Eighth Eurographics Workshop on Virtual Environments (2002)
S. Müller, W. Stürzlinger (Editors)

Active Visualization in a Multidisplay Immersive Environment

W. Blanke[†] and C. Bajaj[‡]

Abstract

Building a system to actively visualize extremely large data sets on large tiled displays in a real time immersive environment involves a number of challenges. First, the system must be completely scalable to support the rendering of large data sets. Second, it must provide fast, constant frame rates regardless of user viewpoint or model orientation. Third, it must output the highest resolution imagery where it is needed. Fourth, it must have a flexible user interface to control interaction with the display. This paper presents the prototype for a system which meets all four of these criteria. It details the design of a wireless user interface in conjunction with a foveated vision application for image generation on a tiled display wall. The system emphasizes the parallel, multidisplay, and multiresolution features of the Metabuffer image composition architecture to produce interactive renderings of large data streams with fast, constant frame rates.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Graphics Systems]: Distributed/network graphics

1. Introduction

One goal of very large tiled immersive displays is to increase a wider experience. In other words, the user is able to see globally with local detail. However, as displays become larger and larger, computer graphics systems encounter a diminishing dpi (dots per inch) problem. At a certain point the system is no longer able to render enough pixels at a real time rate to ensure a high resolution, high dpi image over the entire tiled display wall.

This paper demonstrates a system which attempts to solve this problem by applying dynamic multiresolution techniques in conjunction with wireless control devices. Through the use of foveated displays along with progressive streaming to alleviate bandwidth issues, multiresolution techniques can allow real time frame rates on large tiled display walls while still achieving high dpi imagery wherever the user wishes to look, or more generally, at points of interest in the scene.

Foveated vision techniques rely on the fact that the human eye can only sense detail directly at the point of focus. Objects in the peripheral vision appear in low resolution and lack definition. This basic biological fact is a result of the concentration of rods and cones in the retina of the human eye. A higher concentration exists at the center with the density gradually becoming lower and lower towards the edges. As a consequence of human biology, even though computer visualization systems may render a large display in high resolution, by the time that information gets to the brain, much of the information has been lost by the limitations of the vision system.

Applying this fact to the large displays currently available could significantly improve frame rates and user interactivity. Already CAVE type virtual reality labs[6] employ multiple projectors for an immense immersive display. By tiling the higher resolution projectors or panels available today, creating enormous displays with billions of pixels is practical. IBM, for example, currently has a 3840 by 2400 pixel LCD panel consisting of 9.2 million pixels. Creating an 11 by 11 grid of those panels would result in a display consisting of over one billion pixels. Foveating such displays as these could greatly reduce workload and load balancing issues while increasing frame rate and user interactivity.

The active visualization system shown in this paper con-

[†] Department of Electrical and Computer Engineering, University of Texas at Austin
[‡] Department of Computer Sciences, University of Texas at Austin

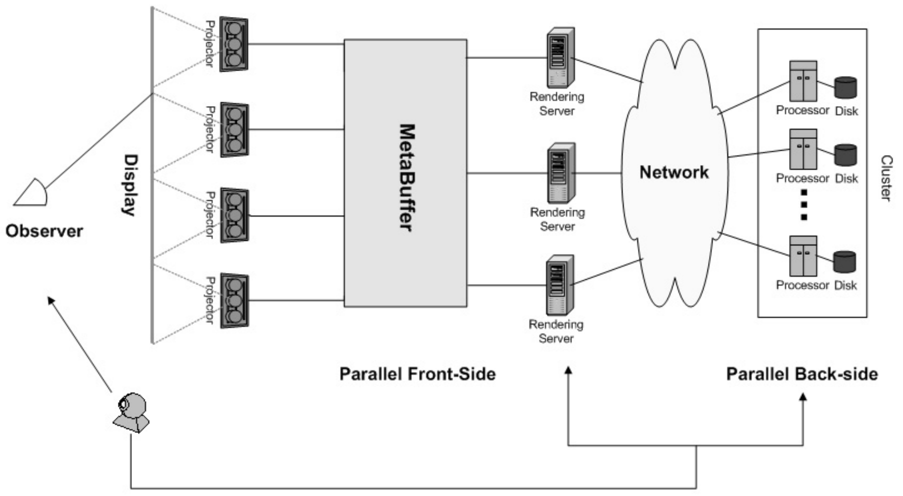

Figure 1: *Closed loop architecture for parallel time-critical rendering of massive data streams*

sists of a wireless user interface, parallel rendering servers, and an image compositing device which drives a tiled display wall. Figure 1 shows the complete closed loop system.

In order to provide the multiresolution functionality required by the foveated vision technique, the Metabuffer multiresolution sort-last image compositing architecture[4] is employed to composite image output generated by the cluster of rendering machines and then drive the tiled display wall. Many other research groups have developed sort last hardware similar to the Metabuffer in order to leverage the combined rendering power of multiple distributed rendering machines. These include the Sepia project[17, 13], PixelFlow[8], and Lightning-2[12]. The Metabuffer is most similar to Lightning-2 in that it uses a crossbar type architecture to composite image data and drive multiple displays. However, one main difference between the two is that the Metabuffer supports multiple levels of resolution which makes the foveated vision technique shown in this paper possible.

Since the Metabuffer architecture has not been deployed in hardware yet, the images and results provided in this paper have been computed by a software emulator of the Metabuffer[5] running on a cluster of standard Linux workstations. This paper shows that a system built using multiresolution techniques such as foveated vision and progressivity in conjunction with a wireless user interface helps to allow for a real time immersive environment when applied to large data sets and large tiled displays.

2. Background

Using the foveation of the human visual system as an advantage is nothing new. Several research groups have tackled problems such as image transmission and image processing by using the low resolution areas of the eye as an asset.

2.1. Image Processing

One problem that benefits from foveated techniques is image processing. Image processing is often a very computationally intensive task. Every pixel in an image must have calculations performed on it to perform pattern matching, edge detection, or other operations.

Many times, though, this image processing is being done to simulate what a normal human eye would be seeing. Facial recognition is one such example. The human eye lacks detail in its peripheral view. Therefore, the brain does not have to process nearly as much information from the edges of the view as it does in the center.

Researchers have taken advantage of this fact by using methods to avoid processing the enormous quantities of high resolution pixels in the periphery. Special foveated CCD cameras have been developed which record high resolution only at the center of the gaze in order to lessen the information overload resulting from taking in imagery from high resolution cameras which sense all areas equally[9, 23, 19]. Image processing applications can then take advantage of this reduced imagery to concentrate their algorithms on the center of the scene, rather than the edges of the gaze.

2.2. Image Transmission

Another problem which has used foveated vision is image transmission. Full motion video can require large amounts of data to be transmitted. Usually the amount of bandwidth available is the limiting factor facing this transmission. Any technique that lessens the need for data will greatly help the image transmission problem. Since the peripheral vision of the human eye cannot see high resolution imagery, it makes little sense to have to transmit this peripheral image data that eventually will not even be processed by the vision system.

This is the technique used by Geisler et al[11] and Lee et al[16]. Their research applies foveated techniques to image and video encoding. Essentially the video stream is recorded at successive levels of resolution. By recording the user's gaze, a "foveated pyramid" is created with high resolution imagery in the center which becomes successively lower the farther the imagery happens to be from the user's gaze.

2.3. Image Generation

Although not specifically tied to applications involving eye tracking, several research groups have studied using multiresolution techniques to speed up image generation. Hoppe[14] illustrates how progressive meshes can be used to significantly increase performance when rendering large data sets. He shows how different levels of detail can be used depending if the data is close or far away from the user. Shamir et al[21] reveal how to use DAGs in order to efficiently create multiresolution meshes on time varying deforming meshes. Magillo et al[18] present a library in order to model multiresolution meshes. Saito et al[20] discuss how to use wavelets to compactly encode and efficiently retrieve hierarchical multiresolution representations of objects.

Progressive meshes will be used by the Metabuffer foveated vision application to present level of detail views of the scene to the user based on gaze location. Currently, the progressive meshes used by the Metabuffer foveated vision application do not use wavelet compression, but this method could serve to compress source data further to better handle large data sets.

3. Implementation

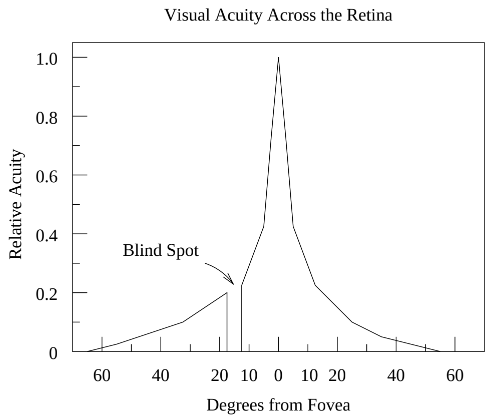

Figure 2: *Coren's acuity graph*

Acuity is the term used to describe the eye's ability to resolve detail. Typically, this measurement is expressed as an angle corresponding to the smallest span the eye can identify. As shown in figure 2 by Coren et al[7], acuity changes as a function of the distance away from the center of the eye. This is due to the concentration of rods and cones in the retina. The highest concentration exists at the center of the eye in the fovea, with the density becoming less and less towards the periphery. In the 2D slice of the eye shown in figure 2, a blind spot also exists where the optic nerve exits the eyeball.

Coren's graph reveals that the drop off in acuity, and thus resolution, in the eye is exponential. In fact, within 10 degrees it drops by almost 80 percent. By matching the rendering resources of the computer graphics system to this acuity graph, the rendering power of the system can be concentrated mainly in the areas where it is needed most–the center of the user's gaze. Only a small portion of the system is needed to generate the low level of detail and resolution towards the periphery.

A foveated vision system can be designed using Coren's graph either via the continuous method or the discrete method. The discrete method using the hardware capabilities of the Metabuffer will be covered in this paper.

3.1. Continuous Method

With the continuous method, level of detail and resolution is matched directly to Coren's acuity graph. By using a wavelet encoded mesh, it is possible to finely adjust the complexity of the scene. Depending upon the distance from the center of the user's view, an error value corresponding to Coren's graph can be used to walk through the wavelet encoding in order to obtain the proper amount of detail for every area in the scene. Likewise, this same error value can be used to adjust the level of resolution used to generate the scene. Higher error values would allow lower levels of resolution. A very similar method employing hierarchical bounding boxes[2] could also be used.

In either case, delays resulting from data locality issues could be quelled by utilizing progressivity. As with progressive image composition[3], switching to lower resolution viewports would allow renderers to cover all the polygons they are responsible for drawing while still keeping the frame rate high. Over time polygons can dynamically be moved to achieve the high resolution output imagery.

Of course, other metrics besides Coren's acuity graph could be used to direct the resolution and level of detail. In these cases, the foveated vision system dealing strictly with user gazes can instead be generalized into a region of interest (ROI) application. This region could be controlled via user input from a wireless device or other input method instead of merely being taken from gaze tracking hardware. The region of interest could also be modified by past user history–keeping previous areas of interest in focus. Another characteristic that could modify resolution and level of detail is prominent features in the data set. Algorithms could detect high frequency changes in the data set and bring those areas into closer focus since they could yield interesting information. Distance from the user is also a trait that could be used to influence the level of detail in a scene such as is done in Hoppe's work[14].

3.2. Discrete Method

Applying the discrete method to the Metabuffer hardware makes sense since the Metabuffer is able to generate viewports only in integer increments of different resolution. This is because the Metabuffer employs pixel replication to generate the low resolution viewports. Using local memory caches to replicate the pixels, instead of transmitting more data over at-capacity buses, alleviates any bandwidth issues in the compositing hardware.

Because of this limitation, instead of using Coren's complete graph as a cue for the level of resolution, individual points on that graph are taken for each Metabuffer viewport resolution multiple. These individual points are used to precompute a hierarchical mesh of the model to be used in generating the scene. Also because of the Metabuffer hardware, the high resolution area as shown in this paper is set to the size of one Metabuffer high resolution viewport. This restriction, though, could be remedied by tiling high resolution viewports together to form a larger area.

For example, in the case shown in figure 3, the foveated vision application using the Metabuffer employs three differently sized viewport. The smallest viewport contains the highest resolution and is centered at the user's focus. This area corresponds to the peak in Coren's acuity graph and will be assigned the highest level of detail data set. The next larger viewport implemented by the Metabuffer is in medium resolution. To find the level of detail for this area, the highest acuity level covered by this area in Coren's graph is used. In this case, it would be about 20% of the detail of the high resolution data set. Likewise, the largest and lowest resolution viewport implemented by the Metabuffer uses a level of detail of approximately 10% as according to Coren's graph. The polygon counts in figure 3 differ slightly from these percentages. The mesh generation algorithm currently used only roughly outputs the number of polygons requested.

With polygon counts in the medium and low resolution viewports running 20% and 10% of the polygon counts in the high resolution viewport, it is possible to match the greater number of rendering servers to the area of the user's focus. Using a cluster of rendering servers, 77% of these can be assigned to generate the imagery for the high resolution high level of detail viewport. Because the medium resolution viewport consists of only 20% of the polygons as the high resolution viewport, only 15% of the machines are needed to render this area. Finally, since the lowest resolution consists of only 10% of the polygons, only 8% of the machines are necessary to render the entire region in a load balanced manner. Because only 10 machines are being used in figure 3, the numbers of rendering servers assigned differ slightly from these stated percentages.

3.3. Load Balancing

The main problem in creating a foveated vision application for the Metabuffer system is how best to utilize the rendering resources available. They should be organized in order to achieve the best degree of efficiency and the fastest frame rates. The multiple parallel rendering machines need to be load balanced no matter what viewpoint the user chooses. This organizational problem is presented formally as follows.

Conditions

1. There exists a screen of n tiles and m rendering servers $(m \geq n)$. Each tile has the same size of $w \times h$ pixels and each server has the same rendering capability c triangles/second.
2. There are p triangles that project into the screen. We assume each triangle takes the same amount of time to render.

Constraints

1. A high resolution $w \times h$ pixel area must be rendered where the user(s) are gazing at all times. Regions surrounding this area can be rendered in diminishing level of detail and resolution corresponding to the drop off in rod and cone concentration in the peripheral view of the eye.
2. The data set could be extremely large, and thus all p triangles along with the varying levels of detail of this triangle set must be evenly distributed across all the machines. There cannot be a global data set that resides on each machines.
3. The frame rate should be at least on par with the $\frac{p}{m \times c}$ time possible with the progressive image composition method. Taking into account the diminished triangle count from decimated data sets, this means that the rendering machines need to be fairly load balanced for any user viewpoint even if the data set is almost certainly heterogeneously distributed across the scene.

The goal is not only to find the best assignment of levels of detail of data to renderers but also the best match of renderers to display space such that the display is rendered in the shortest time.

In order to solve this problem, the multiresolution features of the metabuffer will be used extensively. In the case of a single user, viewports are arranged in a configuration analogous to Geisler's "foveated pyramid". Figure 3 shows the "foveated pyramid" for the visible human example in this paper. High resolution viewports are located at the center of the user's gaze. Successively lower resolution viewports radiate out until the lowest resolution viewport fills the entire display.

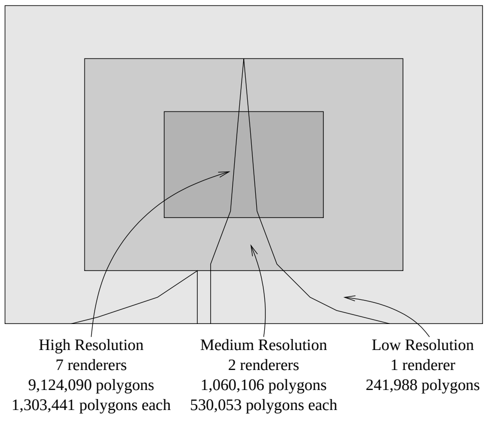

Figure 3: *Foveated pyramid for active visualization of the visible human polygonal isosurface*

The ability to concentrate the rendering power of the Metabuffer in the area of the user's gaze is possible because of progressive meshes that have been created by decimating data sets. The large low resolution viewports in the periphery are required to render a much greater area that would normally consist of a large amount of polygons. By using decimated data sets, however, the quantity of polygons in this area can be much less than the number of polygons contained in the small high resolution viewport. Therefore, a small number of rendering servers can adequately render larger area.

Ensuring that the rendering servers are load balanced despite the user's viewpoint is achieved by assigning the triangles belonging to each progressive mesh modulo the number of processors assigned to that mesh. This means that the polygons for the data set are evenly distributed in 3D space among all the processors. No matter where the user looks, all the processors will be responsible for an even number of polygons. This is the technique used by PixelFlow[8] to load balance its custom hardware even when dealing with nonhomogeneous data sets. By avoiding the assignment of clusters of close together polygons to the same rendering server, load balancing issues with changing user viewpoints are alleviated.

3.4. Compositing

In order to merge the layers of multiresolution imagery together and simulate the "foveated pyramid" using the Metabuffer, it is necessary to ensure that the higher level resolution imagery always takes precedence over lower level resolution imagery. To do this, lower resolution rendering servers remove portions of their viewports that will be covered by higher resolution imagery using the hardware stencil buffer. With most of today's graphics cards, including the GeForce2 boards in the Metabuffer cluster, stencil tests are always performed when doing Z buffer comparisons. Thus, the use of a stencil buffer is essentially free in terms of performance cost. With the areas not covered by the stencil vacant, pixels from high resolution renderers are free to be composited over these areas. This effectively performs a painters algorithm operation using the existing architecture of the Metabuffer.

To allow for continuity, neighboring viewports of different resolutions are allowed to overlap slightly. In these areas of overlap, dithering patterns are performed. Again, this is done using the stencil buffer. Checkerboard patterns are applied at the edges of the higher resolution viewport. By pushing the far and near clipping planes slightly farther back for the neighboring low resolution area, the border area between the two viewports consists of half higher and half lower resolution data, but with a checkerboard mesh that is of the higher resolution. This screen door transparency technique effectively smooths the output image at the transitions between the higher and lower resolutions. Blending this area masks discontinuities in the progressive meshes and in the resolution changes.

3.5. Wireless Control

Employing handhelds to control the visualization system opens up many interesting new research areas to explore. Many different groups have studied the user of handheld devices for user interfaces for such things as ubiquitous computing[22], augmented reality and situated information spaces[10], and context-aware applications including memory prostheses[15].

Recent advances in wireless handheld technology have rendered what used to be a complicated technical undertaking to just plugging in a collection of commercial off the shelf components. The handheld device carried by the user is a Compaq iPAQ Pocket PC. This device runs the Windows CE operating system from Microsoft. For wireless connectivity, an Orinoco RG-1000 residential gateway is employed along with Lucent wireless Ethernet cards. The wireless Ethernet cards plug into the iPAQs by means of a PCMCIA adapter. They are then configured to talk to the RG-1000 which is connected to the Metabuffer cluster's LAN. From this point, communicating over the network is seamless.

Figure 4 shows an actual screen shot of the user interface. At the top of the shot is a representation of the 5 by 2 tiled display wall. The longhorn icon is placed where the user is gazing via the stylus. For now this provides the center of the user's foveated viewing region until a retina tracker is installed in the visualization lab. The X, Y, and Z vectors shown at the lower portion of the screen are a hypervolume control as described by Bajaj et al[2]. Using a classical rotation based user interface in order to visualize data sets in higher dimensions than 3D is very tedious. Examples of such data sets include gated MRI volume scans of heart motion,

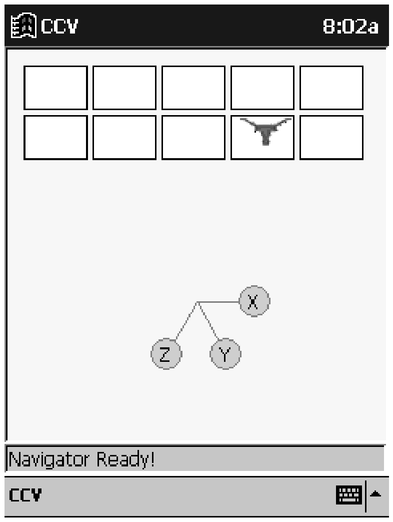

Figure 4: *Wireless visualization device user interface*

time varying data from computational fluids dynamics, and molecular vanDerWaal energies as a function of molecular configurations (bond angles). Using the hypervolume control shown here allows for controlling any number of dimensions in a scalable manner. For each dimension, only a single vector is needed. By controlling the length and relative angle of each vector, it is possible to maneuver the viewed object into any possible position in the 3-dimensional space. As shown by Bajaj et al, for simple 3D case like the one shown in figure 4 it is easy to convert these three individual vectors into a more familiar 3D projection matrix[2].

The orientation and gaze information received from the graphical UI is transmitted over the wireless Ethernet as UDP packets to a server residing on the land based host cluster. This server collects the information from all the wireless devices and stores the current state of all locally. At each frame, the Metabuffer application queries the server about the status of the wireless users. This is done via a named pipes mechanism. The server was separated from the Metabuffer application because the Metabuffer emulator uses MPI as its basis. Currently the version of MPICH running of the Metabuffer's host cluster does not support multithreading. Therefore running it as a separate process allows the Metabuffer to run unencumbered. The individual process model will also make it easier for other applications to have access to the same data.

4. Results

The configuration used to test the foveated vision application consisted of 19 machines in the visualization cluster. Each machine was equipped with a high performance Hercules Prophet II graphics card, 256 MB of RAM, an 800 MHz Pentium III processor and ran the Linux operating system. 9 of the machines were set to actually emulate the Metabuffer hardware. They performed the image compositing and output of the 3 by 3 tiled display space for the examples in this paper. The other 10 machines were tasked with actually rendering the scenes. The actual 5 by 2 display wall in the visualization lab is detailed at http://www.ticam.utexas.edu/ccv.

All 19 machines were connected via 100 Mbps Fast Ethernet. We limited the test to 19 machines instead of the full 32 in the cluster with graphics cards because the higher amounts of data transfer exceeded the capabilities of the network and significantly slowed emulator performance. We anticipate that the addition of Compaq's ServerNet II to the cluster will greatly reduce this constraint. The actual Metabuffer design, when put into hardware form, eliminates this overhead entirely.

Three data sets are used to demonstrate the capabilities of the foveated vision application for the Metabuffer: an isosurface of an engine block, a skeletal isosurface of the visible human, and a epidermal isosurface of the visible human. Both contain progressive meshes generated by the fast isosurface extraction system developed by Zhang et al[24].

Data Set (partition)	Size	Render Time
Engine (7 high res.)	88,273	0.05 seconds
Engine (2 medium res.)	23,041	0.03 seconds
Engine (1 low res.)	10,728	0.02 seconds
Engine (overall)	617,910	0.05 seconds
Skeleton (7 high res.)	907,543	0.57 seconds
Skeleton (2 medium res.)	332,264	0.17 seconds
Skeleton (1 low res.)	138,594	0.06 seconds
Skeleton (overall)	6,352,801	0.57 seconds
Human (7 high res.)	1,303,441	0.81 seconds
Human (2 medium res.)	530,053	0.28 seconds
Human (1 low res.)	241,988	0.12 seconds
Human (overall)	9,124,090	0.81 seconds

Table 1: *Foveated data set information*

The statistics for each are shown in table 1. The render timings for each data set reflect the average time needed to compute each frame in a 720 frame movie with the model being zoomed and rotated. As all data is kept locally, and polygons are assigned so that changing user viewpoints will not upset load balancing, these average timings are in fact the frame times for any point in the movie. The timings are subdivided by viewport resolution. The machines which render the high resolution area take the longest to render and thus determine the overall frame time. As these machines are always the majority (7 out of 10), the parallel efficiency of the method is reasonable.

4.1. Visible Human

In the case of the visible human data set, the highest resolution mesh consists of 9,124,090 polygons. The medium resolution mesh consists of 1,060,106 polygons. Finally the lowest resolution mesh has only 241,988 polygons. Given the processor assignments from figure 3 with the polygon counts from the progressive meshes of the visible human generated by the isosurface extraction, the high resolution mesh is divided among 7 rendering servers resulting in 1,303,441 polygons per server. The medium resolution mesh is divided between 2 rendering servers giving 530,053 polygons per server. The low resolution mesh is assigned to one rendering server which is responsible for all 241,988 polygons. At first it may seem that these assignments are imbalanced, but it is important to remember that, because the high resolution imagery will only be drawn for one area of the display, not all of the polygons assigned to the high resolution renderers will need to be drawn. This is true to a lesser degree for the medium resolution polygons too. Because the polygons for all the servers are distributed evenly across object space, different viewpoints or zooms should not affect loading.

The images in figure 5 (see color section) show 8 of the frames from a 720 frame movie. At the beginning and end of the movie, the nine separate screens in the tiled display split apart to reveal the geometry of the overall scene. In the middle of the movie they join together to show how the unified display would look.

During the movie, the visible human data set is moved through a zoom in and zoom out while being continually rotated. Meanwhile, the user's gaze is being tracked and that area is rendered in high resolution no matter what the viewpoint. The user is not restricted to where he or she may look. Anywhere in the entire display space is a valid place for the high resolution viewport.

Polygons are color coded according to which rendering server created them. This gives the imagery within the high resolution viewport a mottled appearance, since 7 rendering machines are responsible for this area. The medium resolution viewport, on the other hand, only has two colors from the two renderers that are assigned to it. Finally, the low resolution viewport is being rendered by only one machine and thus is a solid green.

Notice that the display decreases in resolution and complexity according to the "foveated pyramid" of multiresolution viewports which are marked as black rectangles. The level of detail differences in the progressive meshes and the resolution differences are most noticeable in the zoomed in views. For example, the fine detail of the lower torso of the human inside the high resolution viewport contrasts with the less detailed data set being rendered by the low resolution viewport of the leg in these views.

Timings from the movie are shown in table 1. Because the polygons are distributed evenly across the scene between the processors, all the timings from the 720 frame movie are flat. No matter where the user looks or how much he or she zooms into the scene, the load will always be the same. Because a parallel application is only as fast as its slowest component, the frame rate for this example using 10 rendering machines would be 0.81 seconds per frame. However, because of the scalable nature of the Metabuffer architecture, adding additional rendering machines only results in additional pixels worth of latency and does not affect throughput. More machines would result in smaller data partitions which should decrease rendering time and increase frame rate.

4.2. Engine

For the engine data set, the highest resolution mesh consists of 617,910 polygons. The medium resolution mesh has 46,082 polygons and the lowest resolution mesh consists of 10,728 polygons. With the processor configuration of above. this means that the high resolution mesh is partitioned into units of 88,273 polygons, the medium resolution mesh is divided into 23,041 polygons each, and the low resolution mesh is assigned to one processor responsible for 10,728 polygons. Again, the polygon distributions are not even across the resolution groups, but the majority of renderers (those rendering the high resolution area) are completely balanced in terms of polygon count. Those high resolution renderers will be the determining factor in frame timings, since they are responsible for the largest polygon counts. Thus, the minority of renderers should not adversely affect either the timings or the efficiency of the system.

Figure 6 (see color section) shows 8 of the frames from the 720 frame movie created using the engine data set. Just as with the visible human example, the data set is zoomed in and out while constantly being rotated. The region of interest controlled by the user is constantly in high resolution with the periphery falling off in detail according to Coren's model and Geisler's "foveated pyramid". Again, each viewport is color coded according to the renderer that drew it.

The timings for the engine movie are shown in table 1. Again, just as with the visible human, the timings are flat no matter what viewpoint or region of interest is chosen. In the case of the engine model, the 10 machines used in the rendering of the frames are more than enough to create 30 frames per second. Again, if this were not the case, the Metabuffer architecture is easily scalable to allow for more rendering machines which will subdivide the polygon count further and allow for faster frame times.

4.3. Skeleton

With the skeletal data set, the foveated vision application for the Metabuffer behaves just as the previous two examples. The skeletal data set consisted of 6,352,801 polygons in the high resolution viewport split over 7 processors resulting in 907,543 polygons per processor. For the medium resolution level of detail, there were 664,528 polygons split over two processors giving 332,264 polygons per processor. Finally, in the lowest resolution level of detail there were only 138,594 polygons assigned to a single machine.

Figure 7 (see color section) shows 8 of the frames from the movie made from the skeletal data set. Again, the model is zoomed in and zoomed out while being rotated. The foveated area is moved around the screen, revealing a constant area of high resolution. The rest of the display falls off in resolution as prescribed by the "foveated pyramid". As with the other two examples, each viewport is color coded according to the renderer that drew it.

The timings for the skeletal data set shown in table 1 mirror the results of the other two examples. All timings are flat regardless of the frame number. The majority of renderers are balanced and grouped in the highest time requirement. The minority of renderers responsible for the medium and low resolution areas of the screen are in the second and third highest respectively.

5. Discussion

One possible criticism of the technique as presented in this paper is that all of the rendering machines are not completely load balanced. While this is not very obvious in the case of the engine model, from the visible human results in table 1 it is evident that the timings are clumped into three groupings. The first, at 0.81 seconds per frame, are the 7 renderers that are doing the high resolution viewport. The second, at 0.28 seconds per frame, are the 2 renderers doing the medium resolution viewport. Finally, at 0.12 seconds per frame is the single renderer responsible for the low resolution viewport. While the renderers in each of these groups are load balanced among themselves, as a whole they are not evenly balanced.

This should not be a concern. The majority of rendering servers are assigned to the high resolution viewport and are load balanced among themselves. The minority of rendering servers doing the low and medium resolution viewports may not have as much work to do, but because of their small number they will not greatly erode the overall efficiency of the algorithm. As long as the workload assigned to the low and medium resolution viewports by the progressive mesh is less than the workload of the primary high resolution rendering servers, these few low and medium resolution renderers will always be faster than the high resolution renderers. Thus, this imbalance will not adversely affect the overall parallel timings.

The flat timings of the foveated vision algorithm presented here provide consistent frame rates no matter what the user viewpoint. However, these initial results should be looked on only as the worst case timings possible with this technique. Much faster timings than these could be possible with efficient frustum culling.

Even though enough data is stored on all the machines in the system to render each level of detail mesh in its entirety, obviously only a small portion of those data sets is rendered at any one frame. This is because the majority of those data sets are located outside of their area of their viewport for that particular viewing angle. To avoid rendering these polygons, it is necessary to employ a very efficient frustum culling algorithm. The frustum culler checks polygons against the boundaries of the viewing area and eliminates extraneous polygons from being sent to the OpenGL rendering stream. The more efficient the algorithm, the better the speedup the foveated vision application will achieve. Assarsson et al[1] discuss many of the methods used in fast frustum culling. Employing efficient culling would improve the overall frame rate of the system.

One issue with frustum culling is the imbalances that could exists among the different resolution viewports. For example, only machines that have a particular sized decimated data set can render polygons to the correspondingly sized viewport. In the example, this effectively means that the cluster of rendering servers has been split into three groups, a high resolution group, a medium resolution group, and a low resolution group. Because members of these groups can not easily shift to help relieve loading pressures, in some instances load imbalances will result. However, the worst case scenario is if the user is looking at a region consisting of no polygons. In that case, the majority of the rendering servers are rendering nothing. Even so, the medium and low resolution rendering servers can at most render all the polygons which they are assigned. Since the polygon count drops off exponentially this count will still be bounded to a worst case frame rate. In the case of the visible human example presented in this paper, that upper bound would be 0.28 seconds.

This paper discusses foveated vision using a single user. In order to support multiple viewers with multiple gazes, replication is necessary. Because of the modulo distribution of polygons among the rendering servers, a single distributed data set can only render one viewport area. Trying to render another viewport with that same data set would result in some polygons being unavailable. To cope with this, it is necessary to have copies of each decimated data set (except for the lowest resolution data set which covers the entire display, of course) and a set of dedicated machines for each viewer. Replication is typically not a good attribute to have when dealing with large data sets, but considering that the number of users will typically be much lower than the available machines that can render, this duplication does not present an inordinate problem for memory requirements.

Acknowledgements

This research was supported in part by grants from the National Science Foundation ACI-9982297, CCR-9988357, NPACI-UCSD-10181410, contract from the LLNL/SNL BD

4485-MOID-1, and the award from the Texas Higher Education Coordinating Board ARP-BD-781.

References

1. ASSARSSON, U., AND MÖLLER, T. Optimized view frustum culling algorithms. Tech. rep., Chalmers University of Technology, March 2000.

2. BAJAJ, C. L., PASCUCCI, V., RABBIOLO, G., AND SCHIKORE, D. R. Hypervolume visualization: A challenge in simplicity. In *IEEE Symposium on Volume Visualization* (1998), pp. 95–102.

3. BLANKE, W. *Multiresolution Techniques on a Parallel Multidisplay Multiresolution Image Compositing System*. PhD thesis, University of Texas at Austin, 2001.

4. BLANKE, W., BAJAJ, C., FUSSELL, D., AND ZHANG, X. The metabuffer: A scalable multiresolution multidisplay 3-d graphics system using commodity rendering engines. Tr2000-16, University of Texas at Austin, February 2000.

5. BLANKE, W., BAJAJ, C., ZHANG, X., AND FUSSELL, D. A cluster based emulator for multidisplay, multiresolution parallel image compositing. Tech. rep., University of Texas at Austin, April 2001.

6. C. CRUZ-NEIRA, D. J. S., AND DEFANTI, T. A. Virtual reality: The design and implementation of the cave. *Computer Graphics 27*, 4 (August 1993), 135–142.

7. COREN, S., WARD, L., AND ENNS, J. *Sensation & Perception*. Harcourt Brace, New York, NY, 1999.

8. EYLES, J., MOLNAR, S., POULTON, J., GREER, T., LASTRA, A., ENGLAND, N., AND WESTOVER, L. Pixelflow: The realization. In *Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware* (August 1997), pp. 57–68.

9. FERRARI, F., NIELSEN, J., QUESTA, P., AND SANDINI, G. Space variant imaging. *Sensor Review 15*, 2 (1995), 17–20.

10. FITZMAURICE, G. Situated information spaces and spatially aware palmtop computers. *Communications of the ACM 36*, 7 (July 1993).

11. GEISLER, W., AND PERRY, J. Variable-resolution displays for visual communication and simulation. *The Society for Information Display 30* (1999), 420–423.

12. HANRAHAN, P. Scalable graphics using commodity graphics systems. Views pi meeting, Stanford Computer Graphics Laboratory, Stanford University, May 17, 2000.

13. HEIRICH, A., AND MOLL, L. Scalable distributed visualization using off-the-shelf components. In *Parallel Visualization and Graphics Symposium – 1999* (San Francisco, California, October 1999), J. Ahrens, A. Chalmers, and H.-W. Shen, Eds.

14. HOPPE, H. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Visualization 1998* (October 1998), pp. 35–42.

15. LAMMING, M., BROWN, P., CARTER, K., ELDRIDGE, M., FLYNN, M., LOUIE, G., ROBINSON, P., AND SELLEN, A. The design of a human memory prosthesis. *The Computer Journal 37*, 3 (1994).

16. LEE, S., AND BOVIK, A. Rate control for foveated mpeg/h.263 video. In *IEEE International Conference on Image Processing* (1998).

17. LOMBEYDA, S., MOLL, L., SHAND, M., BREEN, D., AND HEIRICH, A. Scalable interactive volume rendering using off-the-shelf components. In *Proceedings of IEEE 2001 Symposium on Parallel and Large-Data Visu alization and Graphics* (2001), IEEE Computer Society Press, pp. 115–121.

18. MAGILLO, P., FLORIANI, L. D., AND PUPPO, E. A dimension and application-independent library for multiresolution geometric modeling. Tech. Rep. DISI-TR-00-11, University of Genova, Italy, 2000.

19. PARDO, F., AND MARTINUZZI, E. Hardware environment for a retinal ccd visual sensor. In *EU-HCM SMART Workshop: Semi-autonomous Monitoring and Robotics Technologies* (April 1994).

20. SAITO, N., AND BEYLKIN, G. Multiresolution representations using the auto-correlation functions of compactly supported wavelets. *IEEE Transactions on Signal Processing 41* (December 1993), 3584–3590.

21. SHAMIR, A., PASCUCCI, V., AND BAJAJ, C. Multiresolution dynamic meshes with arbitrary deformations. In *IEEE Visualization Conference 2000* (2000), pp. 423–430.

22. WEISER, M. Some computer science issues in ubiquitous computing. *Communications of the ACM 36*, 7 (July 1993), 65–84.

23. WODNICKI, R., ROBERTS, G., AND LEVINE, M. A foveated image sensor in standard cmos technology. In *Custom Integrated Circuits Conference* (1995).

24. ZHANG, X., BAJAJ, C., AND BLANKE, W. Scalable isosurface visualization of massive datasets on cots clusters. In *Proceedings of IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics* (2001), IEEE Computer Society Press, pp. 51–58.

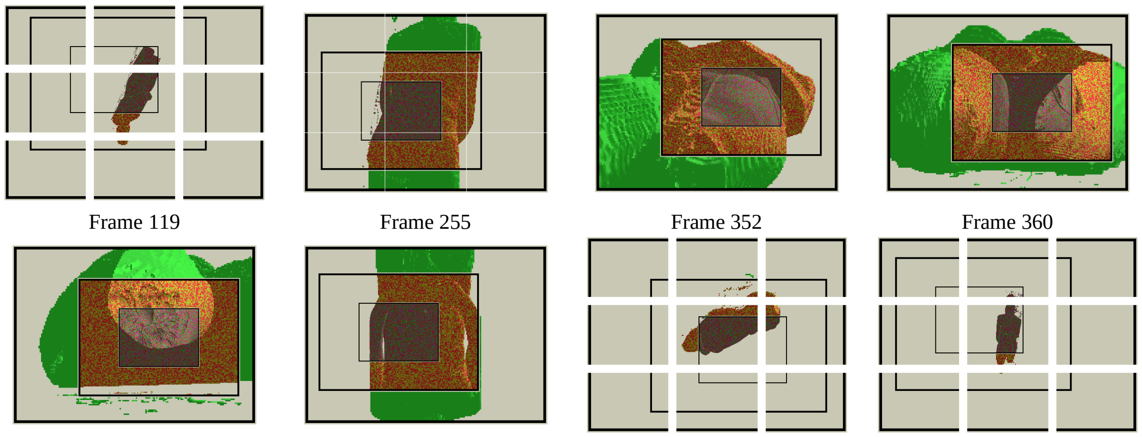

Figure 5: *Sample frames from the visible human movie*

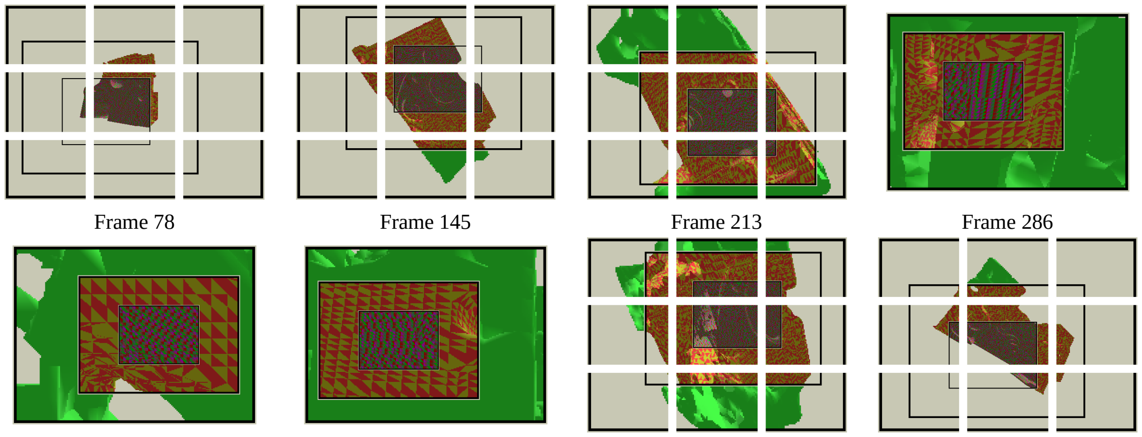

Figure 6: *Sample frames from the engine movie*

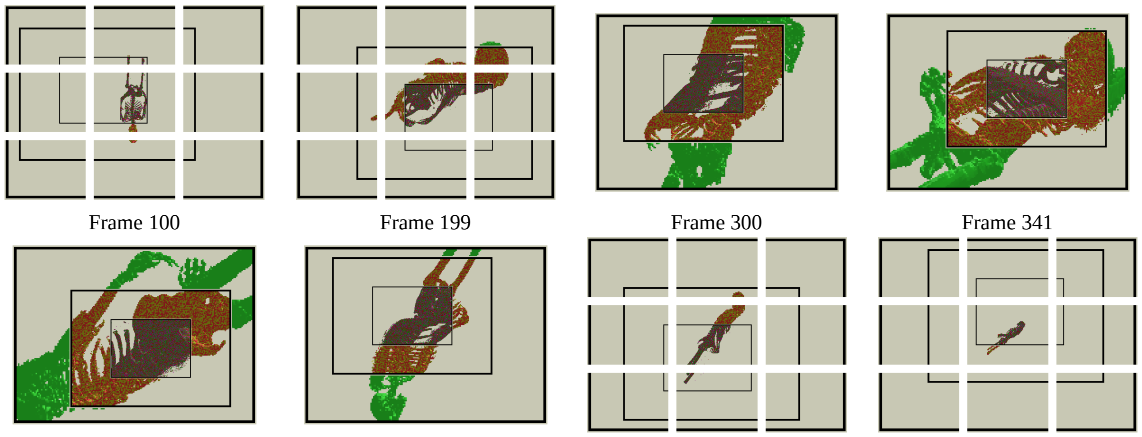

Figure 7: *Sample frames from the skeleton movie*