

The hyperwall

Timothy A. Sandstrom
Exploratory Computing Environments Group
NASA Ames Research Center
sandstro@nas.nasa.gov

Chris Henze
Exploratory Computing Environments Group
NASA Ames Research Center
chenze@nas.nasa.gov

Creon Levit
Exploratory Computing Environments Group
NASA Ames Research Center
creon@nas.nasa.gov

Abstract

This paper describes the hyperwall, a visualization cluster that uses coordinated visualizations for interactive exploration of multidimensional data and simulations. The system strongly leverages the human eye-brain system with a generous 7x7 array of flat panel LCD screens powered by a Beowulf cluster. With each screen backed by a workstation class PC, graphic and compute intensive applications can be applied to a broad range of data in parallel. Navigational tools are presented that allow for investigation of high-dimensional data spaces.

1 Introduction

That data are growing in size and complexity is self evident, no more so than when it comes to multidimensional/multivariate (MDMV) data. The scientific visualization and information visualization literatures are filled with research delving into the issues associated with this data explosion. A common theme emerges: sooner or later, machines run out of some crucial resource, be it CPU, graphics, screen real estate, memory, or disk bandwidth. Screen space, for instance, has been a limiting factor in MDMV visualization systems ever since the first brushed scatterplot matrix [1, 2]. One can display only so many windows, can present only so many variables in a single view before reaching a point of diminishing returns.

We seek to interactively explore large MDMV datasets and families of parameterized simulations. Towards this end we have assembled a system using a combination of commodity hardware and custom software known as the 'hyperwall'. Combining a phalanx of pixels and processors, we seek to overcome some of the graphics and com-

putational limitations found in many MDMV visualization systems, and to work towards a true problem solving environment where many tools can be brought to bear on a given problem at once.

There are many approaches to MDMV visualization, the goal typically being to visually summarize and interact with the data searching for trends and relationships. Tools such as XGobi [4] and XmdvTool [7], use techniques such as scatterplots, glyphs, and parallel coordinates to display MDMV data in lower dimensional projections. Direct manipulation techniques such as interactive brushing are used to find relationships between variables. These packages draw on the works of authors such as Tukey and Cleveland, providing a rich set of the classic statistician's tools. Other research has focused on multiple, coordinated views or visualizations of related data. North and Shneiderman [3] have provided a useful taxonomy of these techniques that is applicable across a broad spectrum of problems. Finally, a number of tools have taken a spreadsheet approach to MDMV visualization [8, 9, 10], where the layout of the views has inherent meaning, implying location and allowing navigation in a given high-dimensional space. This approach also allows a high degree of coordination between views, for instance when a given modification or operation is applied to all visuals in a column or some other subset of the matrix.

The hyperwall exists at the confluence of these streams, combining spreadsheet metaphors, direct manipulation, and multiple linked coordinated views.

2 System Architecture

Our system architecture is summarized in figure 2. The displays in the hyperwall are 18" Samsung 181T flat panel monitors. Each LCD is framed by a uniform black plas-

tic bezel approximately 3/4" in width. A custom designed mounting rack (see figure 1) allows pitch and yaw adjustment of each monitor, as well as translational adjustment between rows and columns. In addition, each monitor can be moved independently up to 14" in "z" - perpendicular to the frame - to allow nonplanar arrangements of the viewing surfaces - for example spherical or paraboloid sections. We designed the rack to provide all these degrees of freedom in order to compensate for the directionality in the monitors, and to accommodate different viewing distances. In practice, the Samsung monitors deliver well on their promised 170 degree omnidirectional field of view, so a wide range of adjustments is effective.

Each display is driven directly by an NVidia GeForce 4 Ti4600 graphics card, at 1280x1024 resolution. The aggregate pixel count for the entire display matrix is thus $7 \times 7 \times 1280 \times 1024 = 64,225,280 = 64$ Mpixels, distributed over some 55 square feet of screen real estate.

Each graphics card is housed in a dual-CPU AMD Athlon MP2000+ rackmounted slave node. The slave nodes each have a 100GB IDE disk, thus providing aggregate storage of 5TB. The slaves are driven by a similarly configured master node, and all communication is via Fast (100 BaseT) Ethernet, coordinated by a pair of Cisco Catalyst 2950 G-48-EI switches.

3 Coordinated Visualizations

In our system, nodes either work together to display one scene (like a powerwall), or they each individually display separate but related scenes. Coordination in the powerwall sense requires that all nodes render the same scene at the same time with an appropriately related set of viewing transformations. At the other end of the spectrum, each node displays possibly a different dataset, or the same dataset using a different rendering parameter, or even a completely different visualization technique. Coordination in these cases may mean that all the nodes need to have the same viewing transformations, or that the same colormap is used across the different datasets, or that items selected in one view are highlighted in other views.

In cases where we want ganged control, a particularly useful ploy is to simply replicate mouse and keyboard events on the master node and broadcast them to the slaves. This strategy allows us to run many standalone X Window System event-driven applications virtually unchanged as hyperwall SPMD applications. For controlling true MPMD applications, which may feature distributed control interfaces and peer-to-peer communication patterns, we use a distributed object framework with a robust signal/slot event service, the details of which are outside the scope of this report.

3.1 Using X to interact with the nodes

Using the X Test extension distributed with X11R6, we can send simulated mouse and keyboard events to X servers running on any of the nodes. Using custom software, called hyperx, a user can sit at the master node and interact simultaneously with any number of nodes in the matrix. This allows one to change such things as view perspective, color mapping, visualization type, or any parameter of a visualization accessible via keyboard input or a GUI. Imagine interactively changing a cutting plane position through 49 different datasets at once, and you begin to see the possibilities of such a system. Another nice feature is the ability to move the mouse and keyboard focus from screen to screen as if you are interacting with one very large virtual desktop.

3.2 Issue: Maintaining View Coherence

In powerwall mode, when a group of nodes is cooperating to show a single scene, all the nodes must agree upon the current set of viewing transformations. Similarly, when groups of nodes independently render related objects, we typically want to have the same viewpoint across all of them. Transformations or viewpoints are usually modified via the mouse or keyboard. Thus, we can often achieve view coherence by sending the exact same event stream to each node though there are again some subtleties that may require modifications to software running on the nodes.

When its view is being adjusted with the mouse, an X-based application will typically pass through the main event loop many times as the mouse is dragged. Since we want all ganged nodes to respond identically, any source of asynchronicity in the event production or consumption must be hunted down and removed. Thus for example, all event compression must be turned off. Other sources of asynchronicity are X workprocs, and callbacks from I/O events on sockets registered via XtAppAddInput. Self-generated X events can be another source of disparity in the event stream and hence lead to divergence of transformations and viewpoints.

Once these and other sources of randomness in either the event stream production or consumption have been removed, the remaining issue is that of graphics throughput. Suppose, as you are drawing your scene across four nodes ganged in a 2x2 array, one node's portion of the view frustum happens to contain the bulk of the polygons in an iso-surface being rendered. Its frame rate drops to say, 3 fps while the other nodes gallop merrily along at 15 fps. The views will in this case diverge until the user releases the mouse, whereupon the views will again converge when all nodes in the gang have digested all the events in their identical event streams. If needed, a finer-grained approach using some sort of distributed synchronization mechanism such

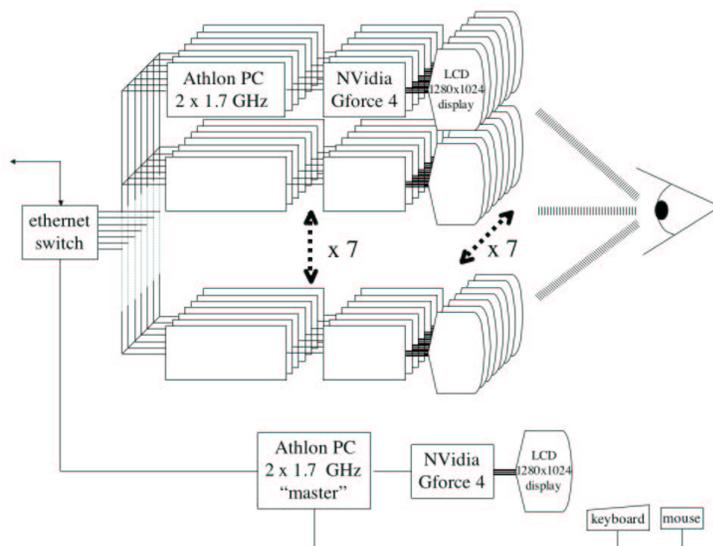


Figure 2. Architecture.

as a barrier can remove this remaining issue, and provide frame for frame view coherence. This is needed for things such as synchronized animation.

3.3 Spreadsheet-Based Visualization

Since we lay our visualizations out in a matrix, our system is related to spreadsheet-based visualization systems [8, 9, 10] and gains many of the inherent benefits thereof. The position of a visualization in our matrix of screens can be directly related to a location in a high-dimensional space, providing the user with a necessary context for understanding and navigating MDMV spaces. For instance, figure 3 shows a parameter study of the Reusable Launch Vehicle. Each row displays a different angle of attack, each column a different Mach number. The user knows, at a glance, the parameters associated with the dataset in each view.

Like other spreadsheet-based systems, the simultaneous display of related visualizations facilitates a broad range of primarily visual activities such as comparing and contrasting related images, tracking features between timesteps, and finding patterns amidst the complexity of a family of bivariate scatterplots. Furthermore, with aggregate visualizations (using possibly several different applications), we can explore visualization space as well as data space, looking at our data in a number of ways at once. Our system provides well for these visual tasks by providing high resolution views of all visualizations and allowing for a high degree of interactivity with these views by the user.

3.4 Examples

Here are examples of some of the first few SPMD applications we have implemented on the hyperwall. They are essentially all preexisting standalone applications, slightly modified, replicated, and running independently on each node. In these simple examples, each node runs the same program using its own local data. The user interfaces are all driven in parallel by mouse and keyboard events broadcast from the master node (see section 3.1).

Molecular quantum mechanics: A set of related molecules is arranged on the wall as a two-dimensional “molecular periodic table”. Each display shows the same quantum mechanical observable (*e.g.* electrostatic potential) for its molecule, using volume visualization in the case of a three dimensional scalar field. The viewing transformations and transfer functions are the same for all nodes, and can be changed in real time.

Computational aerodynamics: Each display shows surface pressure and streamlines for the same vehicle simulated at a different mach number and angle of attack. Mach number increases from left to right, angle of attack increases from bottom to top. That is, all vehicles in the same column are simulated at the same mach number, and all vehicles in the same row are simulated at the same angle of attack. All displays share the same viewing transformations and share the same mapping from surface pressure to colors. Again, all visualization parameters can be interactively changed. See figure 3.

Weather modeling: Each display shows a three dimensional scalar field from a particular timestep of a weather



Figure 3. Examining an RLV parameter study.

simulation. All displays in the same row show the same quantity (at successive timesteps), and all displays in the same column show the same timestep (but different physical quantities, *e.g.* temperature, pressure, humidity, *etc.*). All displays in a given row share the same visualization parameters (*e.g.* the same isoscalar surface values), and these may be interactively manipulated for the row together. All displays share the same viewing transform, which is also dynamic.

Planetary geology: Each row displays a different quantity associated with Mars. The first row shows visible imagery, the second shows laser altimetry, the third shows difference between daytime and nighttime temperatures, *etc.* Spatial registration is maintained between rows, and all rows are interactively zoomed and panned identically.

Remote sensing: Each display D_{ij} shows a satellite image of the same region of a planet. Displays $D_{i=j}$ along the main diagonal show the image in band i , displays $D_{i>j}$ show the ratio image of band i to band j , and displays $D_{i<j}$ show another binary function of the bandwise images. All displays are interactively panned and zoomed together. Image processing operations can be applied to any subset of the images.

3.5 Caveats

One way in which we differ from spreadsheet-based visualization systems is that, in general, we do not have a built in programming language. This would allow one, for instance, to calculate the difference between the scalar fields on two nodes, and display the results on another. However,

this might require a high degree of integration between software on the master node and software running on the nodes. While we have written applications that are as tightly coupled as this, node application software can often be run unmodified. This loose coupling greatly extends the number of applications we can use on the cluster especially those for which we do not have source. Alas, some features may require code modification no matter what. For example, synchronized animation across multiple nodes will typically involve some external synchronization mechanism probably not anticipated in a given application.

3.6 Powerwall mode

Sooner or later, everyone eventually asks if we can show one image over all the screens (like a powerwall). The answer is yes, with some reservations. Unless you are using Chromium [12] to divide up of the graphics work, in general, node application software will need to be modified. We have used Chromium on our cluster but at the time there were performance issues related to using only fast ethernet between nodes, as well as issues with display lists and applications using multiple windows.

For 2D scenes, such as rendering an image, there is an implied XY offset into the image based upon its location in the wall. Image rendering software would need to calculate the relevant subrectangle for its portion of the view, possibly affecting I/O, buffering, and display routines. Displaying 3D scenes across an array of screens can be achieved by dividing up the view frustum appropriately and having each node render the entire scene. While this seems wasteful, the

efforts required for culling the scene via octrees, or other suitable schemes, usually does not become necessary until the rendered scenes become completely unwieldy. Modern graphics cards like the GeForce4 and its contemporaries are quite capable of rendering millions of polygons per second. Of course, one could come up with a giga-polygon isosurface to foil us here but in practice, this has not been an issue. View coherence is an issue however as discussed in section 3.2.

4 Navigating in Hyperspace

The complexity of MDMV datasets inspires us and cries out for novel tools to search around for new unseen relationships. Inevitably, there is the tension between complexity of task and simplicity of interface.

For example, navigation is one of the primary challenges in MDMV visualization. Useful interfaces for high-dimensional navigation need to be intuitive, flexible, and provide contextual information. A good example of MDMV navigation is the HyperSlice [6] interface. The user is presented with an array of bivariate plots, each of which is centered on an n -dimensional point $\mathbf{C} = \langle \mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n \rangle$. This point can be moved around in n -space by direct manipulation. By grabbing in the $\mathbf{X}_1, \mathbf{X}_2$ subplot the user changes those coordinates of \mathbf{C} , leaving all the other dimensions $\mathbf{X}_3 \dots \mathbf{X}_n$ constant. This interface gives immediate feedback to the user via direct manipulation, provides contextual information for the user through the meaningful layout of the subplots, and allows the user to navigate the center point \mathbf{C} , anywhere in the n -dimensional space.

We have implemented a high-dimensional browser that shares some features of the HyperSlice paradigm, as well as its recent extension called HyperCell[5]. Our browser is currently more limited in application than those two schemes, since it is specialized for six-dimensional scalar fields, but for this class of data we feel it provides a more powerful interface than either HyperSlice or HyperCell.

In our 6D Browser, we select 3 dimensions of the data, say $\langle \mathbf{X}, \mathbf{Y}, \mathbf{Z} \rangle$, for display on the master console. In this scene, we embed a 7×7 array of sampling points, arranged in a regular planar array. The plane can be scaled, translated, or rotated freely, in order to position the sampling points. Each of the 49 points corresponds to a screen on the hyperwall, where we display the remaining 3 dimensions of the data (call them $\langle \mathbf{U}, \mathbf{V}, \mathbf{W} \rangle$). Thus each of the 49 hyperwall screens shows the entire $\langle \mathbf{U}, \mathbf{V}, \mathbf{W} \rangle$ field, for some fixed $\langle \mathbf{X}, \mathbf{Y}, \mathbf{Z} \rangle$ determined by the location of its corresponding point on the master display: $\langle \mathbf{X}_{ij}, \mathbf{Y}_{ij}, \mathbf{Z}_{ij}, \mathbf{U}, \mathbf{V}, \mathbf{W} \rangle$, where i and j are the indices of the 7×7 array of sampling points.

We use volume rendering to display the 3D slices on the hyperwall screens. The orderly 2D array of 3D slices pro-

vides a rather direct view of 5 dimensions of a 6D dataset – but since the sampling array is not necessarily axis-aligned we are generally seeing variation across all 6 dimensions.

For our purposes, we have several datasets requiring navigation in a 6D space. One example is the visualization of correlation holes in the electron pair density function. Given a molecule, we fix the position of an electron. Then, for a given position of the reference electron, we consider the spatial density of the remaining electrons. Since both the position of the reference electron and the spatial density field of the remaining electrons have three degrees of freedom, the correlation holes are structures in six dimensions. See figures 4 and 5.

5 Interactive Parameterized Simulations

With the advent of workstation class PCs, significant computational power is available to throw at a problem. When combined with an array of graphics displays, we achieve an environment where we can exercise computational steering[11] of *families* of related simulations.

Our system allows us to run parameterized families of simulations in parallel. However, computational steering environments often require that one 'instrument' the simulation code in order to give the user feedback (monitoring) and allow interactive control (steering). For monitoring, the simulation code is often modified to allow display of the state of the simulation. Additional modifications allow access to the simulation's runtime parameters for steering. Both of these modifications are delicate and obviously require in-depth analysis of any simulation code.

As an example, a molecular simulation code employing the reactive bond order Brenner potential has been instrumented by NASA researchers Chris Henze and Bryan Green. Every time the molecular dynamics code completes a timestep, it sends the updated atom positions to a viewer. Within the viewer, the user can interact with the simulation by selecting individual atoms and moving them around. Furthermore, using an interface on the master node, forces such as compression, extension, rotation, and shear can be applied.

Within the hyperwall environment, dozens of these simulations can be run in parallel. Figure 6 displays thumbnail snapshots from a set of carbon nanotube simulations each of which has been subjected to an extensional force interactively supplied by the user. Again, layout in the matrix implies position in this parameter space of nanotubes. Along the diagonal, from top to bottom, the nanotubes grow larger. Greater distances above and below the diagonal correspond to more twist in a clockwise or counterclockwise direction. We can see that, in general, the smaller tubes tend to break apart with the level of force applied by the user.

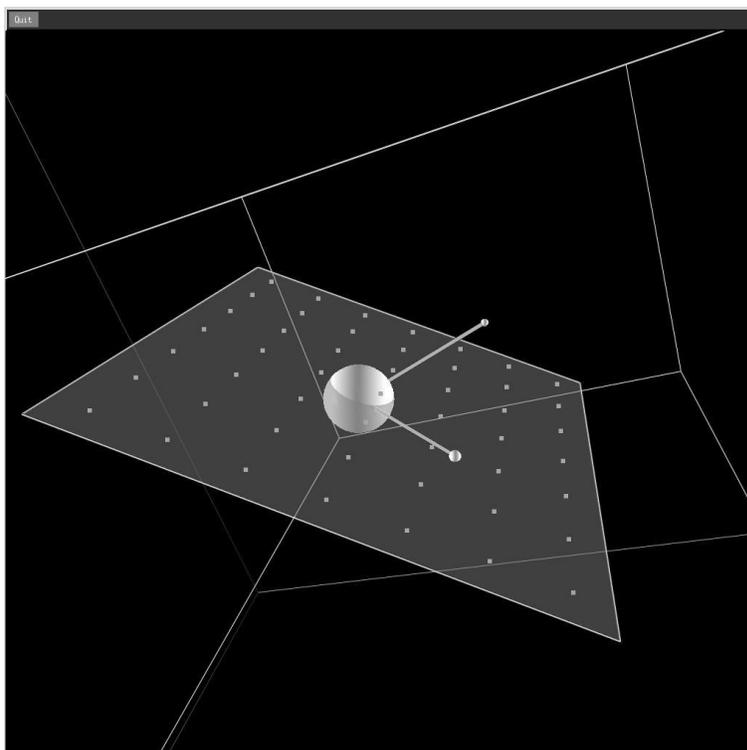


Figure 4. Six-D browser interface here shown with a schematic water molecule. This interface allows the user to coordinate the simulations seen in figure 5.

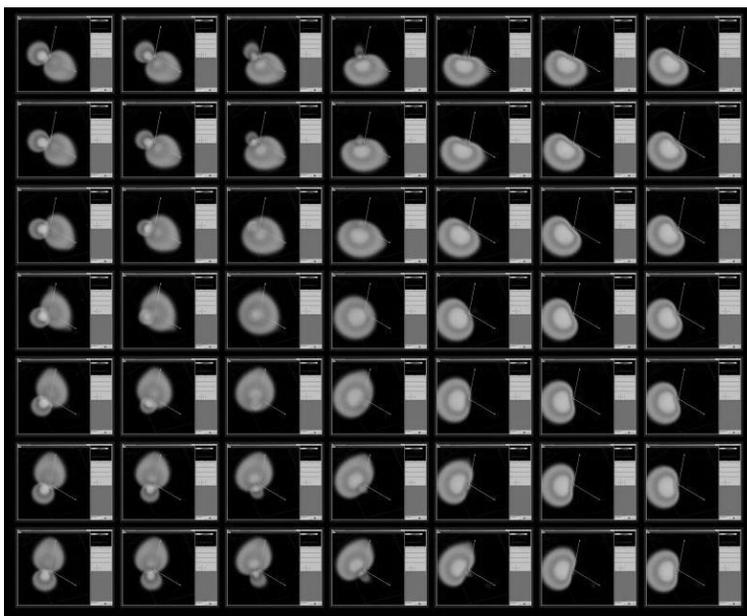


Figure 5. The Six-D browser allows interactive exploration of the electron pair density function around a water molecule.

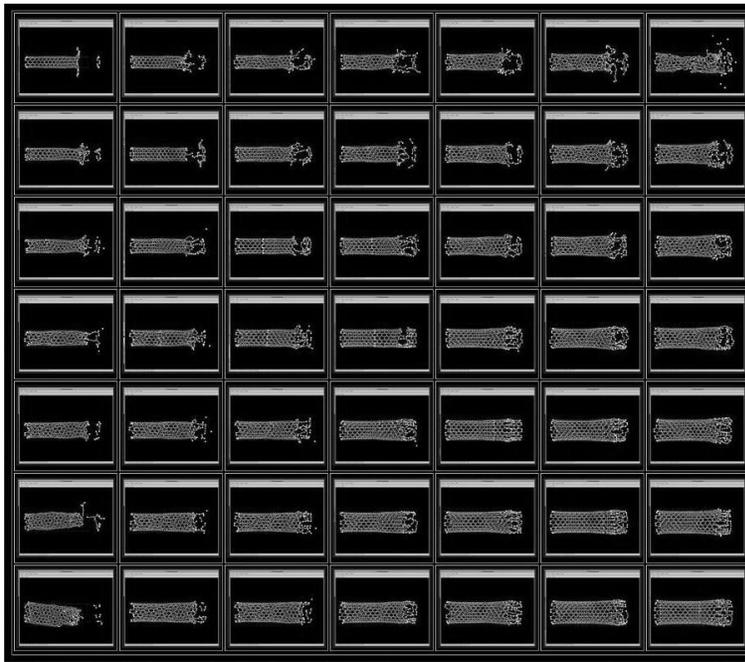


Figure 6. Realtime interaction with a family of nanotube simulations.

6 Human Factors

We have sized the hyperwall as a 7×7 array for several reasons. Some are pedestrian: the total system fit within our budget and the display array fits within our lab. The 7×7 array of 18 inch LCD monitors is a comfortable size - about the size of an office wall. From the middle of our lab (a standard size office) a user can easily fixate on any hyperwall monitor with little or no head movement. When viewed from this natural distance, the hyperwall display subtends about 80 degrees, and each pixel subtends about 0.01 degree. This nicely matches both the field of view and resolution of the human eye.

However, a somewhat more rigorous justification for the hyperwall display's dimensions (seven screens high by seven screens wide) is found in Miller's classic paper [13] that first married experimental psychology with information theory. In [13], Miller persuasively argues that seven is about the *number of classes available for absolute univariate discrimination*, or the number of objects in the span of attention, or the size of a "chunk" in human short term memory, or the depth of our stack.

Assume that each display D_{ij} on the hyperwall shows its own visualization $V_{ij} = V(\alpha_i, \beta_j)$. α and β are each parameters that change discretely in seven steps as we go from left to right, or top to bottom, respectively, across the wall-sized display array. The "magic" size, according to Miller, of a two dimensional (bivariate) chunk in human percep-

tion is 7×7 . On the hyperwall we have two dimensions to work with simultaneously, and thus our system is sized to "impedance match" this natural perceptual chunking in an obvious way.

Questions about possible arrays containing far more or far fewer displays often come up during demonstrations. Upon reflection (and after our experiences with earlier, smaller, prototypes) it seems Miller's guidance towards 7 ± 2 is correct. If the array of images is much larger than seven in either dimension, then we are unable to form a gestalt of the entire information field - we forget what we were looking at on the left as we scan across to the right. If the array is much smaller than seven in either dimension, the system's display capability falls below our maximum perceptual and short term memory capacities. Neither a 50×50 hyperwall or a 3×3 hyperwall would be a good impedance match to an individual or a small group.

There is also the obvious issue of the mullions, or frames, around the LCD displays, and the gaps they introduce. We have found that these are not a problem. Remember - the hyperwall is primarily used to display arrays of related images rather than to display a single large image. But even when displaying a single large high-resolution image, the hyperwall's frames are no more distracting than the "seams" which always rear their ugly heads in even the most carefully calibrated and aligned "seamless" multi-projector powerwall displays.

Viewing a single large high-resolution image with the

hyperwall is rather like viewing the outside world through a large, multi-paned window. And certainly, viewing multiple related images is most natural when the individual images are framed. These issues are addressed thoughtfully in the context of architecture by Alexander *et al.* [14], in a book which interestingly has had profound impact on software engineering:

When plate glass windows became possible, people thought that they would put us more directly in touch with nature. In fact, they do the opposite. They alienate us from the view. The smaller the windows are, and the smaller the panes are, the more intensely windows help connect us with what is on the other side.

This is an important paradox. The clear plate window seems as though it ought to bring nature closer to us, just because it seems to be more like an opening, more like the air. But, in fact, our contact with the view, our contact with the things we see through windows is affected by the way the window frames them. When we consider a window as an eye through which to see a view, we must recognize that it is the extent to which the window frames the view, that increases the view, increases its intensity, increases its variety, even increases the number of views we seem to see - and it is because of this that windows which are broken into smaller windows, and windows which are filled with tiny panes, put us so intimately in touch with what is on the other side. It is because they create far more frames: and it is the multitude of frames which makes the view.

Therefore:

Divide each window into small panes. These panes can be very small indeed, and should hardly ever be more than a foot square. [14]

7 Conclusion

We have described the hyperwall, a system supporting interactive exploration of MDMV data and simulations. Because we have a full blown Beowulf cluster, each node of which is armed with its own graphics card, we can run compute and graphics intensive applications, bringing a powerful array of tools to bear on many problems. This allows us to compute whole arrays of visualizations or simulations in parallel, displayed at high-resolution, in a highly interactive fashion. The sheer visual nature of the display system encourages people to scan the displays looking for trends, relationships, and anomalies. We find that scientists want to walk right up to the wall of screens, look closer, and point

out observational curiosities to co-investigators. The hyperwall is becoming a useful, high-bandwidth collaboration environment for a variety of scientific teams.

Acknowledgements

This work was sponsored by NASA contract TOA61812D. Thanks to David Ellsworth, for all the encouragement and helpful insights.

References

- [1] J. W. Tukey. "Exploratory Data Analysis," Addison-Wesley, 1977
- [2] R. A. Becker and W. S. Cleveland. "Brushing Scatterplots," *Technometrics*, 29(2), 127-142, 1987
- [3] Chris North and Ben Shneiderman. "Snap-together visualization: Coordinating multiple views to explore information." Technical Report CS-TR-4020, University of Maryland Computer Science Department, 1999
- [4] D. F. Swayne, D. Cook, A. Buja. "XGobi: Interactive Dynamic Graphics in the X Window System with a Link to S," in *ASA Proceedings of the Section on Statistical Graphics*, p. 1-8, 1991
- [5] C. Russo Dos Santos and Ken. W. Brodli (2002). "Visualizing and Investigating Multidimensional Functions." In *Proceedings of the Conference on Visualization 2002*.
- [6] J. van Wijk, R. van Liere. "Hyperslice - visualization of scalar functions of many variables," In *Proceedings of IEEE Visualization*, 1993
- [7] M. Ward, "XmdvTool: integrating multiple methods for visualizing multivariate data," In *Proceedings of IEEE Visualization 1994*, pp. 326-333
- [8] M. Levoy. "Spreadsheet for images," In *Computer Graphics (SIGGRAPH '94 Proceedings)*, volume 28, pages 139-146. SIGGRAPH, ACM Press, 1994
- [9] A. Varshney and A. Kaufman. "FINESSE: A financial information spreadsheet," In *IEEE Information Visualization Symposium*, pages 70-71, 125, 1996
- [10] Ed H. Chi, J. Riedl, P. Barry, J. Konstan. "Principles for Information Visualization Spreadsheets," In *IEEE Computer Graphics and Applications (Special Issue on Visualization)* July/August, 1998. IEEE CS, pp. 30-38.

- [11] J. Mulder, J. van Wijk, and R. van Liere. “A Survey of Computational Steering Environments,” in *Future Generation Computer Systems*, 13(6), 1998
- [12] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. Kirchner, J. T. Klosowski. “Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters,” presented at SIGGRAPH, San Antonio, Texas, 2002
- [13] G. A. Miller. “The magical number seven, plus or minus two: Some limits on our capacity for processing information,” *The Psychological Review*, 63, 81–97, 1956. Reprint available at <http://psychclassics.yorku.ca/Miller/>
- [14] C. Alexander, S. Ishikawa, M. Silverstein *A Pattern Language*. Section 239 “small panes”. Oxford University Press. 1977. see also <http://www.patternlanguage.com/apl/apl239/apl239.htm>