Advanced ParaView: Manta (CPU raytracing) and Adaptive (Multi-resolution Streaming)

Jon Woodring – Los Alamos National Laboratory

Jim Ahrens (LANL), Dave DeMarle (Kitware), Li-Ta Lo (LANL), John Patchett (LANL)

Executive Summary

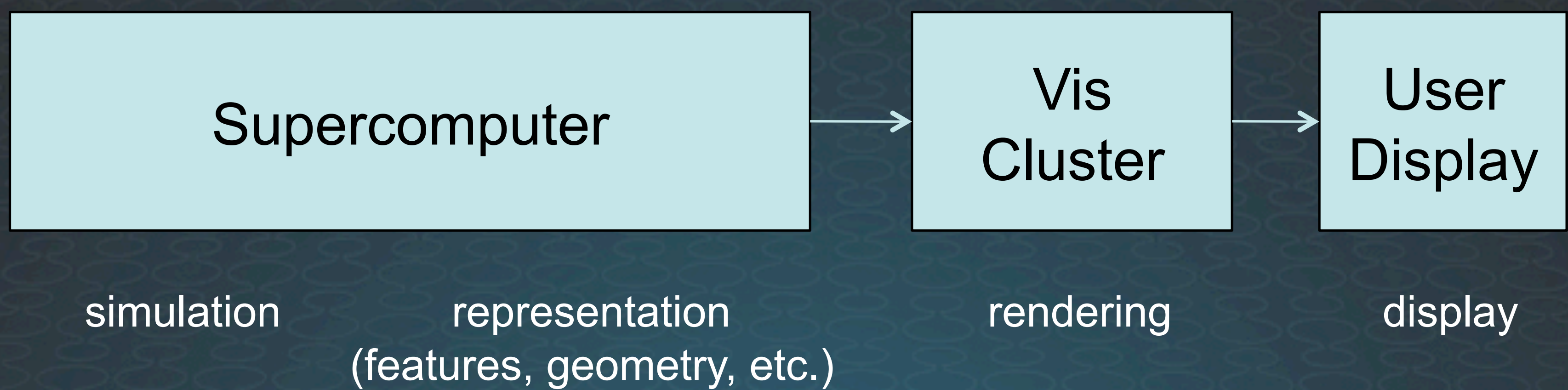
- University of Utah's Manta raytracer as a rendering plugin
  - Using CPU raytracing as an alternative to hardware rendering (Manta is fast!)
- Adaptive ParaView application
  - Multi-resolution streaming for large scale data distance visualization
- Available in CVS ParaView!
  - Download the head and try it out
  - Detailed instructions to follow

General Outline

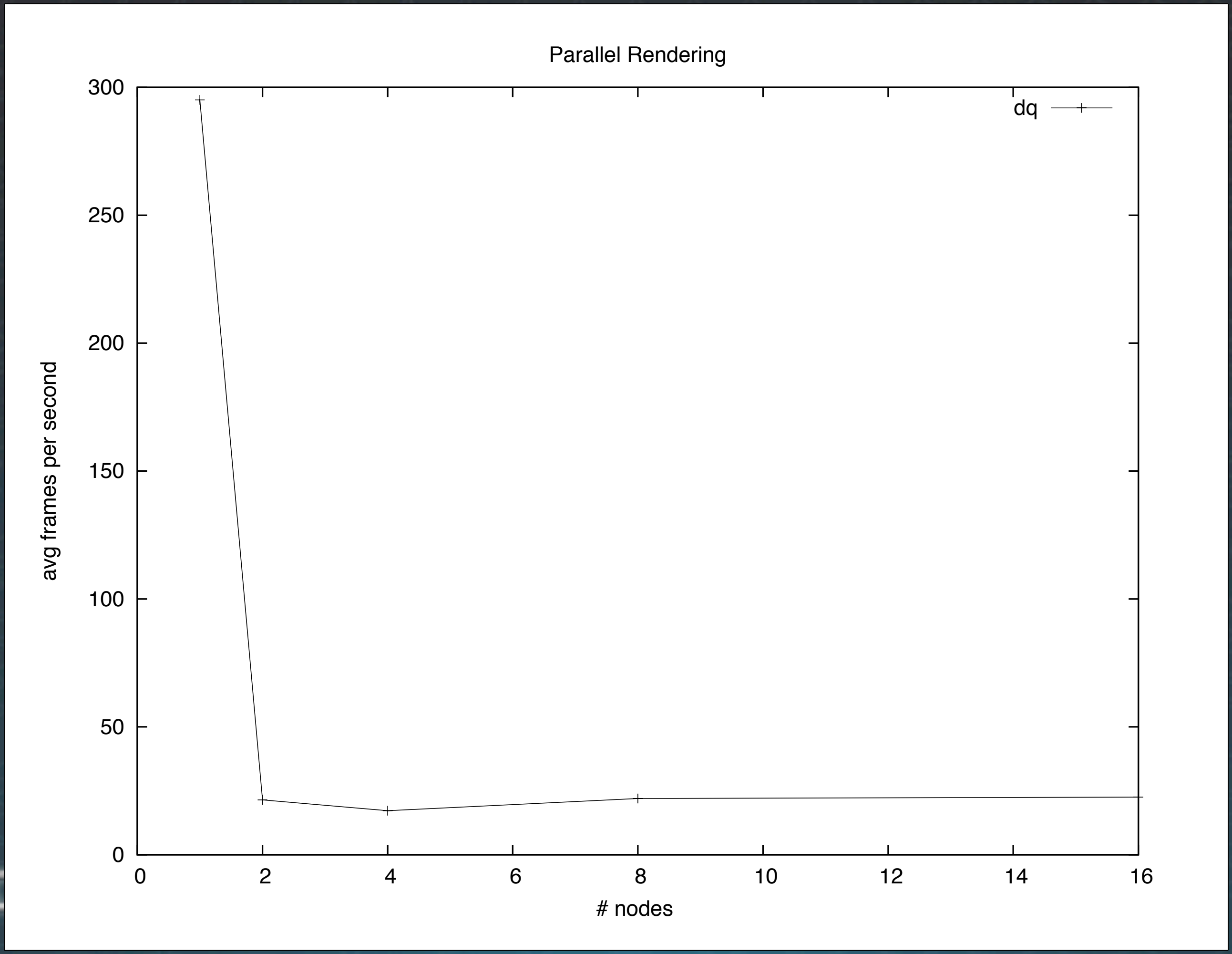
- Manta plugin
  - A live demo
  - Why we did it
  - How we did it
  - How you can do it, too
  - Questions
- Adaptive ParaView application
  - Same outline as above

Manta ray tracing Demo

Current LANL Production Vis



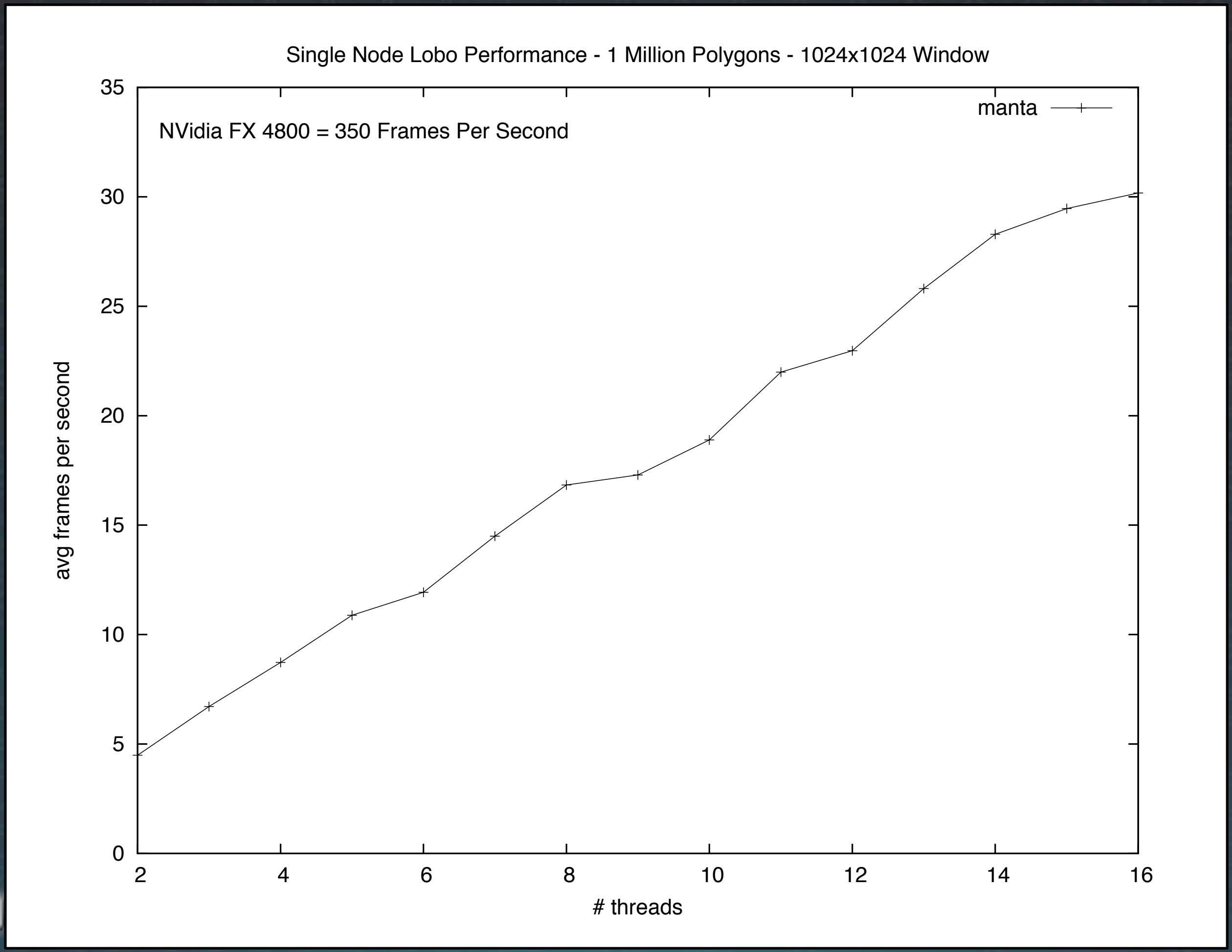
Parallel Rendering –
The Bottleneck is Compositing



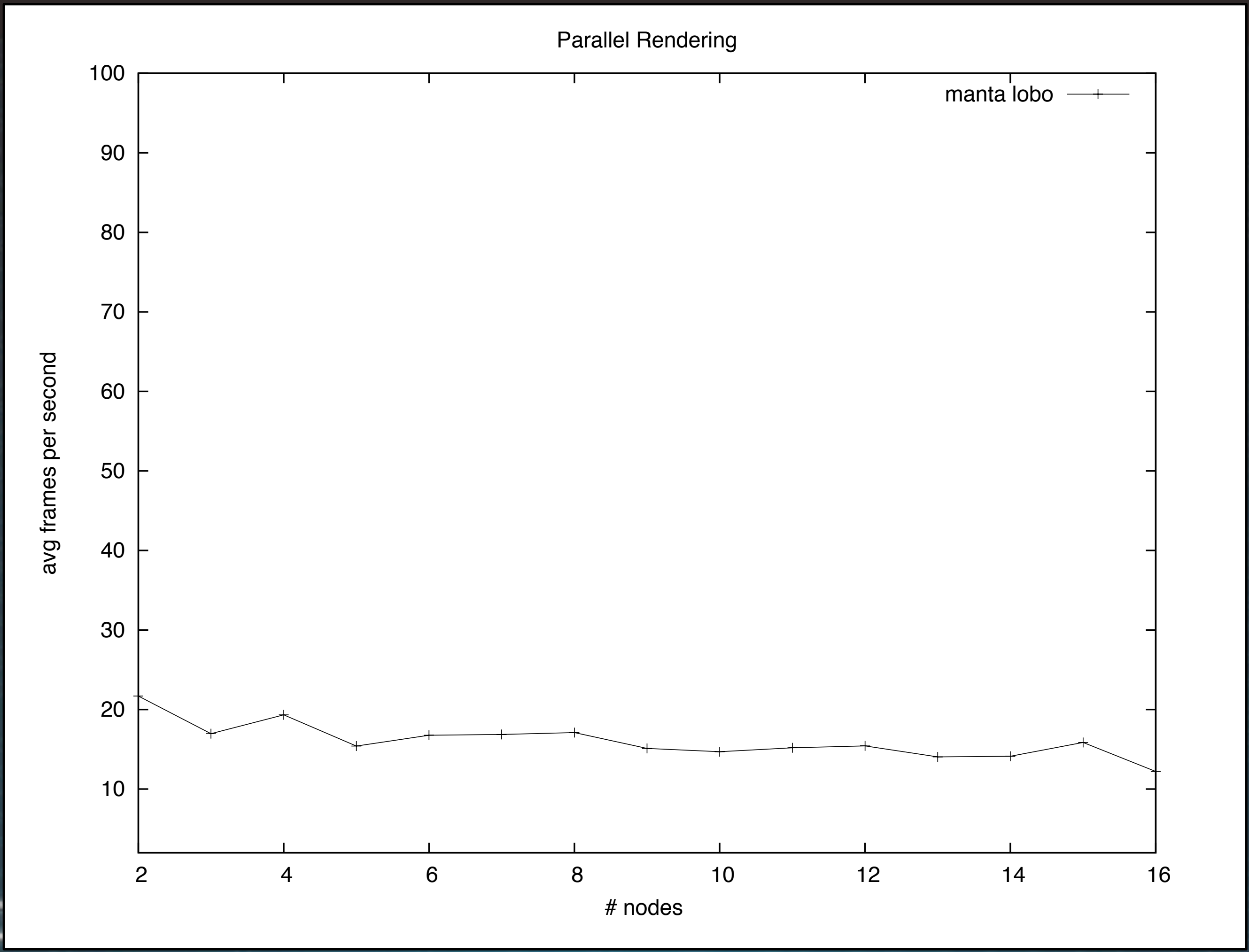
Target Frame Rate

- Maximum parallel compositing frame rate
  - 20-30 frames per second
- To meet this target frame rate
  - GPU rendering
    - 300-350 frames per second – overkill
  - CPU rendering?
    - Mesa 3D – pretty slow, doesn't meet target rate – about 4-5 fps with 16 cores on a node
    - Manta raytracer – much better than Mesa

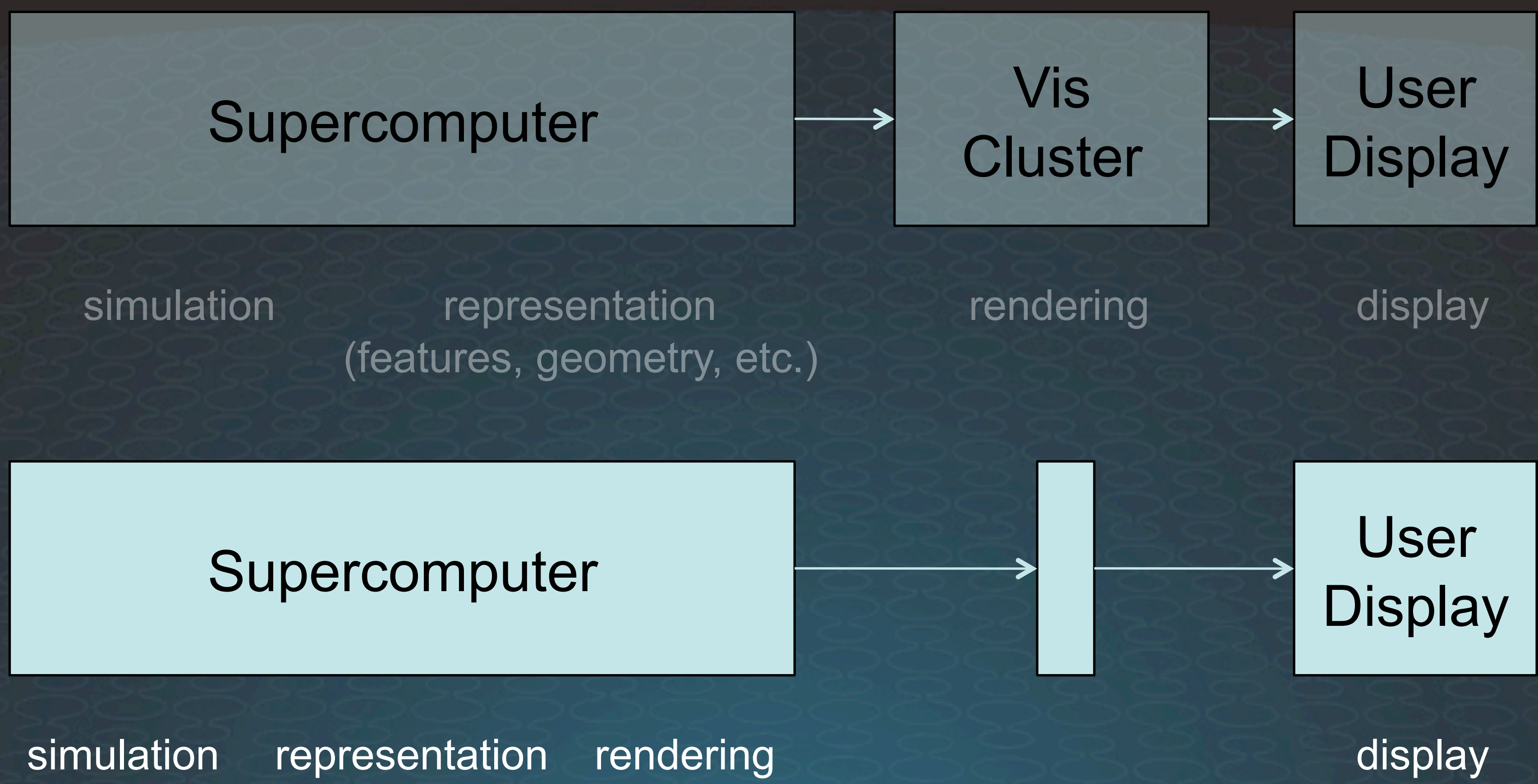
Manta raytracer 1 million polygons – Similar performance with 2, 4, & 8 mil



Manta (16 threads per node) with Parallel Compositing



Back to the CPU (supercomputer)



Benefits

- Fewer specialized visualization requirements
  - Visualization is a supercomputing application
  - Fewer specialized hardware
  - One HPC resource to manage
- Data is already there – no need to move it
- Manta raytracer
  - High quality images – shadows, multi-sampling, reflection, refraction, etc.
  - Gets faster the more cores you throw at it
- Faster and cheaper rendering software development (heterogeneous computing development is expensive)

Drawbacks

- Interactive Queue?
  - Supercomputing queues are batch
  - May be an uphill battle to get good (large node counts and short waiting) interactive queues for vis
- No more specialized hardware cluster
  - Visualization (rendering) isn't a special application snowflake anymore
- (GP)GPU is riding high
  - Up front cost of going back to CPU rendering
- Frame rate is not high enough for stereo/RAVE/CAVE

Vis09

Manta View Plugin Implementation

- Manta View is a plugin that implements a 3D view
  - Wraps Manta, provides data, makes render requests
- Override the vtkRenderWindow object factory mechanism
  - When VTK asks for a render window, a vtkMantaRenderWindow is returned
  - This override can be seen in action with some of the test VTK applications in the build bin directory, like marching – a Manta View will be used
- Disable IceT compositing
  - IceT uses OpenGL concrete classes, which bypasses the object factory override

Manta View Plugin Implementation

- Use standard VTK depth compositing classes
  - Similar to binary swap
- Z (depth) channel added to Manta
  - Used for parallel depth composition
  - Code checked into Manta
- vtkMantaPolyDataMapper
  - Copies triangles to Manta (like a OpenGL display list)
  - Generates tubes and spheres for lines and points

How to Run Manta ParaView

- Download CVS ParaView (make sure you have Cmake, Qt 4.5+)
- Download Manta
- Build Manta (it uses Cmake to build, too)
  - MANTA_USE_X11 OFF
- Build ParaView
  - PARAVIEW_BUILD_PLUGIN_Manta ON
  - MANTA_BUILD <absolute Manta build path>
  - MANTA_SOURCE <absolute Manta source path>
  - ParaView_DIR <current absolute PV build path>

How to Run Manta ParaView

- Start pvserver
- Start ParaView
  - Close the current view
  - Connect to your pvserver
  - Load MantaView plugin (libMantaView.so/.dylib/.dll) on client and server
  - Close the current view, again
  - Open a Manta view
  - Visualize some polygons!

Vis09

Questions?

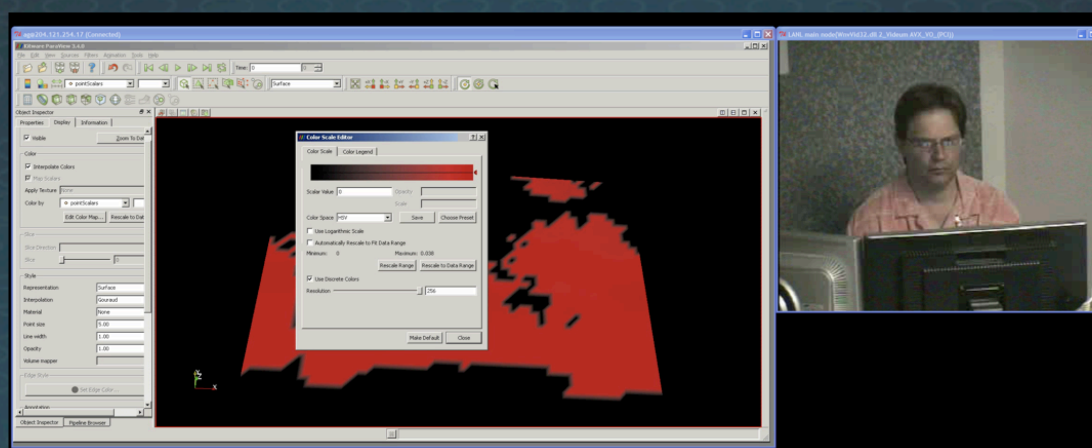
- Try it out on your supercomputer platform
  - More users = more demand = more likely to get better interactive queue support to support visualization on the platform in the future
  - User feedback
- Print/press quality visualizations out of the box
- Future volume support? (raycasting)
- Fast scan conversion – a better Mesa 3D
- Work in progress
- Next up: Adaptive ParaView

Adaptive ParaView Demo

Vis09

Remote Data

- Mat Maltrud works at LANL (Los Alamos, NM) on the Climate team and runs climate simulations at ORNL (Oak Ridge, TN)
  - Mat is responsible for generating and analyzing the simulations



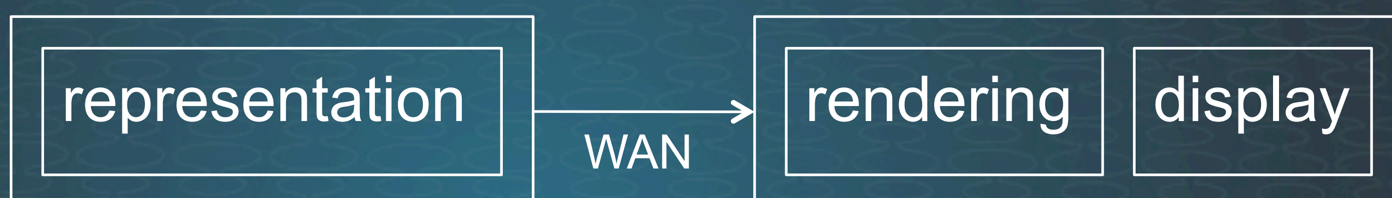
Remote LARGE Data

- Using 100 TeraFLOPs of Jaguar (ORNL)
  - 6 fields at 1.4GB each 20x a day
  - 3600 x 2400 x 42 floats
- Transfer to LANL would take > 74 hours (~3 days)
  - ~5 Mbps between LANL and ORNL
- Transferring all the data from ORNL to LANL will not work!
  - 250 TeraFLOPs
    - 12 fields
  - 1 PetaFLOP
    - 24 fields and 40x a day = 740 hours (~1 month)

Remote Visualization Approaches Available In ParaView

- Server side rendering
  - Run data server and render server on the supercomputer – send images

- Client side rendering
  - Run data server on the supercomputer – send geometry
  - Render client side



Client side rendering?

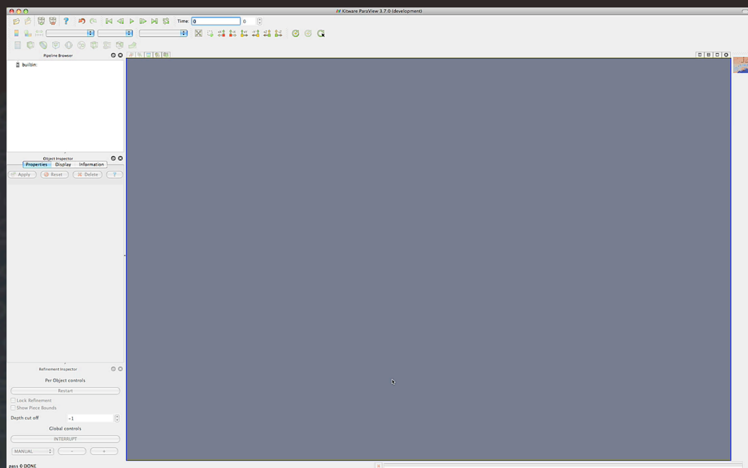
- Image-based distance vis: it works, but…
  - Completely server side based (dumb client)
  - Frame rate is network bandwidth limited
- Client side rendering?
  - Higher potential frame rate because of that nice client side GPU if the data can fit on it
  - Can render without needing the server (caching)
  - Science for science sake – explore both approaches
- Though… this is LARGE data – too big for the client, network, and display... Is it even practical?

Subset the Data to Fit Displays and Networks

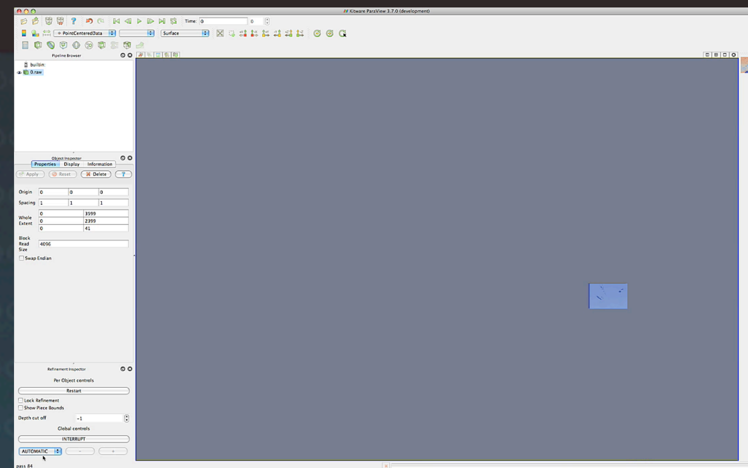
Prefix	Mega	Giga	Tera	Peta	Exa
10^n	10^6	10^9	10^{12}	10^{15}	10^{18}
Technology	Displays, networks, clients →		Data sizes and super-computing →		

Downscaling
Sampling
Subsetting

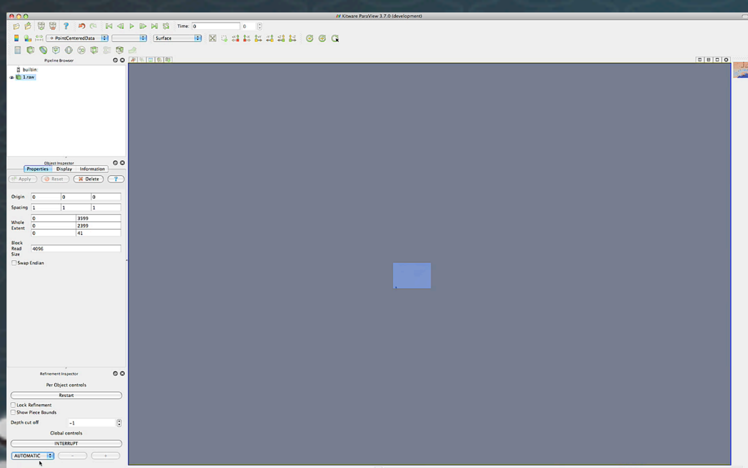
Streaming in ParaView



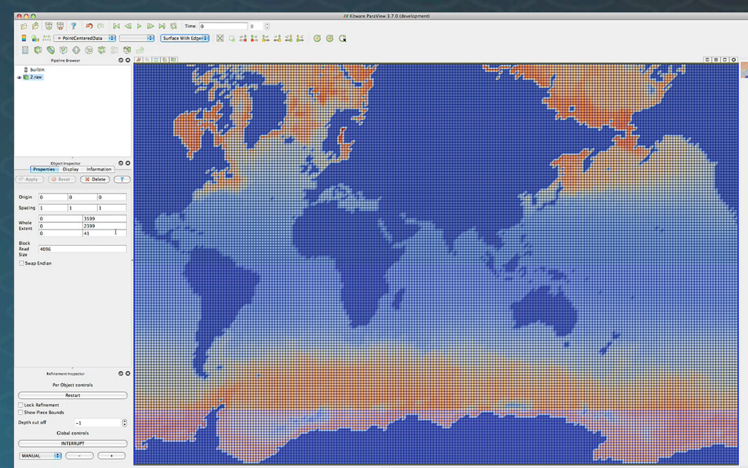
standard



streaming

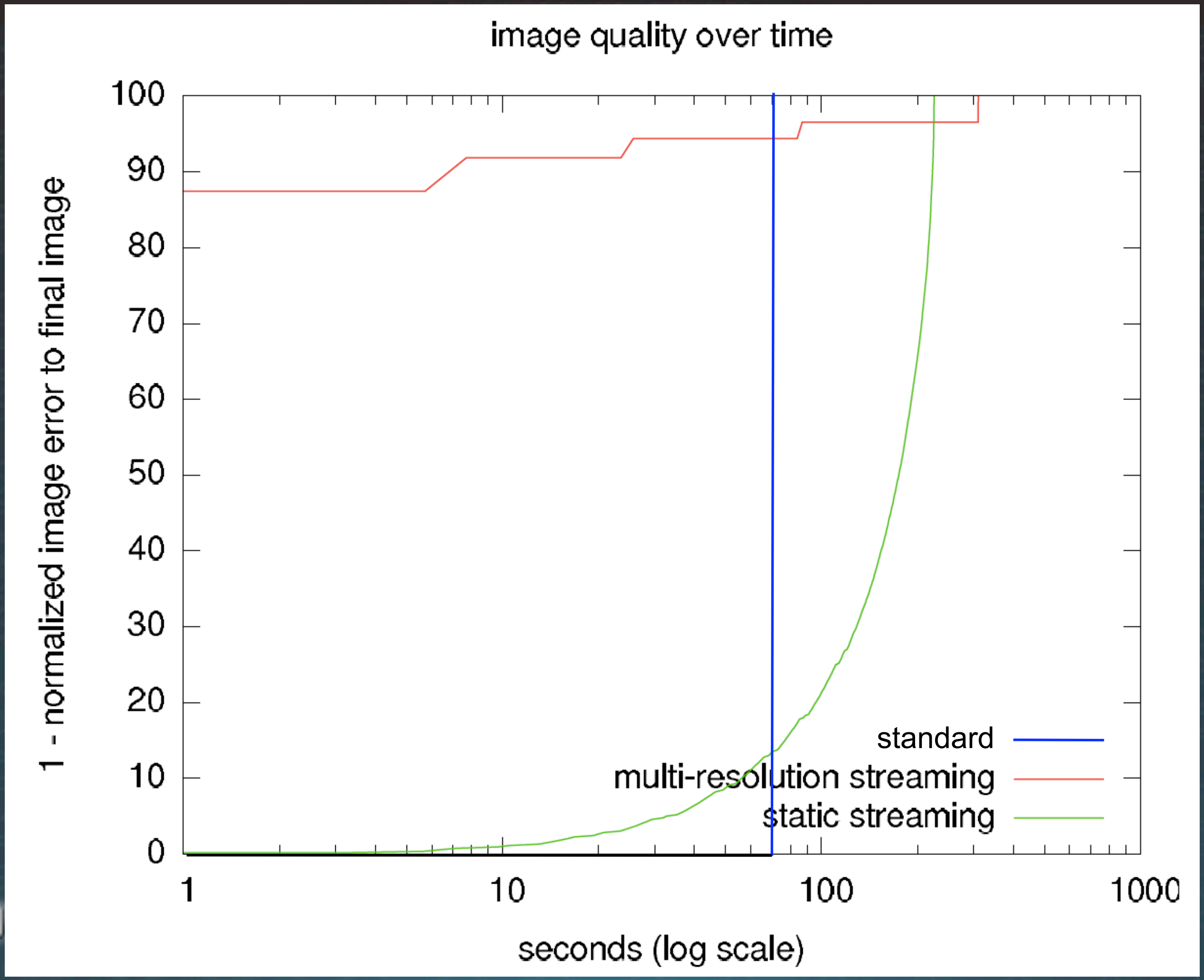


prioritized streaming

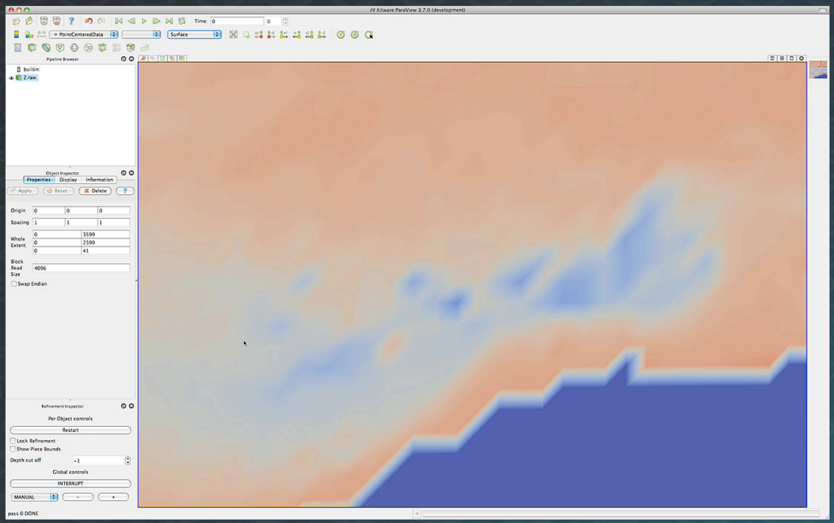


multi-resolution prioritized streaming

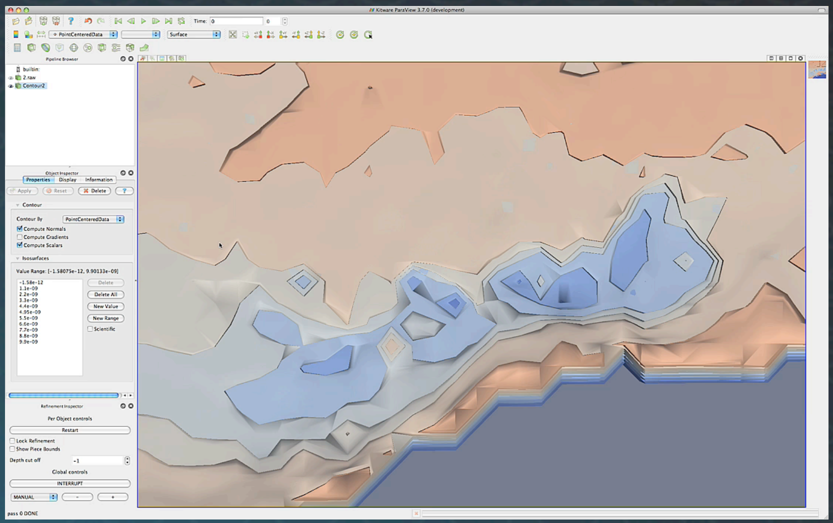
Image Quality over Time for Whole Extent Client Rendering



Culling and Multi-Resolution Everything

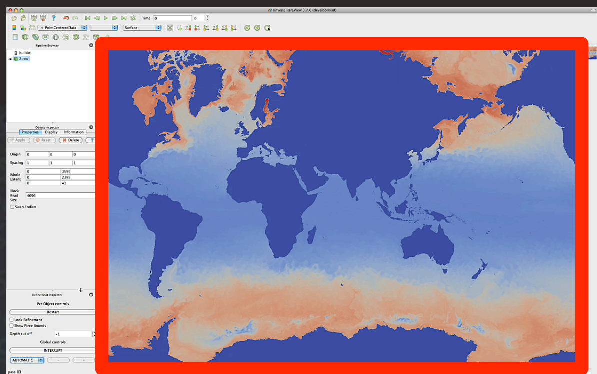


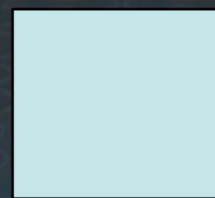
culling



isosurfacing

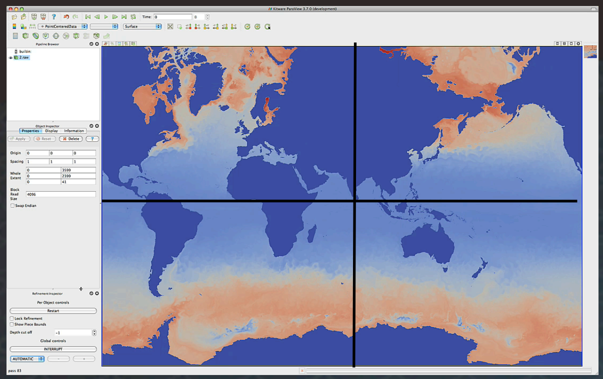
Multi-resolution Prioritized Streaming

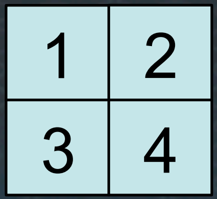




1) Send and render lowest resolution data

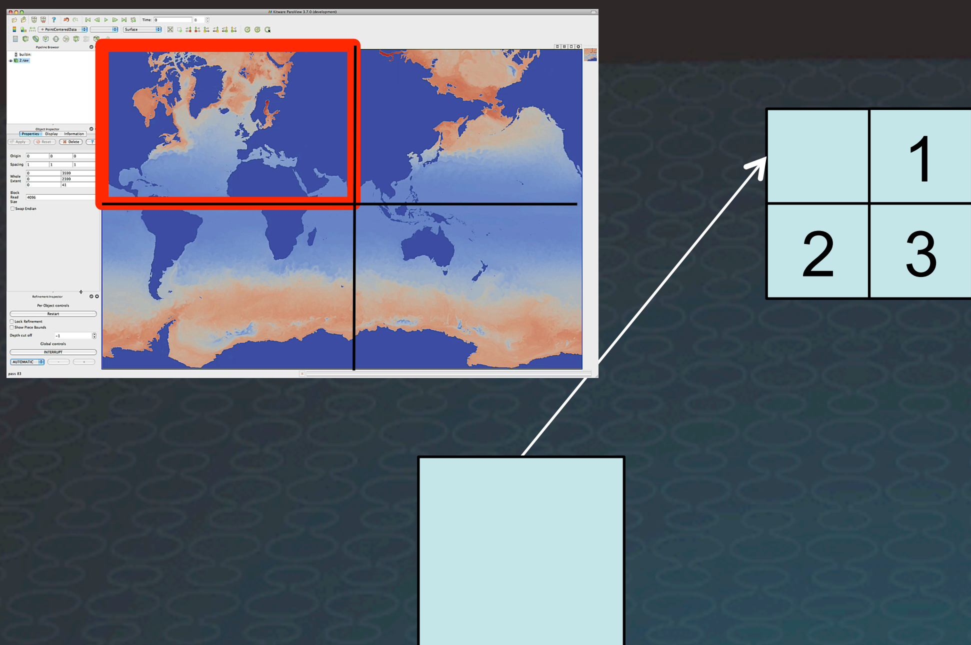
Multi-resolution Prioritized Streaming





1) Send and render lowest resolution data
2) Virtually split into spatial pieces and prioritize pieces

Multi-resolution Prioritized Streaming

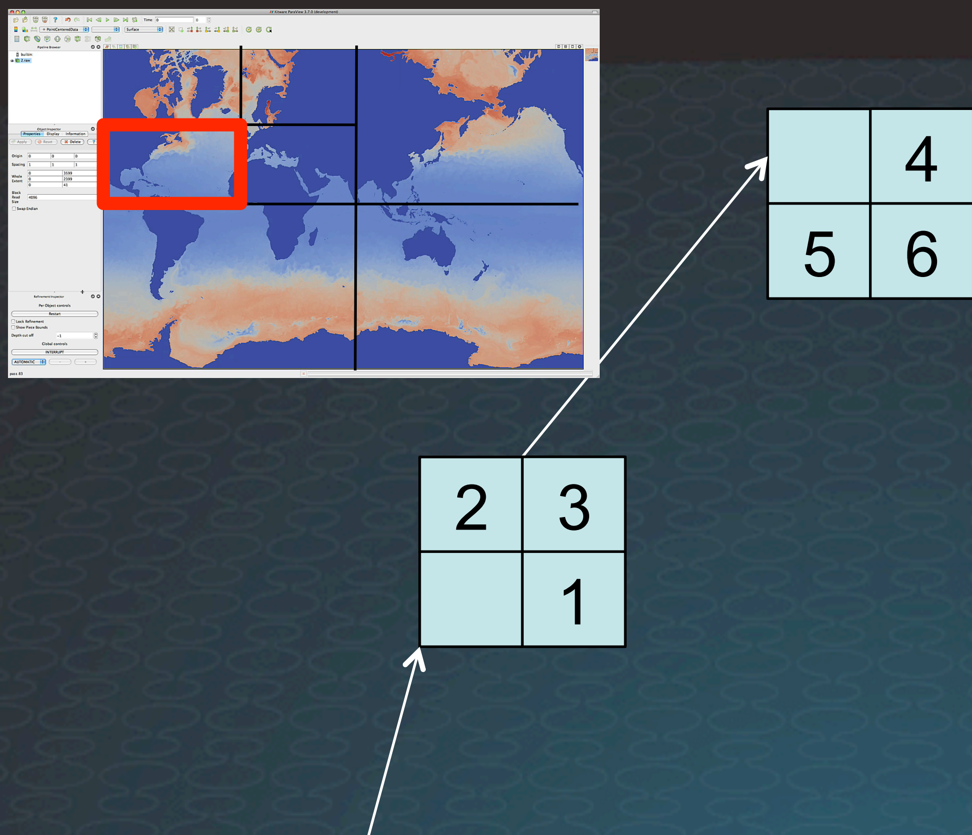


1) Send and render lowest resolution data
2) Virtually split into spatial pieces and prioritize pieces
3) Send and render highest priority piece at higher resolution

Multi-resolution Prioritized Streaming

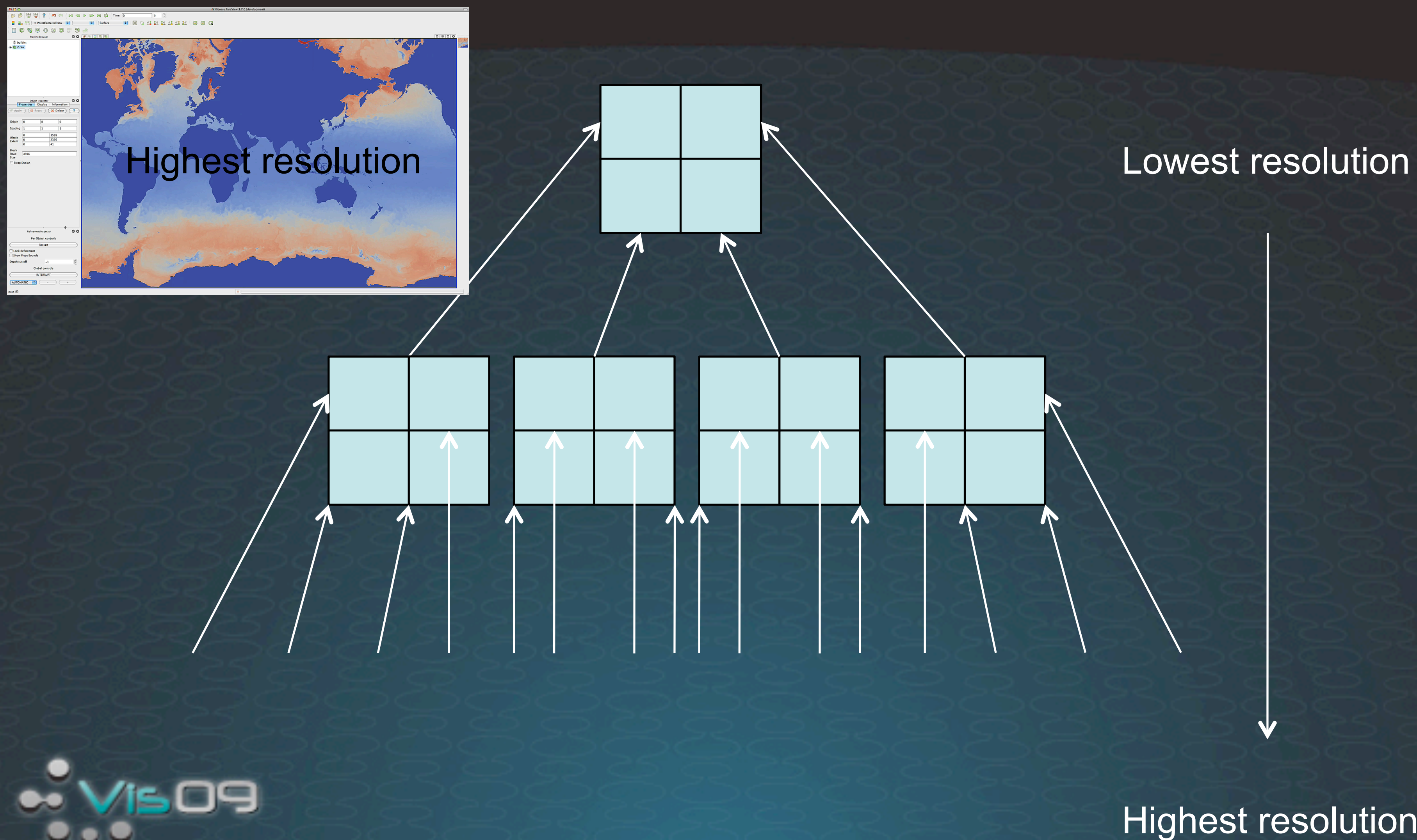
1) Send and render lowest resolution data
2) Virtually split into spatial pieces and prioritize pieces
3) Send and render highest priority piece at higher resolution
4) Goto step 2 until the data is at the highest resolution

Multi-resolution Prioritized Streaming



1) Send and render lowest resolution data
2) Virtually split into spatial pieces and prioritize pieces
3) Send and render highest priority piece at higher resolution
4) Goto step 2 until the data is at the highest resolution

Multi-resolution Prioritized Streaming



Adaptive ParaView Implementation

- Progressive multi-resolution renderer (upstream sink)
  - Implements the high level algorithm on the previous slides – also has a cache for re-rendering
  - Progressively updates and refines the rendering, by requesting pieces in priority order
- Multi-resolution preprocessor (generating source data)
  - Writes additional low resolution data to disk
  - Our implementation uses subsampling/striding – fast to generate (takes about the same amount of time to read the data once)
  - Doesn't modify the original data – left as-is (highest resolution) worst case uses x1 additional space

Adaptive ParaView Implementation

- Multi-resolution reader (downstream source)
  - The reader provides data pieces based on resolution request and piece request (spatial extent)
  - Uses the preprocessed multi-resolution data for fast reads
  - Multi-resolution tree helper class determines the axis splits, piece extents
- Meta-information keys (meta-data moving in the pipeline)
  - RESOLUTION request (what resolution is needed)
  - UPDATE_EXTENT request (what is the spatial extent of the piece needed)
  - Priority keys for prioritization sorting and culling

How to Run Adaptive ParaView

- Download CVS ParaView (make sure you have Cmake, Qt 4.5+)
- Build ParaView
  - PARAVIEW_BUILD_AdaptiveParaView ON
- Create the multi-resolution hierarchy (reader and hierarchy only for .raw float bricks currently)
  - adaptivePreprocess command line tool in bin directory
  - ./adaptivePreprocess <height> <degree> <rate> <i> <j> <k> <input file>

How to Run Adaptive ParaView

  - height = additional multi-resolution levels, degree = # pieces during refinement (power of 2), rate = striding/sampling spacing per axis on split, <i, j, k> = float brick data dimensions
  - example: height 4, degree 4, rate 2 = 4 additional multi-resolution levels, a piece is broken and refined into 4 pieces (split on 2 largest axes), downsample by 2x2 in largest dimensions for each level
- Start AdaptiveParaview (not the normal ParaView client)
  - Close the current view
  - Load the AdaptiveParaview plugin (vtkAdaptivePlugin.so/.dylib/.dll)

How to Run Adaptive ParaView

- Close the current view, again
- Open an Adaptive view
- Open the Preferences/Settings
  - Go to the Adaptive options
  - Enter your height, degree, rate of the multi-resolution preprocessed data
- Open your .raw float data
  - Enter your dimensions into extents (0, i – 1) (0, j – 1) (0, k – 1)
- Visualize it! (multi-resolution volume rendering works too, just tested it recently, turn off view prioritization)

Questions?

- Make your own multi-resolution reader and preprocessor
  - Reader needs to respond to resolution (refinement level) and update extent (pieces) information keys
  - See vtkRawStridedReader[1-2].*, vtkGridSampler[1-2].* and downsample.cxx for examples in Applications/AdaptiveParaView
- Work in progress
  - Client/server is being completed – currently only built-in works at the moment

Additional Contact Information

- woodring@lanl.gov Jon Woodring
- ahrens@lanl.gov Jim Ahrens
- dave.demarle@kitware.com Dave DeMarle
- ollie@lanl.gov Li-Ta Lo
- patchett@lanl.gov John Patchett

- These slides will be available online (soon)