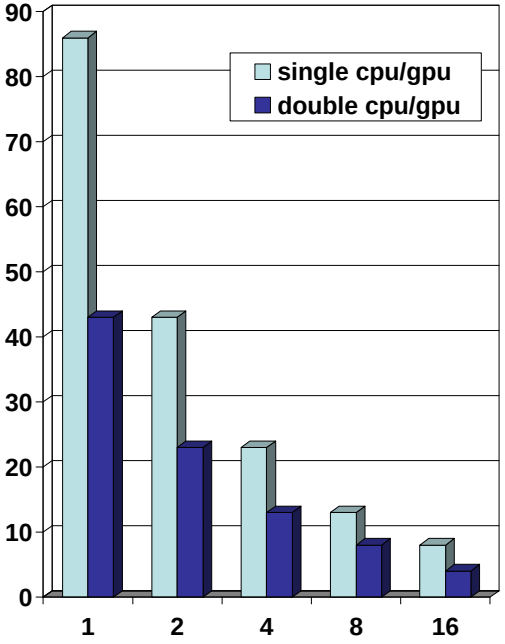
Handle any request, including ghost-cells



Astrophysics; 233 AMR grids, 9 levels



All processors manage grids from all levels



Summary of requirements

- Efficient splitting of cells and nodes, or grid slabs is a must. The data format should foster efficient data access. (cf. HDF5)
- Ghost-cells can be added later
- We focus on static data distribution for load balancing to get the best overall performance (with focus on time)

- Visibility of very large opaque surfaces can be enhanced with Ambient Occlusion. But, …
  - The simple implementation of the parallel evaluation needs the full geometry.

The datasets are courtesy of Prof. Harry van Lenthe, Prof. Peter Arbenz, Martha Evonuk and Rolf Walder, ETH Zürich

Distributed rendering

- Many of our applications use opaque geometry where sort-last rendering works really well (VTK)
- Our parallel volume rendering experience is based on the Equalizer Graphics toolkit (eVolve)
- HP has opened their Pixel Compositing API recently offered to the open-source community

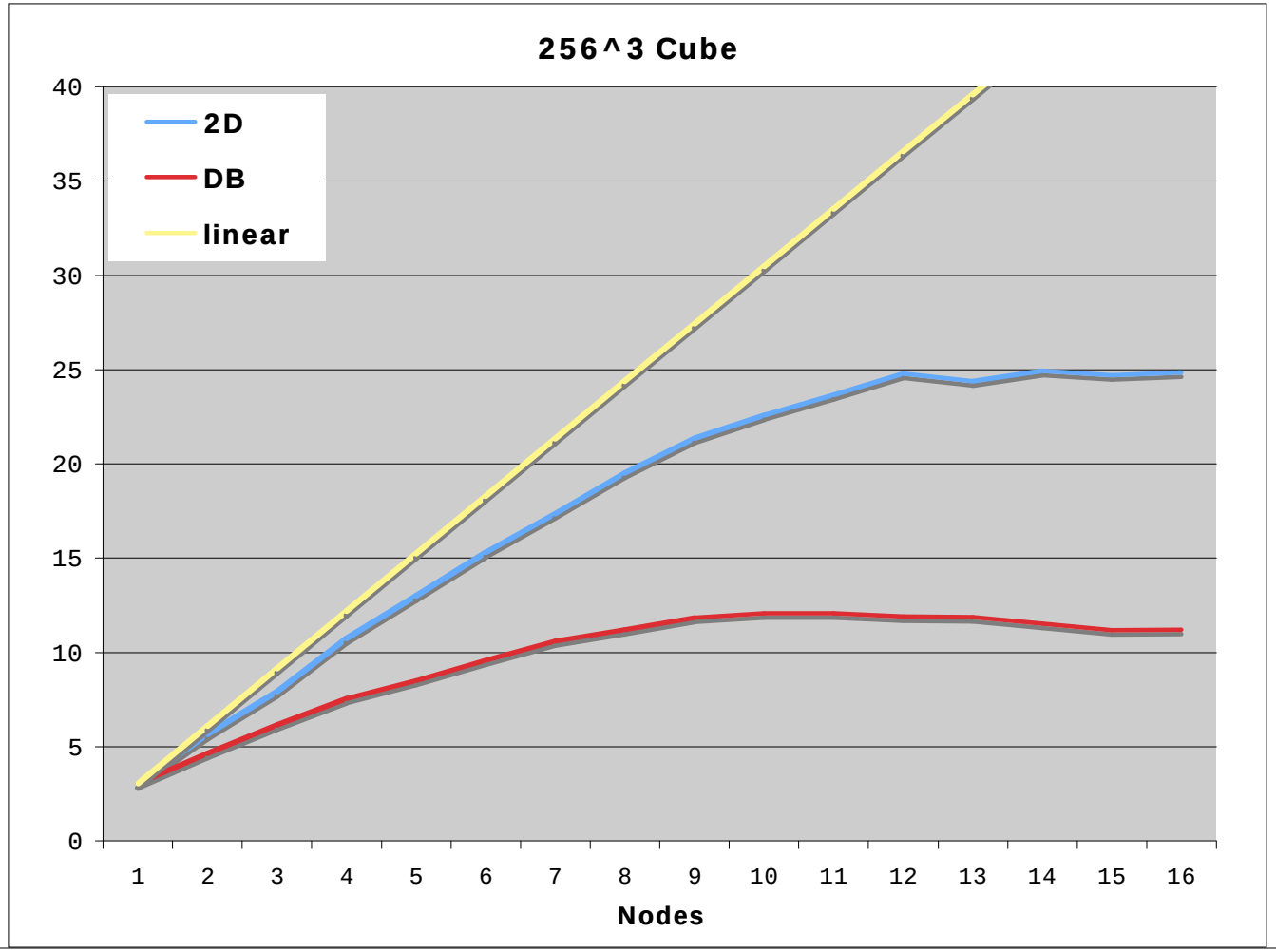
eVolve benchmarking

- Performance tests on 1 to 16 nodes
- Only one GPU per node was used
- 1280x1024 pixels full screen resolution
- 3D texture based volume rendering algorithm

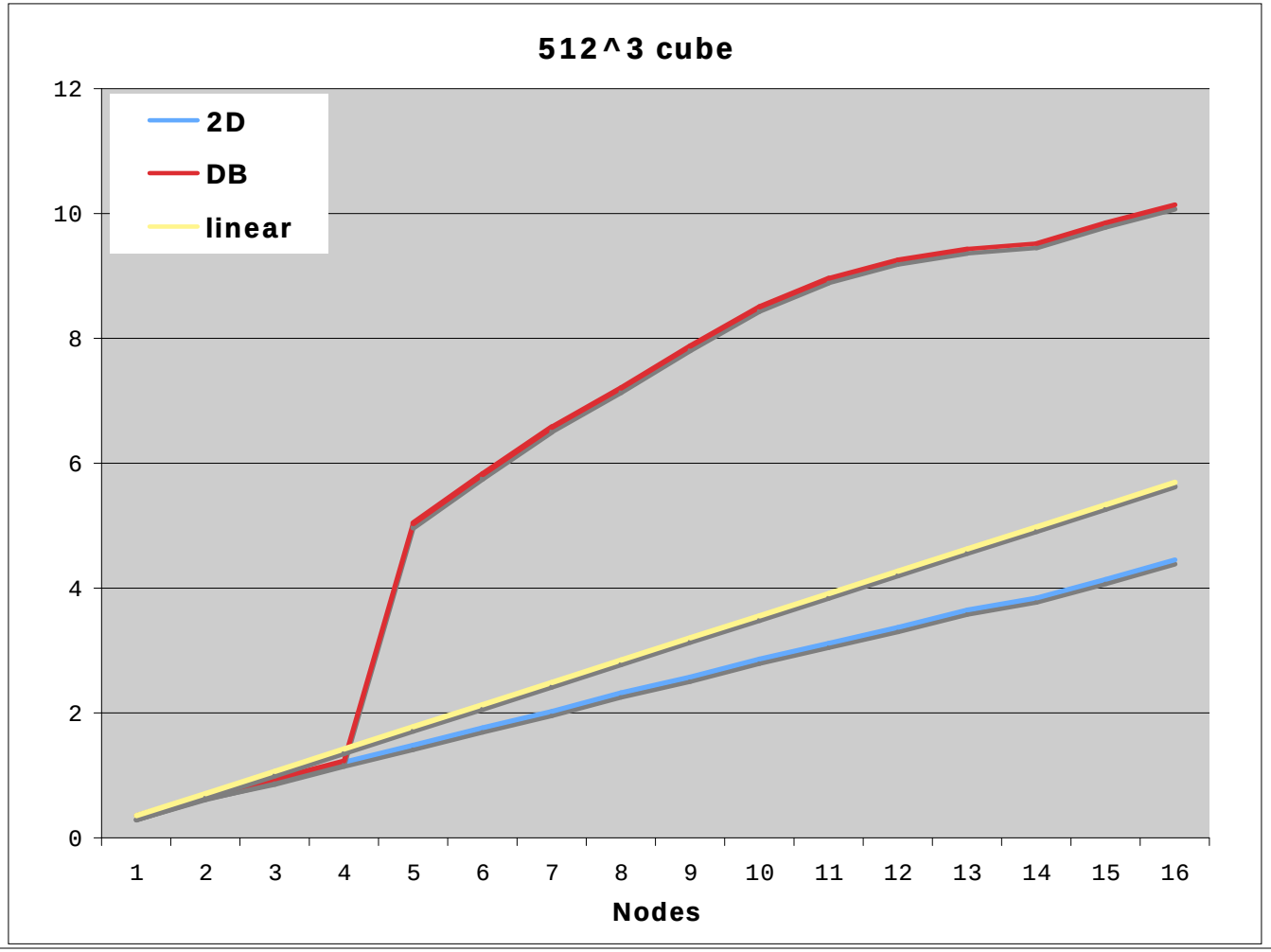
Composition Modes

- 2D Decomposition mode
  - Each node renders whole model but only for one region of the screen
  - One node receives all finished parts and composites them into a complete frame
- DB Decomposition
  - Each node renders part of a model for the whole screen
  - Each node receives specific part of a frame and performs compositing for that part
  - One node gathers all composed parts and composes final frame

Performance 256^3



Performance 512^3



Parallel Volume Rendering summary

- Currently, for smaller models, performance is bounded by hardware limitation of the compositing stage
  - In particular drawPixel performance is poor
- For larger models it is possible to achieve even more than linear speed up
- Equalizer can provide distributed rendering on interactive speeds for larger volumes

- Thanks to Renato Pajarola, Stefan Eilemann and Maxim Makhinya for sharing these results from the CSCS cluster benchmark

CSCS

Swiss National Supercomputing Centre

The Parallel Compositing Library

Steve Greenwood
HP Scalable Visualization Team

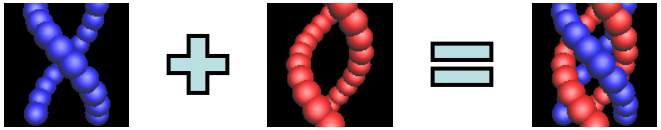
5 slides borrowed without permission
(available on the Web)

HP's Parallel Compositing Library?

- A Code Library
  - Greatly simplifies the task of distributing the rendering of graphics across a set of graphics cards
  - Comes in a few flavors
    - A sample implementation that runs on many platforms
    - A tuned implementation released as a part of HP's SVA 2.0
  - In a phrase: The Library is an MPI for Visualization

How does it work?

- The Library is designed to facilitate sort-last parallel rendering
  - Sort-last parallel rendering
    - Produces an image in parts using multiple graphics cards installed on 1 or more nodes
    - Enables
      - Rendering very large datasets with reasonable frame rates
      - Creating very large images
  - When an image is created in parts, the separate image parts must then be combined to produce the final image
    - the combining of the pixels from multiple images is called compositing
  - The Library supports compositing on a network of nodes
    - The application present image parts to the distributed Library
      - The image parts are called framelets
    - The Library combines the parts using one or more pixel operators
    - The Library delivers the composited image back to one or more parts of the application for display
      - The composited image parts are called outputs

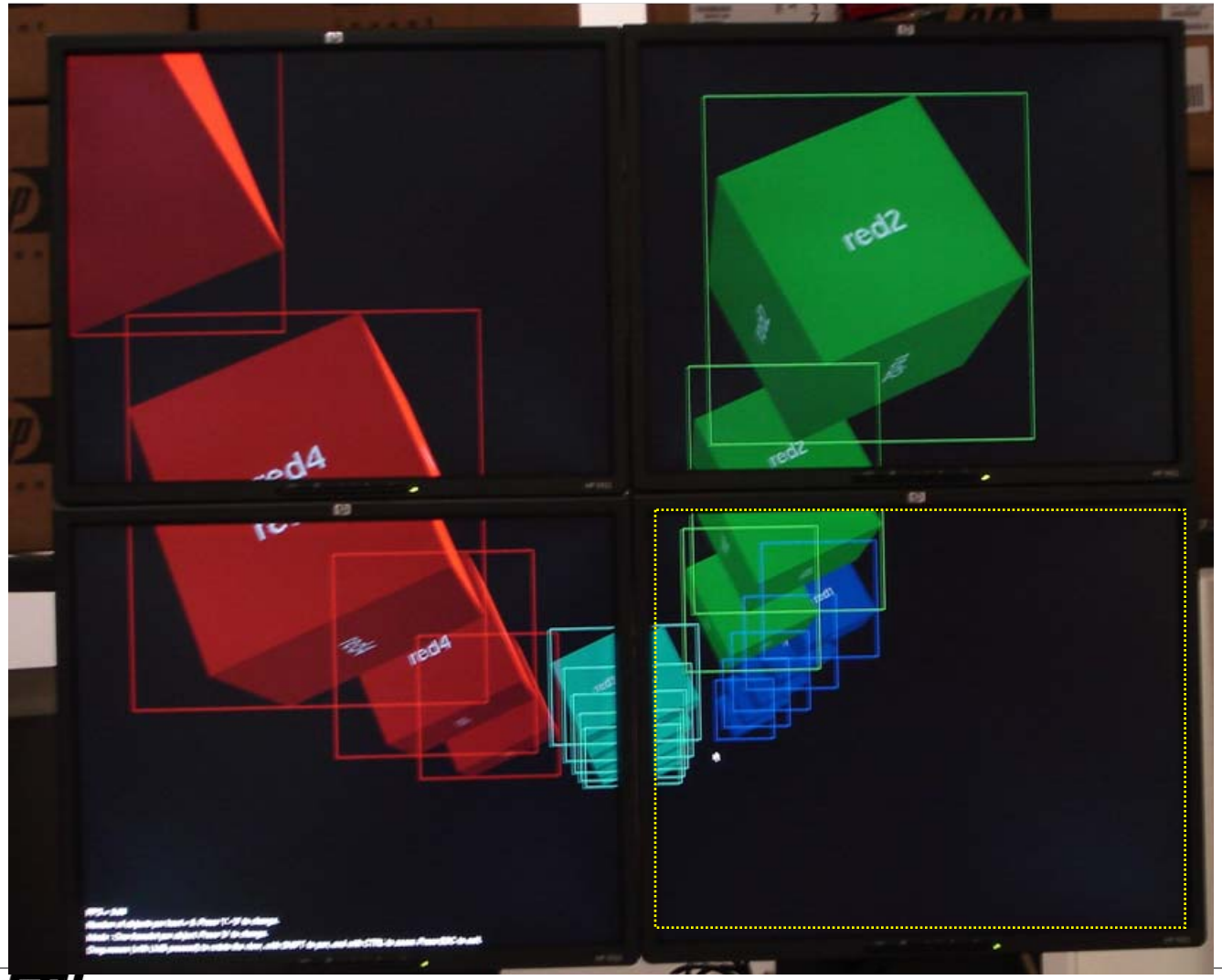


framelet #1 framelet #2 output

What can the Library do?

- Process much more graphics than an individual graphics card
  - A high-end graphics card: 225 Million Triangles/second
  - The library can process many times this number of triangles
    - It does this by letting you distribute the load over multiple cards
- Create images larger than any individual graphic card
  - A high-end graphics card is still limited to
    - 2 3840 x 2400 pixels displays
    - 4K x 4K pixels when rendering 3D graphics
  - The library can produce images many times this size
    - It does this by letting you use multiple cards to produce the images

Photo of Sample Program



Each box is drawn separately and presented to the library

The final image is displayed by 4 different cards

The library takes the images given to it, composites them, and directs the resulting image to the proper card for display

Conclusion

- Customized & optimized solutions in visualization
  - Data decomposition and partitioning for parallel visualization
  - Resource management for multi-user, multi-site, collaborative access to graphics hardware
- Robust support for Parallel execution & rendering
  - HP SVA cluster
  - VTK and EqualizerGraphics are our best apps
- Resource allocation on a *first-come first-serve* basis is being augmented with a reservation scheme
- Request for allocation of computing resources in 2008 now include 1000s of hours of parallel visualization