Interactive Remote Large-Scale Data Visualization via Prioritized Multi-resolution Streaming

Jon Woodring, Los Alamos National Laboratory

James P. Ahrens[1], Jonathan Woodring[1], David E. DeMarle[2], John Patchett[1], and Mathew Maltrud[1]

[1]Los Alamos National Laboratory

[2]Kitware, Inc.

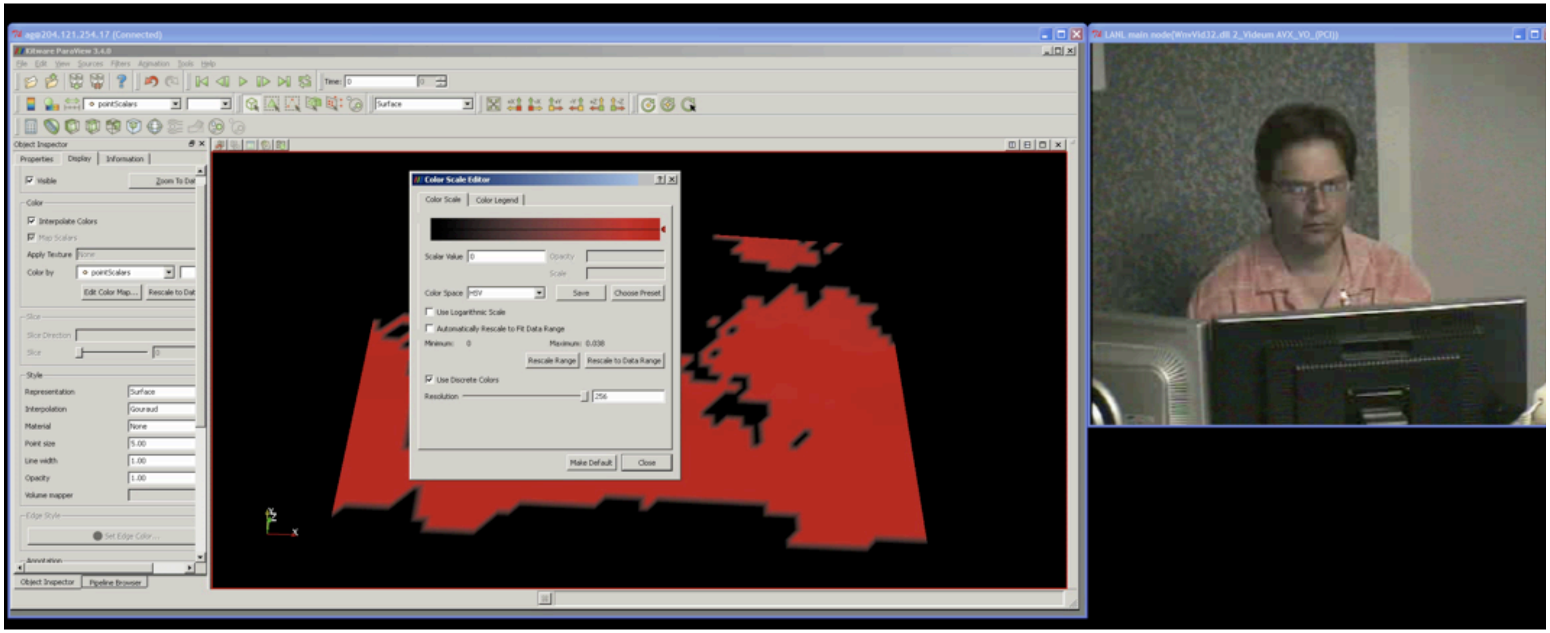
Executive Summary

- **Multi-resolution streaming visualization system for large scale data distance visualization**
  - Representation-based distance visualization (process data, send data, render client-side)
    - Alternative approach to image-based (process data, render data, send images)
  - Send low resolution data initially
    - Incrementally send (stream) increasing resolution data pieces over time and progressively render on the client side
    - Sends pieces in a prioritized manner, culling pieces that do not contribute
  - Implemented in ParaView/VTK and is publically available in the ParaView developer CVS archive
    - Works with most filters – the structural system changes only take place in the reader, renderer, and new pipeline messages

Adaptive ParaView Demo

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

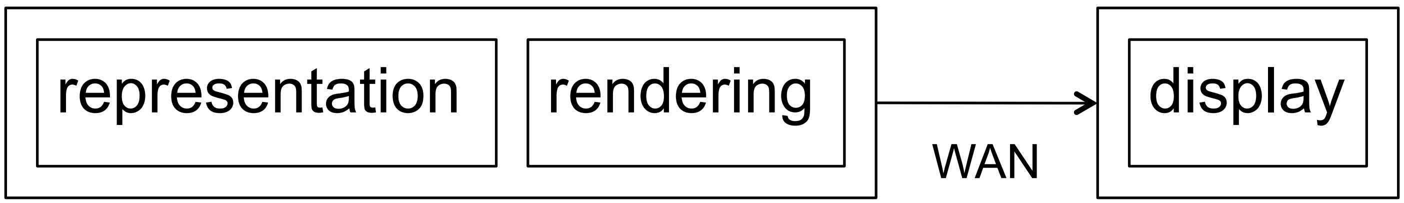


Remote LARGE Data

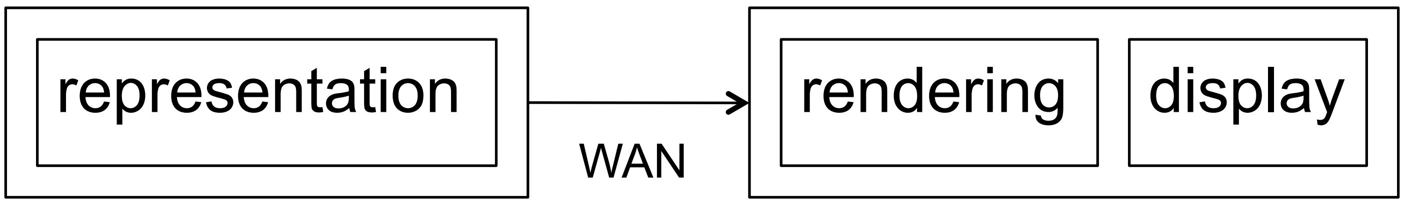
- **Using 100 TeraFLOPs of Jaguar (ORNL)**
  - 6 fields at 1.4GB each 20x a day
  - 3600 x 2400 x 42 floats
- **Transfer to LANL would take > 74 hours (~3 days)**
  - ~5 Mbps between LANL and ORNL
- **Unable to transfer the data from ORNL to LANL**
  - 250 TeraFLOPs
    - 12 fields
  - 1 PetaFLOP
    - 24 fields and 40x a day = 740 hours (~1 month)

Two Remote Visualization Approaches

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



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



Why use client side rendering for remote visualization?

- **Image-based distance vis: it works, but…**
  - Completely server side based (dumb client)
  - Frame rate is network latency and bandwidth limited
- **Client side rendering?**
  - Higher potential frame rate because of that nice client side GPU
  - Can render without needing the server (caching)
  - Explore the alternative approach for feasibility
- **Though… this is LARGE data – too big for the client, network, and display... Is it even practical to send representational data?**
  - The default mode is not practical, it can send data sizes on the order of the original data (isosurfacing a terabyte data set at full resolution can still be (mostly likely be) on the order of a terabyte)

Subset and Downscale 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		

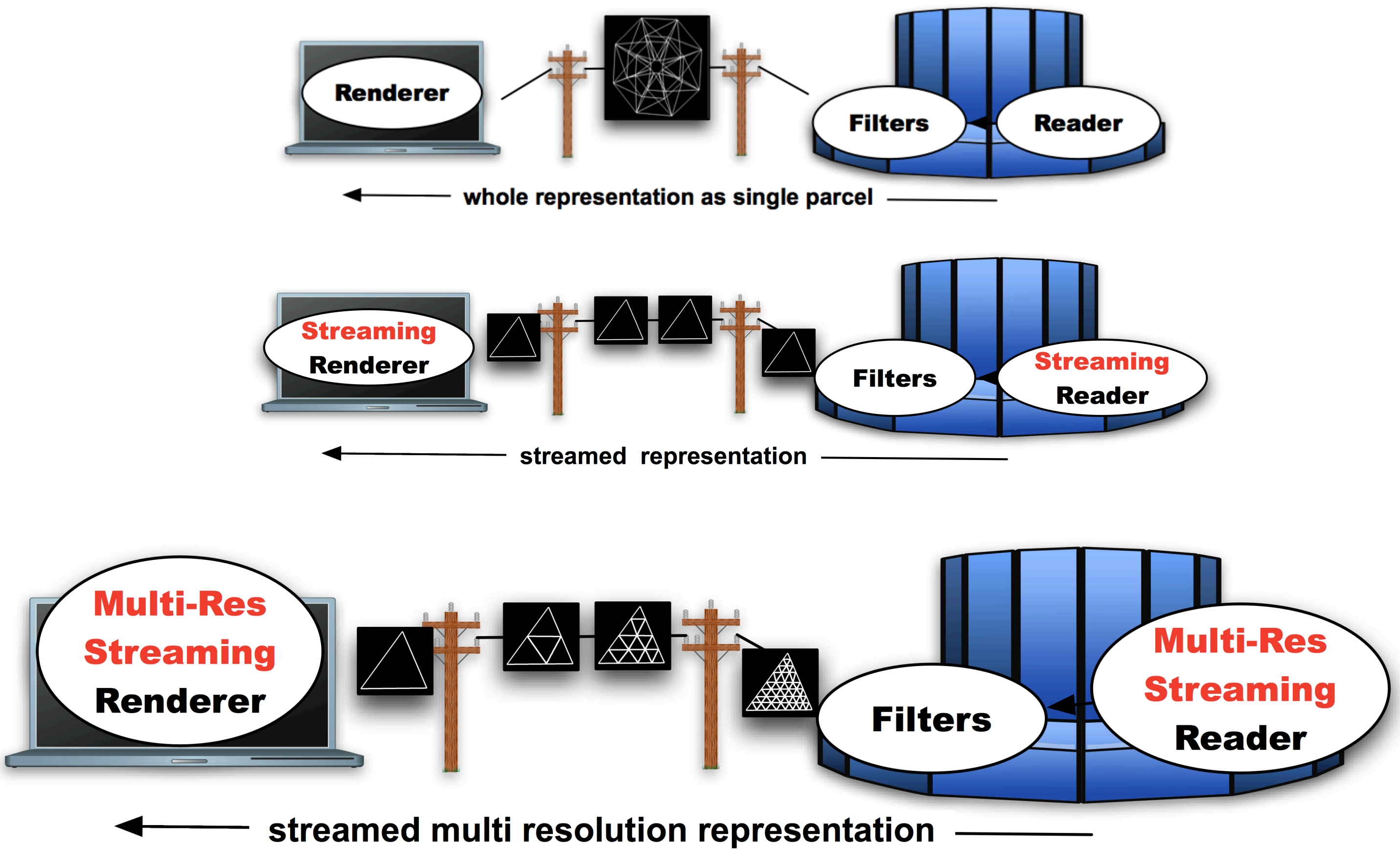
Downscaling
Sampling
Feature Extraction

The data has more points than available display pixels…
We need to reduce the data, anyways

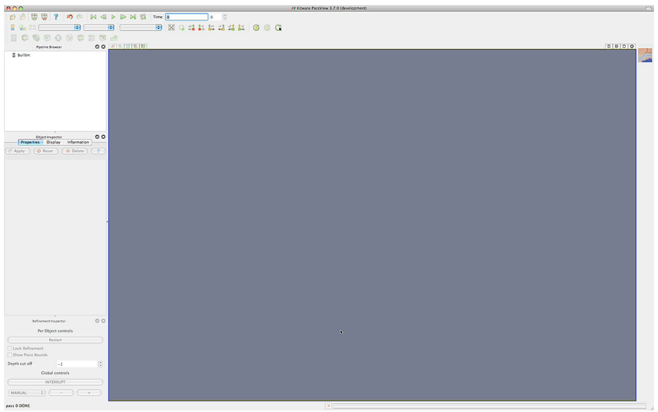
Multi-resolution and Streaming Related Work

- Pascucci and Frank
- Wang, Gao, Li, and Shen
- Norton and Rockwood
- Clyne and Rast
- LaMar, Hamann, and Joy
- Prohaska, Hutanu, Kahler, and Hege
- Rusinkiewicz and Levoy
- Childs, Duchaineau, and Ma
- Ahrens, Desai, McCormick, Martin, and Woodring

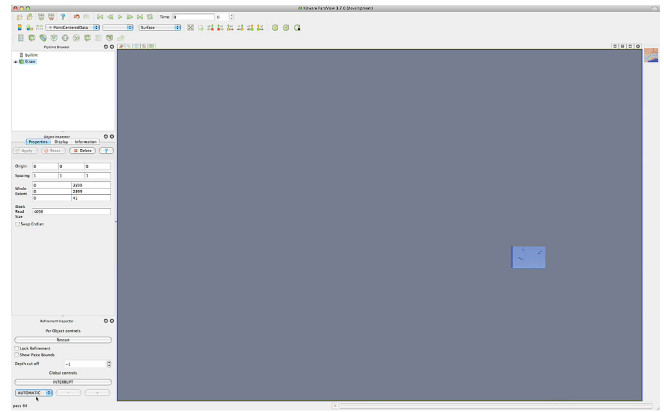
Standard, Streaming, and Adaptive Streaming Pipelines



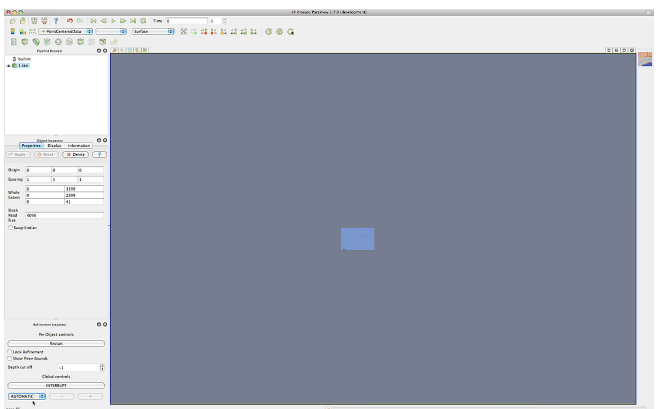
Pipeline Approaches in ParaView



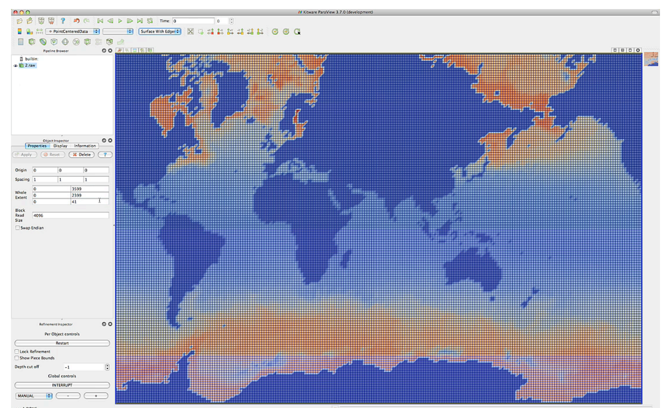
standard



streaming

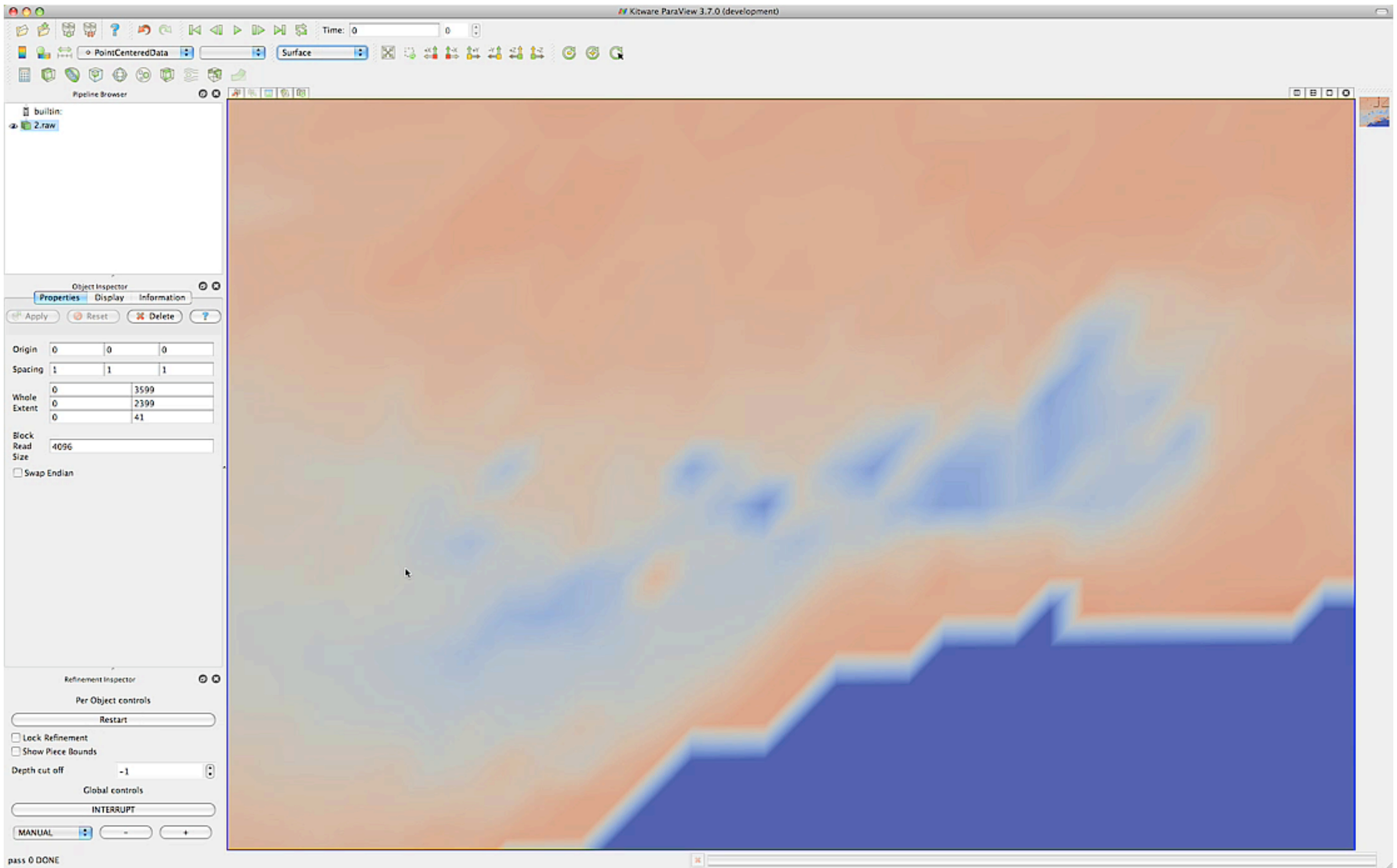


prioritized streaming

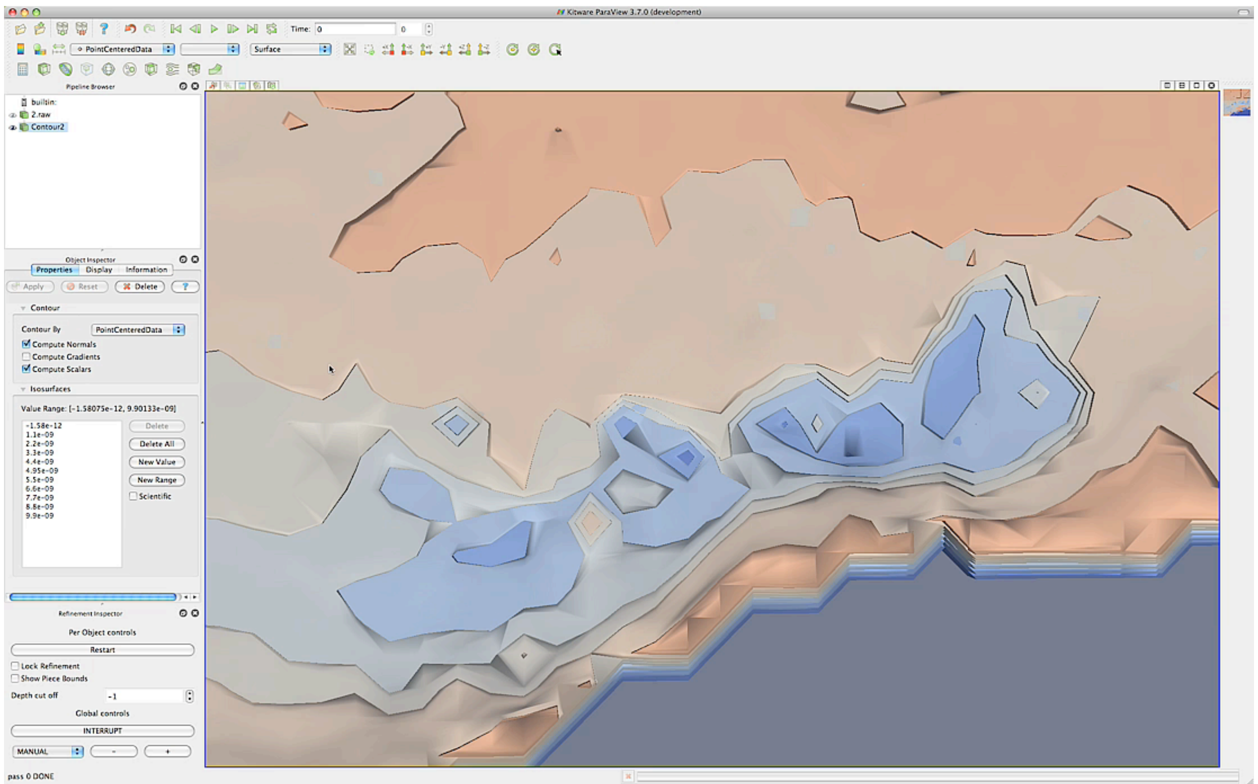


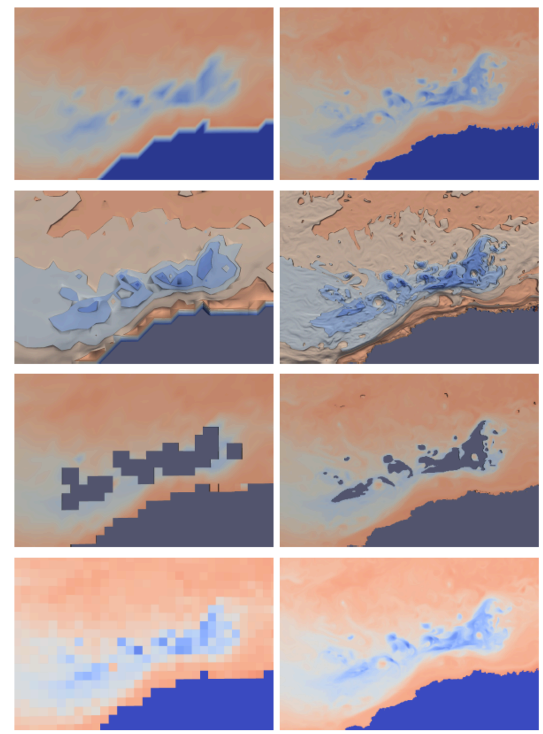
multi-resolution prioritized streaming

Using Culling and Prioritization to Improve Interactivity

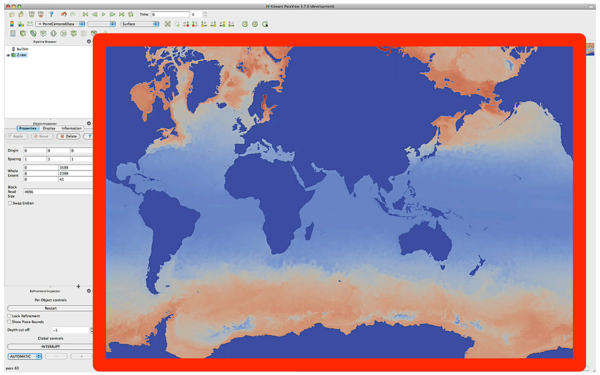


Multi-resolution Visualization System



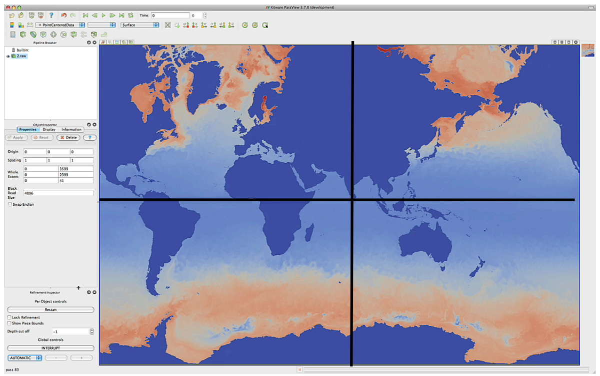


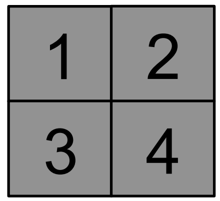
Multi-resolution Prioritized Streaming



1) Send and render lowest resolution data

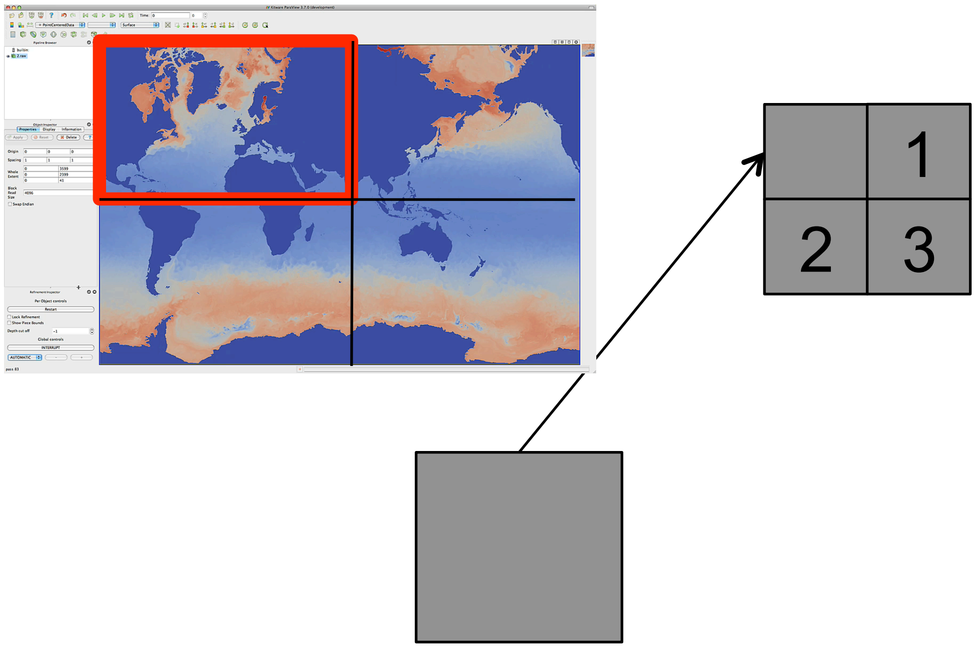
Multi-resolution Prioritized Streaming





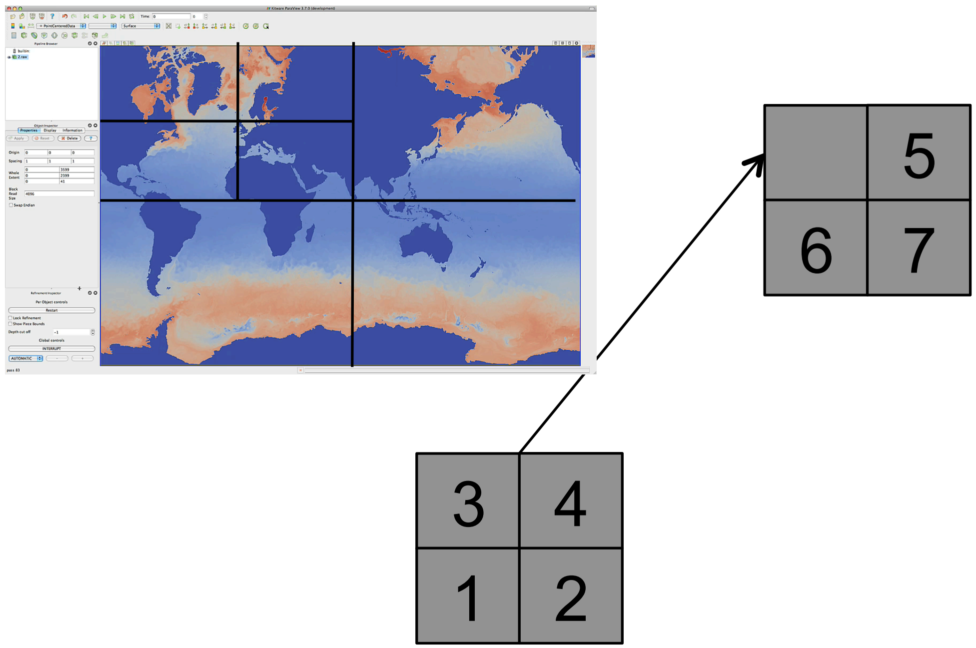
1) Send and render lowest resolution data
2) Virtually split into spatial pieces and prioritize pieces

Multi-resolution Prioritized Streaming



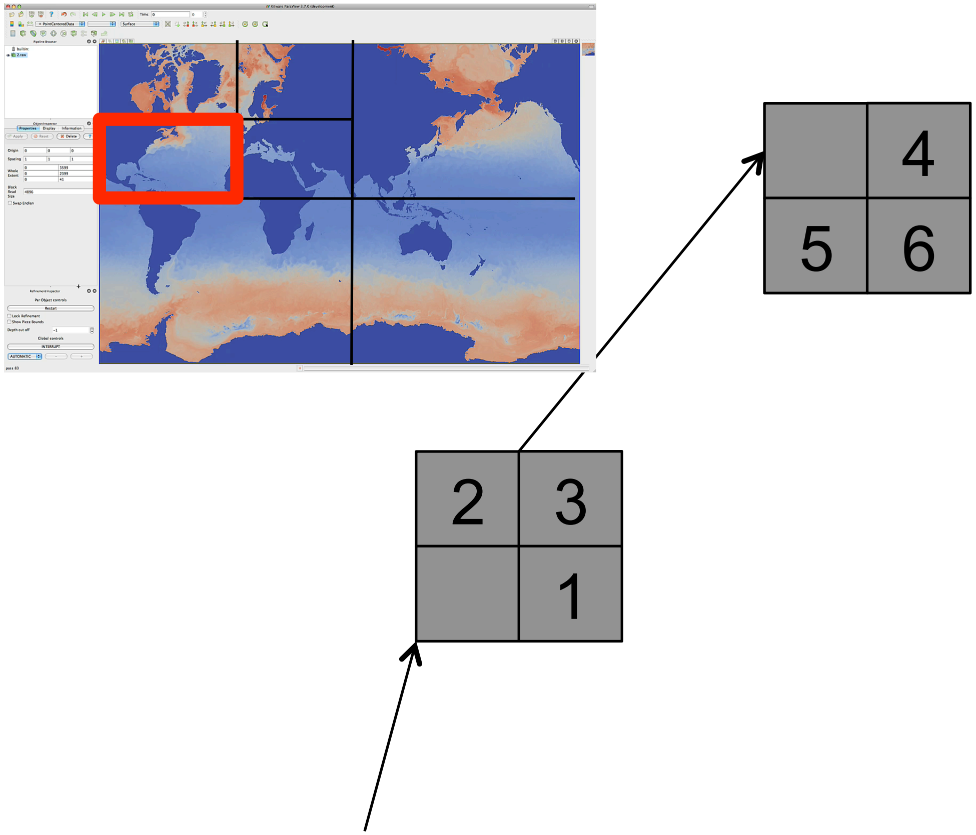
1) Send and render lowest resolution data
2) Virtually split into spatial pieces and prioritize pieces
3) Send and render highest priority piece at higher resolution

Multi-resolution Prioritized Streaming



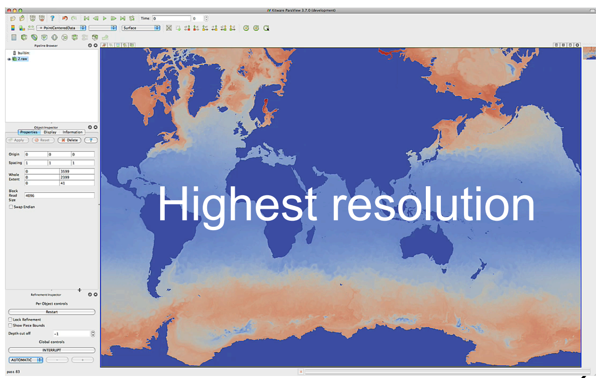
1) Send and render lowest resolution data
2) Virtually split into spatial pieces and prioritize pieces
3) Send and render highest priority piece at higher resolution
4) Goto step 2 until the data is at the highest resolution

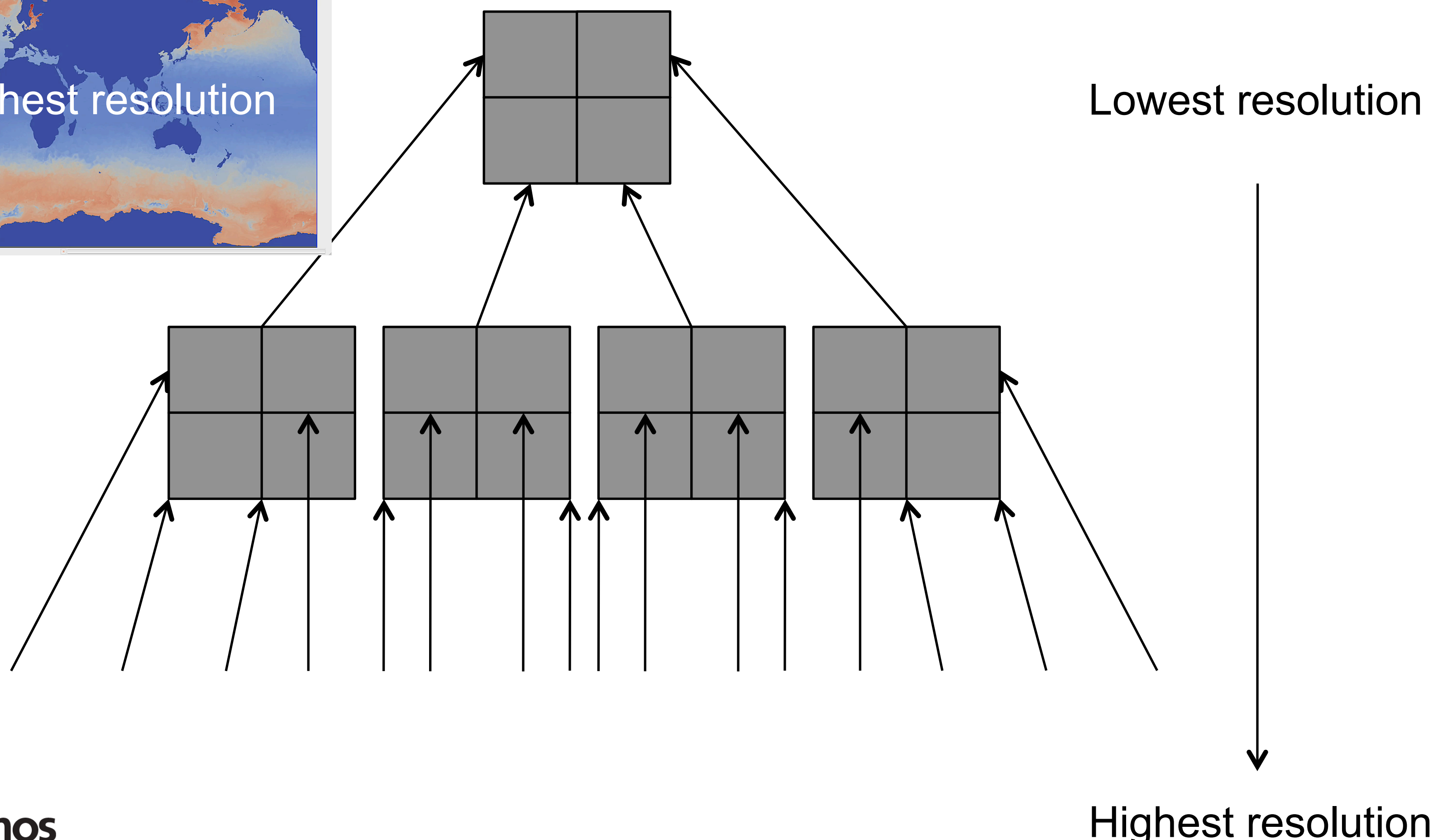
Multi-resolution Prioritized Streaming



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





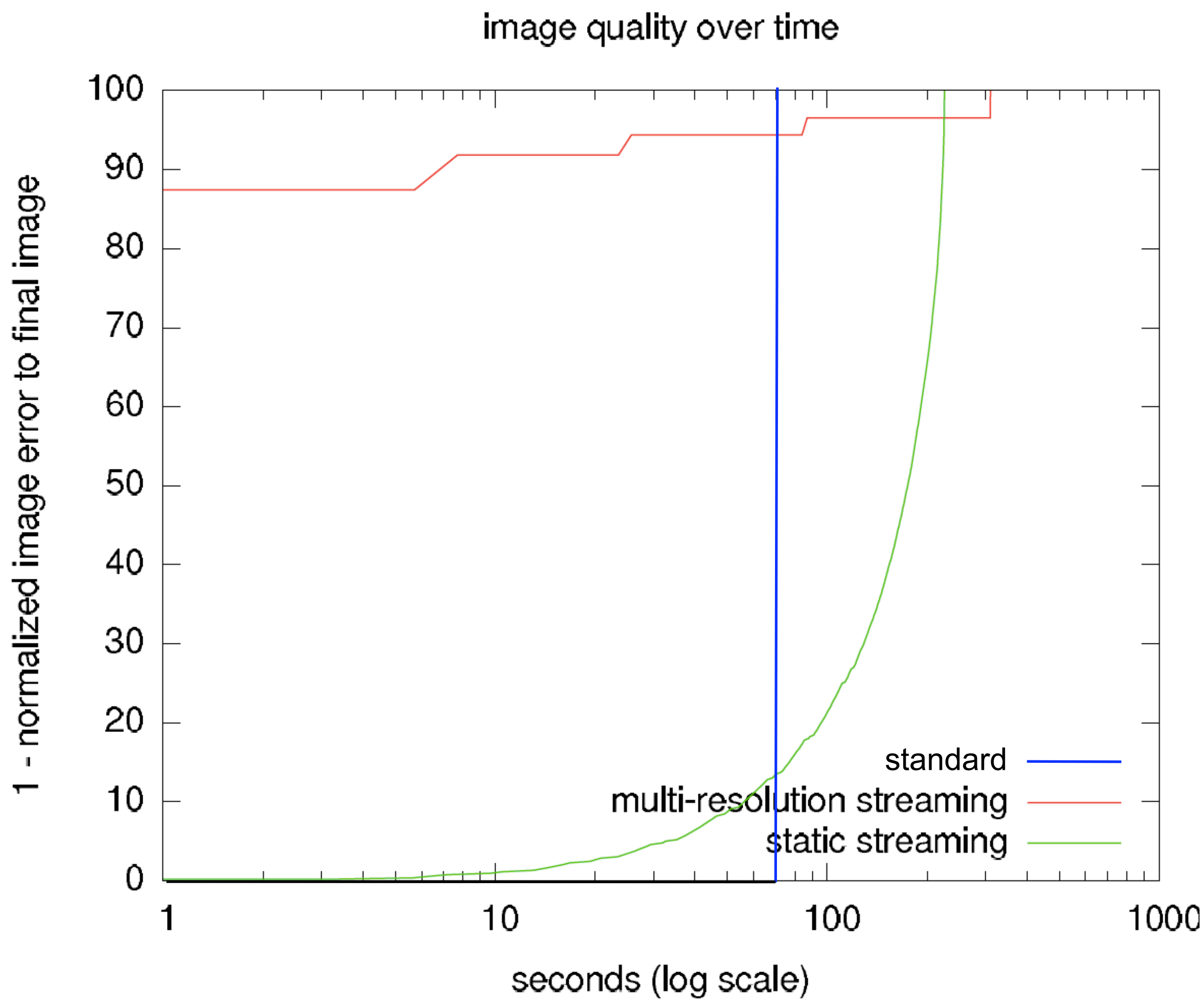
Adaptive Implementation

- **Progressive multi-resolution renderer (upstream sink)**
  - Implements the high level algorithm on the previous slides – also has a cache for re-rendering so data does not need to be processed and sent again
  - Progressively updates and refines the rendering, by requesting pieces in priority order
    - The highest priority is back to front (or front to back) prioritization for rendering accuracy (composition correctness)
    - Culls pieces if they are not in the view frustum
- **Meta-information keys (meta-data requests and information)**
  - New RESOLUTION information key (what resolution is needed)
  - Utilizes the UPDATE_EXTENT key (what is the spatial extent of the piece needed)
  - Priority information keys (from previous work in for prioritization sorting and culling)
    - Filters, if they are aware of the keys, are able to prioritize and cull pieces as well, otherwise the meta-information just passes through the filter unaltered

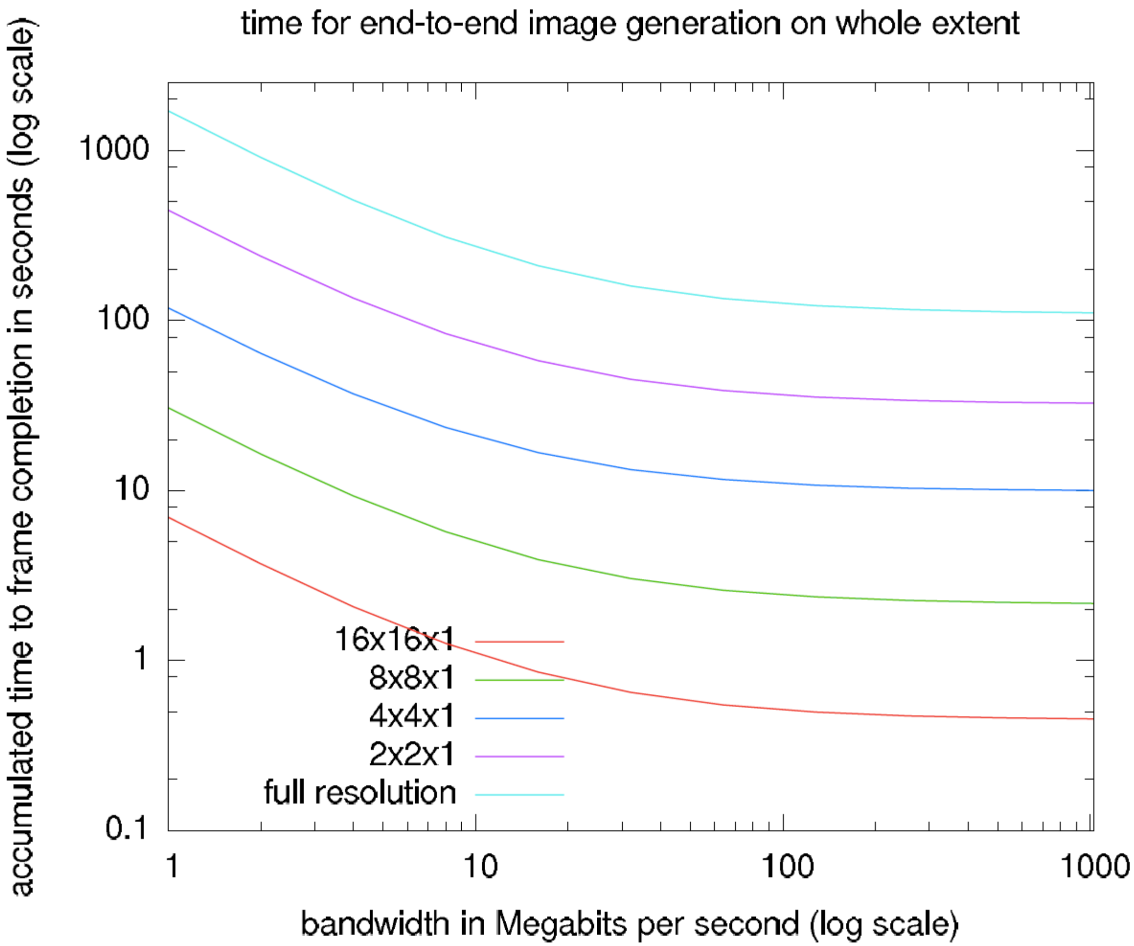
Adaptive Implementation

- **Multi-resolution reader (downstream source)**
  - The reader provides data pieces based on resolution and piece request keys (spatial extent) that moves down the pipeline
  - Uses preprocessed multi-resolution data for fast reads
    - Multi-resolution tree helper class determines the axis splits, piece extents
- **Multi-resolution preprocessor (generating source data)**
  - Writes additional low resolution data to disk in the same data format (multiple files, just pre-downsampled)
  - Our test implementation uses striding (nearest neighbor sampling) – fast to generate (takes about the same amount of time generate as to read the data once)
    - Easy to incorporate filtering for higher quality low resolution data – just change the sampling kernel
  - Doesn’t modify the original data – left as-is (highest resolution)
  - Worst case uses N additional space, more likely to use N/2 or N/3 additional space

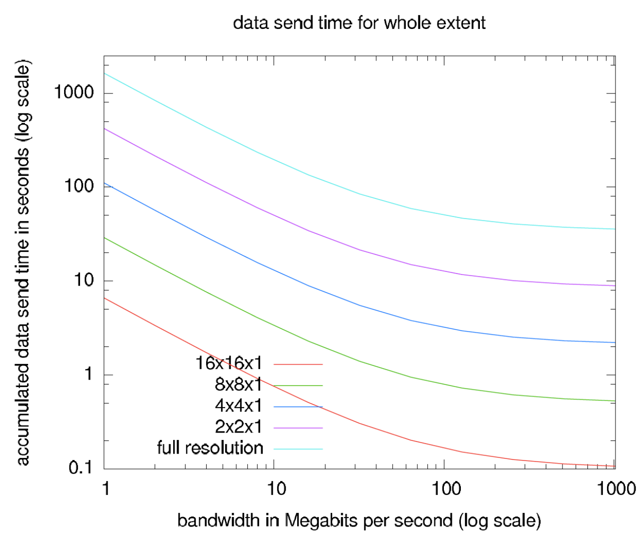
Image Quality over Time for Whole Extent
(POP 3600 x 2400 x 42 floats, 10 MBps, 100 ms latency)



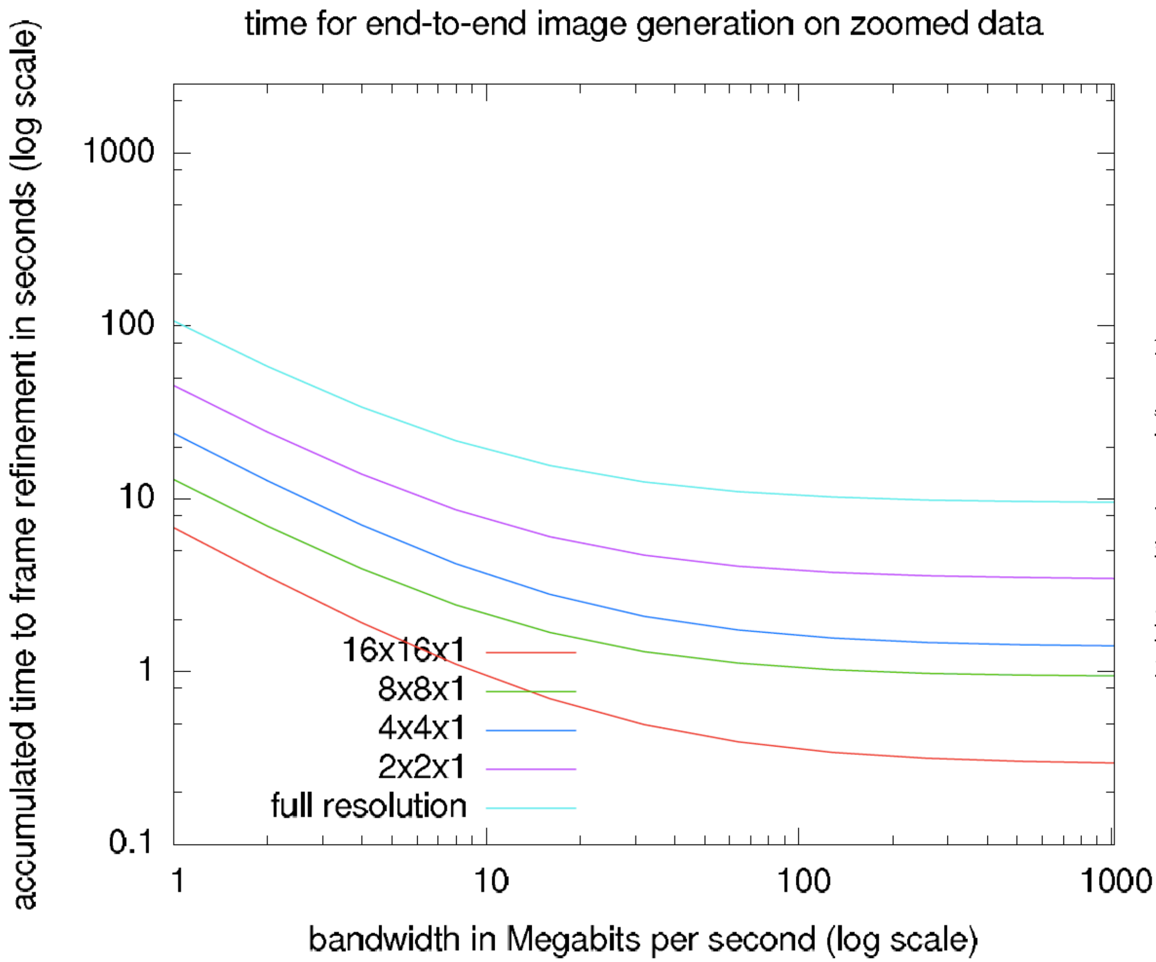
Whole Extent (POP data, 100 ms latency)
Total Rendering, Client Rendering, and Send Time



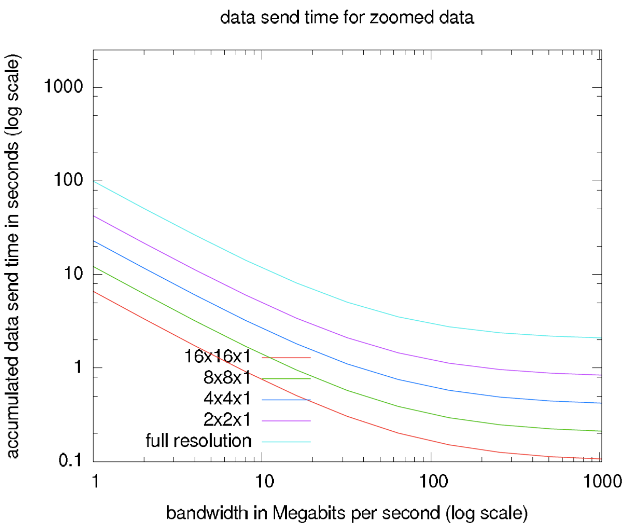
Full Extent	16x16x1	8x8x1	4x4x1	2x2x1	Full
Render	0.03 s	0.10 s	0.38 s	1.4 s	5.6 s



Zoomed In (Culling and Prioritization) (same params) Total Rendering, Client Rendering, and Send Time



Zoomed	16x16x1	8x8x1	4x4x1	2x2x1	Full
Render	0.03 s	0.03 s	0.05 s	0.09 s	0.27 s



Cold Start Read and Write Timings (POP data)

Time to Read Whole File	Time to Read and Create 4 levels	Time to Read and Create 20 levels
30.0 s	46.5 s	150.6 s

Full Extent	16x16x1	8x8x1	4x4x1	2x2x1	Full
Read	0.18 s	0.92 s	5.0 s	11.8 s	35.6 s
Accum. Read	0.18 s	1.1 s	6.1 s	17.9 s	53.5 s

Zoomed	16x16x1	8x8x1	4x4x1	2x2x1	Full
Read	0.18 s	0.47 s	1.4 s	2.8 s	6.1 s
Accum. Read	0.18 s	0.64 s	2.0 s	4.8 s	11.0 s

Thank You

- **Multi-resolution Distance Visualization System**
  - Overviews obtained quickly
  - Increasing details over time
  - Zoomed details on demand
  - Fast client side rendering
  - Usable for large scale local visualization, too – possibly integrate into render server, as well (multi-resolution used on supercomputer)

- **This work was funded by the DOE Office of Science ASCR**

  - woodring@lanl.gov Jon Woodring
  - ahrens@lanl.gov Jim Ahrens
  - dave.demarle@kitware.com Dave DeMarle
  - patchett@lanl.gov John Patchett
  - maltrud@lanl.gov Mat Maltrud

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>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

How to Run Adaptive ParaView

- **Start AdaptiveParaview (not the normal ParaView client)**
  - Make sure the AdaptiveParaview plugin is loaded (vtkAdaptivePlugin.so/.dylib/.dll)
  - Close the current view
  - 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