PERGAMON

Computers & Graphics 27 (2003) 681–691

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

Graphics hardware

Active visualization in a multidisplay immersive environment

William J. Blanke[a], Chandrajit Bajaj[b,*]

[a] Department of Electrical and Computer Engineering, University of Texas at Austin, USA

[b] Center for Computational Visualization, Department of Computer Sciences and Institute of Computational and Engineering Sciences, University of Texas at Austin, TX 78712, USA

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 two different multiresolution techniques—foveated vision and progressive image composition—to generate images on a tiled display wall. The system emphasizes the parallel, multidisplay, and multiresolution features of the Metabuffer image composition hardware architecture to produce interactive renderings of large data streams with fast, constant frame rates.

Keywords: Parallel rendering; Metabuffer; Multiresolution; Progressive image composition; Foveated vision

1. Introduction

Today imaging and simulations are increasingly yielding larger and larger data streams. These data sets can range in size from gigabytes to terabytes of information. In many cases, such data sets are much too large to store and render on a single machine. Viewing these large data sets poses yet another problem. In some cases, the detail allowed by a single high performance monitor may not be adequate for the resolution required.

To cope with these issues, many systems have been designed which use parallel computation and tiled screen displays. Dividing the data set among a number of computers reduces its enormous bulk to more reasonably sized chunks that can be quickly rendered. Likewise, using tiled displays results in a larger amount of display space. Small details that might be culled out on a single monitor can be spotted in an immersive visualization laboratory with hundreds of square feet of screen space. Such approaches have been used since the 1960s in single-display systems [1–6]. More recent work includes the PixelFlow [7], Sepia [8], and AIST [9] systems. Multiple display systems, which are the focus of this paper, include Lightning-2 [10], the Metabuffer [11], the Princeton project [12], and Chromium [13].

Most current parallel, multidisplay systems have common problems in regards to user interactivity. Because they usually depend on data locality in some form (dividing the data set evenly among the processors), changing the view-point of the user can often wreck any careful load balancing done on the data set. An unevenly load balanced data set will significantly degrade the frame rate which a user experiences. Even worse, in some cases if the tiled displays are linked only to certain machines, large quantities of data or pixels may need to be moved immediately simply to render the frame correctly. This can result in a significant delay to the user. Also, large tiled displays require immense

*Corresponding author. Tel.: +1-512-471-8870; fax: +1-512-471-0982.

E-mail address: bajaj@cs.utexas.edu (C. Bajaj).

0097-8493/$ - see front matter © 2003 Published by Elsevier Ltd.
doi:10.1016/S0097-8493(03)00141-9

amounts of computing resources to render. This is despite the fact that, in most cases, much of the display is either not in the user's view or is only within the user's peripheral vision. Current parallel, multidisplay systems are limited in how they can allocate their computing resources to cope with a partially viewed scene in order to accelerate the possible frame rate.

By using multiresolution techniques, such as those supported by the Metabuffer hardware architecture, these data locality and resource allocation problems can often be alleviated in parallel multidisplay systems that render interactive large-scale data streams. This paper shows how the Metabuffer system, coupled with a flexible user interface, uses multiresolution techniques to provide an essential balance between display quality and frame rate in an active visualization environment.

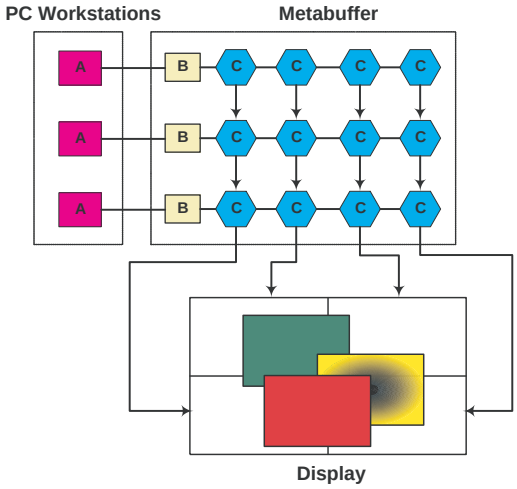

Fig. 1. Metabuffer architecture.

2. Metabuffer architecture

The Metabuffer [11] hardware supports a scalable number of PCs and an independently scalable number of displays—there is no a priori correspondence between the number of renderers and the number of displays to be used. It also allows any renderer to be responsible for any axis-aligned rectangular viewport within the global display space at each frame. Such viewports can be modified on a frame-by-frame basis, can overlap the boundaries of display tiles and each other arbitrarily, and can vary in size up to the size of the global display space. Thus, each machine in the network is given equal access to all parts of the display space, and the overall screen is treated as a uniform display space, that is, as though it were driven via a single, large frame buffer, hence the name Metabuffer.

Because the viewports can vary in size, the system supports multiresolution rendering, for instance allowing a single machine to render a background at low resolution, while other machines render foreground objects at much higher resolution. Also, because the Metabuffer supports supersampling, antialiasing is possible as well as transparency using the screen door method.

Fig. 1 shows a Metabuffer architecture using three rendering engines and four output displays utilizing multiple pipelined data paths and busses as interconnects. External to the board, commercial off-the-shelf (COTS) rendering engines (A) deliver their data to on-board frame buffers (B) by means of the recently adopted industry standards for digital video transmission, the digital visual interface (DVI). Since COTS rendering engines (A), at this time, transfer only 24 bits per pixel over these digital links, color is transferred on even frames, while alpha and Z information is transferred on odd frames. At a refresh rate of 60 Hz, this is still fast enough to provide enough RGB, alpha and Z information for 30 frames/s. The on-board frame buffer (B) stores information from both transmissions in memory. Control information, such as the location of the viewports and their final destination in the overall display, is stored on the first scan line of each rendering engine's image (A). This first scan line is never displayed. Instead, DSP code, viewport data, or anything else that is needed by the control logic of the Metabuffer can be written here using standard OpenGL glDrawPixels() calls.

When a full frame has been buffered, data are selectively sent over a wide bus to the composer units (C) based on viewport locations. The composers (C) take only the data that are required to build their column's output image and ignore the rest. Each composer (C) then sends its data in pipeline fashion down the column to the next lower composer (C) so that the pixel Z-order information can be compared with those Z values from the other COTS renderers (A). This way, only the front-most pixel is saved. The collaged data are then stored on another on-board frame buffer. These smart frame buffers can perform post processing on the data for anti-aliasing and are also able to drive the off-board displays again using the DVI specification.

Because the composers compute their data in a pipelined manner, adding more COTS rendering engines to a system only results in more latency–the overall throughput of the system is not affected. However, since this latency is measured in pixels per rendering engine (the number of pixels depending on the width of the data pipeline), even a Metabuffer consisting of 1000 rendering engines would only be penalized at most one or two scan lines worth of latency. The overriding latency problem is during the buffering of the COTS image to the on-board Metabuffer frame buffer. This results in a

full frame of latency because of the need for random access within the imagery.

Although the Metabuffer architecture is very similar to the Lightning-2 system, there are several differences. The Lightning-2 system reorganizes the data on the video card before sending it over the DVI port to avoid the buffering latency mentioned above, but this can affect throughput by tying up the video card. Also, the Metabuffer supports multiresolution image compositing while there is currently no such feature on Lightning-2 (although this could be added to their architecture as well).

2.1. Data flow analysis

The Metabuffer supports multiresolution techniques by employing viewports of varying size and position. It is important to demonstrate that the bandwidth requirements of the composers will not exceed the limited data rate of the bus that connects them to the input frame buffers. If the bandwidth requirements are exceeded in certain viewport configurations, glitches in the output image are certain to occur. The analysis that follows proves that the Metabuffer has a constant bandwidth requirement regardless of the size or orientation of the viewports that are used.

In order to analyze the worst case data flow of the board, a scheme is used similar to the one presented in the paper by Kettler et al. [14]. Since all data needs are periodic (because of the raster display), each task (display) can be described in terms of the amount of data needed (C), its period (T), and its deadline (D). By quantifying these values for a sample case, it is easy to see that the bandwidth requirements do not change as the viewport geometry becomes more complex.

In Fig. 2, the example viewport is four times as large as the image generated by the COTS rendering engines and overlaps the displays generated by nine composers. The numbers superimposed over the figure show how the area of the nine displays covered by the viewport on the left, when rearranged, form four distinct screens of data on the right. Thus, when rasterized, these four screens of data will simultaneously need to be placed on the bus at the same time. However, because the ratio of pixels is 1:4 (because of the lowered resolution), there is one-fourth the bandwidth requirement and buffering can solve this problem of multiple bus accesses.

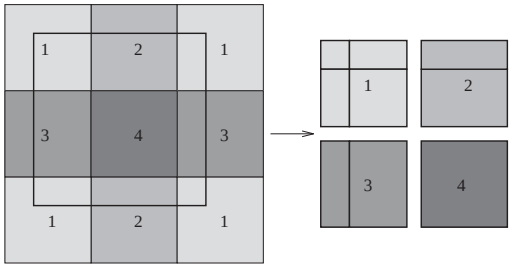

Fig. 2. Nine screen low-resolution viewport.

Table 1
Bandwidth analysis

Data	Period	Deadline
$C_{1a} = l/2$	$T_{1a} = 2w$	$D_{1a} = 2w$
$C_{1b} = (w - l)/2$	$T_{1b} = 2w$	$D_{1b} = 2w$
$C_2 = w/2$	$T_2 = 2w$	$D_2 = 2w$
$C_{3a} = l/2$	$T_{3a} = 2w$	$D_{3a} = 2w$
$C_{3b} = (w - l)/2$	$T_{3b} = 2w$	$D_{3b} = 2w$
$C_4 = w/2$	$T_4 = 2w$	$D_4 = 2w$

The same numbers that demarked Fig. 2 are also used in Table 1 to show the contributions of the individual image segments. The variable w refers to the width of a display. The variable l, which cancels out in the resulting equations, is used to delineate the vertical break between segments 1a and b and the vertical break between segments 3a and b. The horizontal break seen in segments 1 and 2 only changes the destination of the data and does not affect the scheduling of the bus for a rasterized display, and so is ignored for the purpose of these calculations.

As shown in Table 1, because pixels are being replicated to twice their size, the period (T) of the scheduling increases by a factor of 2 because there are half as many rows to process. Likewise, the data needed (C) decrease by a factor of 2. If all of the C values are totaled, the result is $2w$, which is the same as the period.

2.2. Buffering the bus

Although the bandwidth requirements remain constant across the generation of the display, at many times, multiple composers must simultaneously have data. Because of this fact, supplying a local buffer on each composer is necessary to allow for simultaneous access of the image data.

The more interesting fact about creating local buffers on the composers is that it is then possible to do multiresolution pixel replication. The buffer that each composer maintains closely resembles a queue, except for one important difference. While the buffer acts in a FIFO manner when there is no replication (the source pixels and destination pixels are in a 1:1 ratio), if pixel replication needs to be done, it is necessary to remember data from the previous row. If advanced smoothing is being performed then multiple rows may be needed. Therefore, the cache behaves like a queue, but also has a moving window of data that always stores the previous row.

In order to send data to the local buffers on the composers in a simple, yet good performing manner, an idle recovery slot allocation (IRSA) round robin approach [14] is employed which distributes data to the composers evenly based on the amount of data needed (C), the period (T), and the deadline (D). No effort is made to look ahead in the geometry of the viewports to find the most efficient way to send the data out. However, because of the previous discussion, the uniformity of the data transmitted to each buffer will result in few delays using this simple method.

3. Metabuffer output

Fig. 3 shows the three original images that were rendered for this example of Metabuffer output: a ball, a tube, and finally a seascape. The fourth and final diagram illustrates how these images were distributed using the Metabuffer to the four output displays by being broken up into viewports. Note that every image is sent to at least two output displays. As discussed earlier in this paper, the location and geometry of the viewports are arbitrary. The bandwidth requirements over the bus remains constant.

Running the three images into a three input frame buffer by four output frame buffer Metabuffer yields the four output screens in Fig. 4. Note that the tube resides in four separate displays, despite being rendered on a single machine. Also, see how the seascape here is being used as a low-resolution background display with the higher-resolution foreground images layered on top. Finally, the Z order of the input images is always taken into account, whether that means that the ball is in front of the tube, or that the ocean surface laps at the base of the foreground objects.

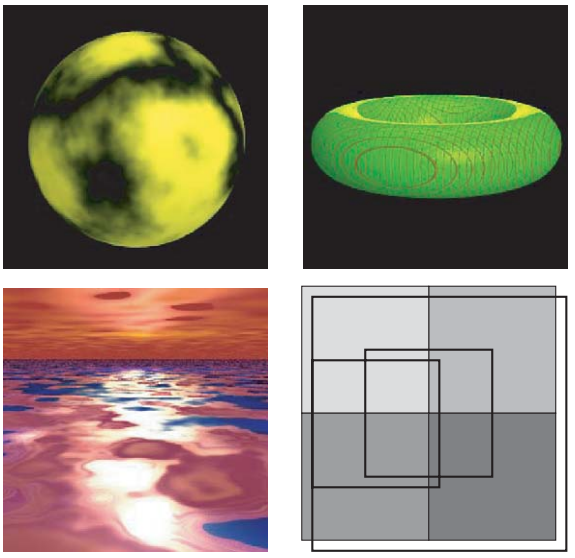

Fig. 3. Three input images with viewport configuration.

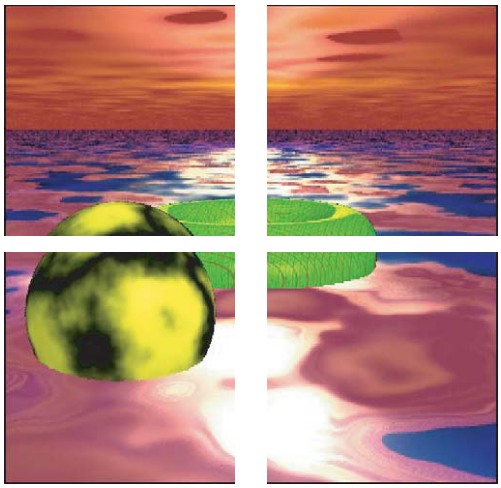

Fig. 4. Composited output images.

3.1. Metabuffer antialiasing

One problem with compositing separate images like the ones above is the aliasing that results on the edges. A solution that has been implemented on the Metabuffer involves supersampling. Simply increasing the detail of the input images and then having the output frame buffers average the pixel values down to the original size effectively smoothes the image. Only the problem pixels at the edges are affected. The rest of the composited image pixels remain as sharp as on the original.

This technique is commonly used in graphics cards to antialias displays. It is extremely simple, since the only major change to the graphics pipeline, besides the increase in resolution, is an averaging step at the very end. The main disadvantage is the fact that the graphics hardware has to run so much faster in order to generate the extra pixels. This is not much of an issue inside the tightly coupled hardware of a graphics card. In a more loosely coupled system like a cluster, these heightened bandwidth requirements could be a problem. But, even with the bandwidth concerns, supersampling has been implemented in PixelFlow [7], another sort last system similar to the Metabuffer.

The two images generated by the Metabuffer in Fig. 5 (magnified eight times to show the difference in detail) demonstrate the effect supersampling has on the resulting image quality. On the left, no supersampling has been performed. There is a jagged transition between the different input images at the Z buffer transition. On the right, the input images were rendered to be four times as detailed and the final output pixels

Fig. 5. Zoomed image without (left) and with (right) antialiasing.

were averaged by the output frame buffer from the four nearest pixels that traveled through the composer pipeline. The jagged transition is now much smoother while the rest of the image has lost no quality.

3.2. Metabuffer transparency

Sort-last image compositing systems, such as the Metabuffer, suffer from many known problems with order-dependent rendering semantics such as stencil buffers, alpha-blending, and transparency. The screen door transparency method was implemented on the Metabuffer primarily because of the flexibility it gives regarding the ordering of the compositing pipeline. Unlike Sepia [8], with its configurable ServerNet II network, the Metabuffer's pipeline is fixed in hardware. But since the screen door algorithm requires no polygon sorting, changing user viewpoints will not require shuffling the data set and, thus, will not adversely affect the frame rate.

Another advantage of the screen door algorithm is that the Metabuffer system already uses pixel replication for multiresolution and employs supersampling for antialiasing. This abundance of redundant pixels makes it quite easy to create screen door masks without affecting the quality of the image. For instance, on non-supersampled viewports, each pixel is replicated four times and then averaged down to one pixel on the final display. By employing a simple checkerboard mask on the replicated pixels, the averaged output pixel correctly achieves a 50% transmission coefficient. An example using this method is shown in Fig. 6.

The screen door technique is not without its problems. Because the Metabuffer only employs 4× supersampling, transparency can only be quantized into four levels. Also, if multiple transparent layers of polygons overlap, the screen door patterns may interfere with each other creating undesirable effects. Fig. 7 is a zoom of Fig. 6 showing how the ball completely obscures the tube behind it as a result of these mask collisions. In addition, performing the screen door mask on replicated pixels will produce problems if polygons from different machines interleave, since only the front-most Z values for each machine's viewport are recorded. However, if these limitations are taken into account, screen door is an adequate way to achieve transparency.

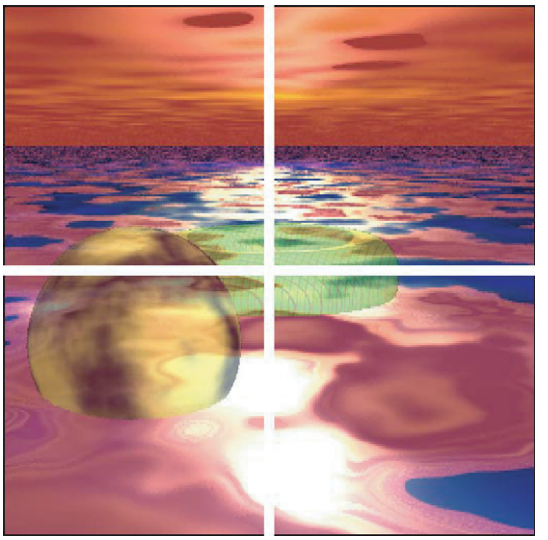

Fig. 6. Screen door transparency Metabuffer output.

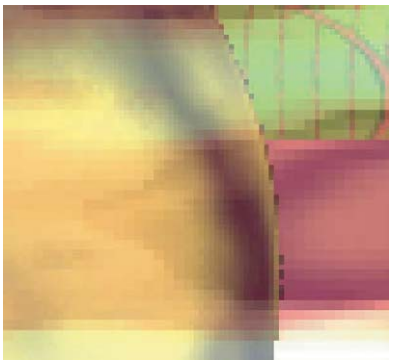

Fig. 7. Zoom of transparency example.

4. Multiresolution techniques

Multiresolution techniques, either dealing with object-space (polygon count) or image-space (resolution), have been used frequently for frame rate control [15]. Two multiresolution techniques in the context of active visualization were studied using the Metabuffer framework: progressive image composition and foveated vision [16].

The configuration used to test these two techniques consisted of 19 machines in our 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. Nine of the machines were set to actually emulate the Metabuffer hardware. They performed the image compositing and output of the 3×3 tiled display space. The other 10 machines were tasked with actually rendering the scenes.

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.

4.1. Progressive image composition

How to quickly navigate a large data set while still retaining high-quality image output is a problem for parallel rendering on multiple displays. To achieve good user interactivity, an application must guarantee time-critical rendering of the massive data stream. However, for the instance of displaying a triangular mesh, though a good load balanced partition among the parallel machines can be computed for a given user view point, new computation and data shuffling are required whenever the view point is significantly changed. Either triangles may fall out of the viewport because of the movement of the viewing direction or the viewport cannot cover all the polygons assigned to it because of zooming. Redistributing primitives or imagelets in order to render all of the polygons correctly takes time. If the user is simply navigating the data set, this additional time will result in slower frame rates hampering user interactivity.

To solve this problem, we propose adapting the concept of progressivity to the generation of images via image compositing on the Metabuffer, terming the technique progressive image composition. By employing the Metabuffer's multiresolution feature, it is possible to ensure the user will always have constant frame rates no matter what the viewing angle or zoom factor. Instead of redistributing polygons or imagelets while the user is rapidly changing views, a viewport can instead go to a lower resolution and enlarge in order to accommodate the current polygons assigned locally to the machine. When the user finally arrives at the view of interest and stops changing viewpoints, frame rate is no longer a concern. At this point, polygons are redistributed in order to once again form completely high-resolution viewports.

The example data set used in this paper to test the progressive image composition technique is an isosurface generated by Zhang [17] from the visible human model. This data set consists of 9,128,798 polygons split into 10 partitions of 912,880 polygons each using a greedy polygon to viewport allocation algorithm [16].

To demonstrate that the frame rates do not change regardless of the user's viewpoint a 720 frame movie was generated in which the data set was zoomed in and out while constantly being rotated. A sample of the frames taken throughout the movie is included in Fig. 8. All the movies presented in this paper can be seen in their entirety on the world wide web at http://www.ices.utexas.edu/ccv/projects/DiDi/Metabuffer.htm.

At the beginning of the movie, the image is cleaved into the nine tiles that form the 3×3 tiled display space. During the movie, these tiles are rejoined to show the overall display, and then separated again at the end to reinforce the fact that the Metabuffer is acting on a multitiled display space.

The black boxes visible in the frames show the viewport locations. As the data set is zoomed in and out, it is readily apparent when the viewports shift from high to low resolution by the sizes of these black boxes.

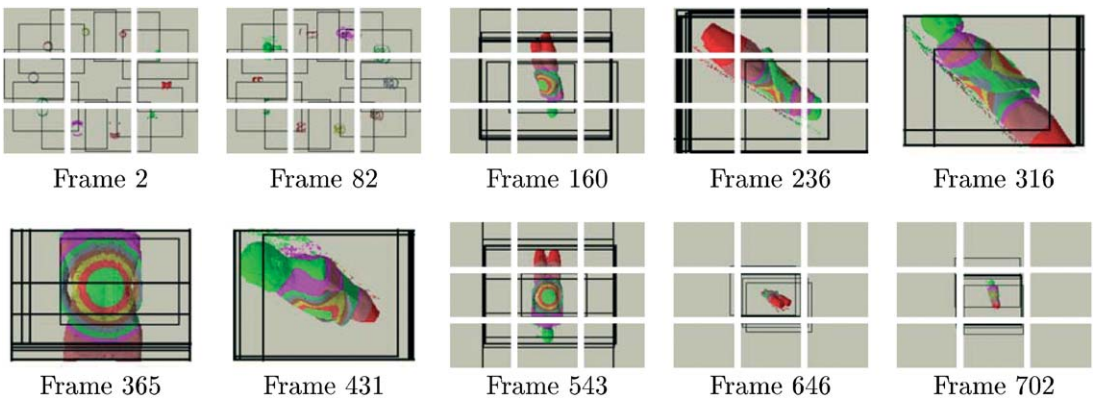

Fig. 8. Sample frames from the progressive image composition movie using the isosurface from the visible human data set.

Initially, the individual viewports belonging to each renderer are cycled around in a circle to demonstrate that they can be located anywhere within the global display space and are indeed disparate. Each viewport is color coded according to the renderer that drew it.

Later, the viewports are composited together to form the data set. The user zooms in while rotating the scene. As this is occurring, viewports dynamically move and resize themselves to adjust to the expanding extent they must cover to render all of their triangles. Finally, the user zooms out and the viewports shrink.

Note that the timings shown in Fig. 9 for rendering the viewports for each frame are almost completely flat. No communication between rendering machines has to occur between frames, and this lack of communication overhead means that the user sees no drop in interactivity regardless of how the data set is viewed.

From the graph, it is evident that the renderers are not completely load balanced. Currently, the greedy polygon assignment algorithm balances viewports in terms of polygon count. In the case of the visible human isosurface, this metric was not sufficient to evenly load the rendering machines. Other data sets fare much better, but including better metrics in the polygon assignment algorithm to more accurately measure rendering time would improve the load balancing of the visible human and other similar data sets.

All of the frames for this movie were created for a 3×3 display to facilitate an easier presentation of them for this article. In reality, the cluster hosting the Metabuffer is connected to a 5×2 tiled display space in our visualization laboratory. Typically, 10 machines are used to do the Metabuffer emulation, each responsible for driving one of the displays. The composited visible human is pictured in Fig. 10 from our visualization laboratory during a test run.

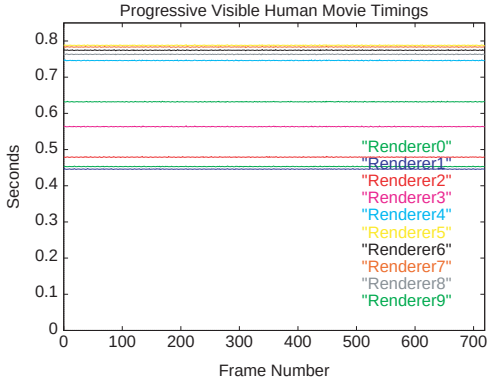

Fig. 9. Rendering times for the frames from the progressive image composition movie using the isosurface from the visible human data set.

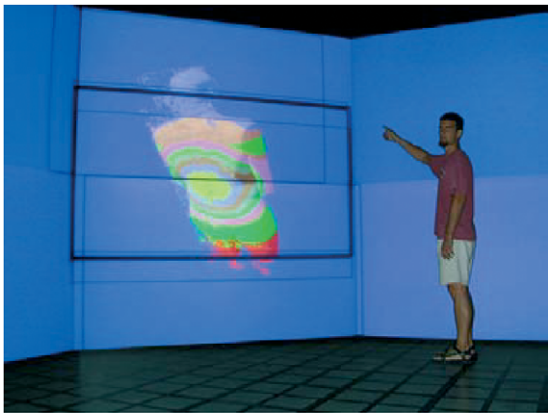

Fig. 10. Composited visible human in visualization lab.

4.2. Foveated vision

Rendering to multiple panel displays in high resolution to visualize extremely large data sets uses a tremendous amount of computing resources, takes a large amount of time and, thus, results in slow frame rates. This despite the fact that, because of our limited vision systems, much of the display either will not be seen at all (because it is behind us in a cave arrangement) or only in the periphery in low resolution.

Using the physical characteristics of the eye as an advantage, the foveated vision multiresolution technique matches the computing resources of the Metabuffer to the areas in the display that are being examined [18]. The majority of the rendering servers concentrate their work where the user is gazing. In this manner, a high-resolution image is generated quickly and exactly where the user is focused. The periphery of the display is rendered in lower and lower levels of resolution and detail corresponding to the rod/cone concentration in the human eye. Assuming the frame rate of the system is not fill limited, this allows only a few renderers to be used to create the entire periphery of what could be a building-sized display. This fact is what provides the speed gains of the algorithm. While the gazes of multiple users could be tracked resulting in multiple foveated viewpoints, this would lessen the low-resolution periphery. Thus, the foveated vision approach usually only works well in single-viewer environments.

Decimated data sets coupled with variable sized viewports means that rendering servers can be concentrated at the user's gaze. In the example presented in this paper with a 3×3 tiled display and 10 renderers, seven renderers deal with the high-resolution viewing area, two deal with the next larger area, and one works with the lowest-resolution viewport covering the entire display. The data set with the highest level of detail is divided evenly among the seven machines. The middle level of

detail data set is divided between the two. Finally, the lowest level of detail data set is given to the one machine which is responsible for the extreme periphery. This even division means that large data sets can be easily used in the Metabuffer system. The large amount of memory on the cluster as a whole is used collectively to store the polygon count.

In the case of the visible human data set (the same data set used in the progressive image composition example), 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 above with the polygon counts from the progressive meshes of the visible human generated by the isosurface extraction, the high-resolution mesh is divided among seven rendering servers resulting in 1,303,441 polygons per server. The medium-resolution mesh is divided between two rendering servers giving 530,053 polygons/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. Therefore, the larger number of polygons assigned to the high-resolution rendering servers are not necessarily the number that will be required to be drawn, given a good frustum culling algorithm.

Even so, efficiency must still be considered. The important fact to remember is that the rendering servers that are responsible for the high-resolution area are in the majority and that these rendering servers will always be balanced among themselves. Because the polygons for all the servers are distributed evenly across object space, different viewpoints or zooms should not affect loading. On the other hand, the rendering servers doing the medium- and low-resolution areas, while not fully loaded, will not adversely affect parallel efficiency. This is because, first, they are in the minority and, second, they will always finish their work before the high-resolution rendering servers thus guaranteeing that they will not undermine the overall frame time.

The images in Fig. 11 show 10 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 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 seven 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" [19] 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 (such as frame 352). For example, the fine detail of the lower

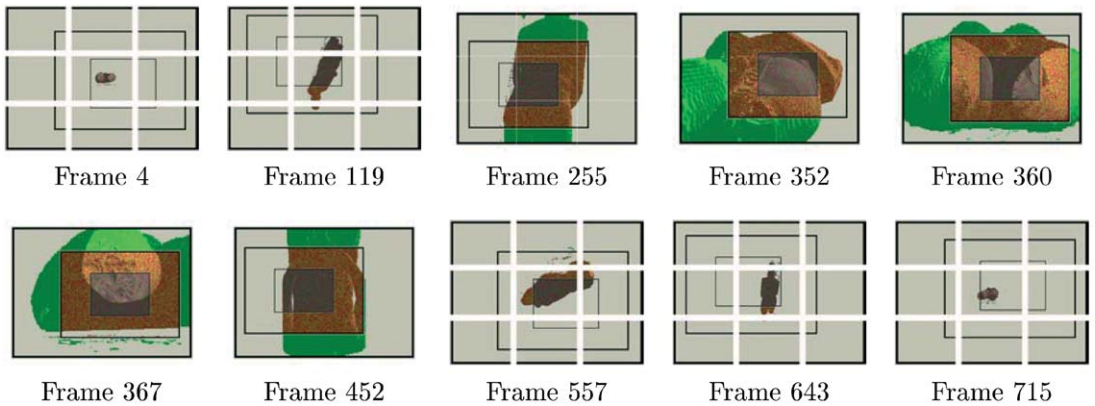

Fig. 11. Sample frames from the foveated vision movie using the isosurface from the visible human data set.

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 Fig. 12. Because the polygons are distributed evenly across the scene between the processors, all the timing lines from the 720 frame movie are flat (frustum culling has not yet been applied to this technique). No matter where the user looks or how much he or she zooms into the scene, the load will always be the same.

A parallel application is only as fast as its slowest component, so the frame rate for this example using 10 rendering machines would be 0.81 s/frame. That is the speed at which the seven high-resolution rendering servers draw their viewports. The second grouping of timings is the two medium-resolution rendering servers at 0.28 s/frame. Finally, the sole low-resolution rendering server takes 0.11 s/frame to render the background. Again, because the three non-high-resolution rendering servers are in the minority, the lower loads should not adversely affect the overall parallel efficiency of the system.

The timings shown in these examples are not real time. 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. By applying 100 machines to render the same examples, the data set would be further reduced by a factor of 10 and so would the rendering times. More rendering machines would result in similar increases in frame rate.

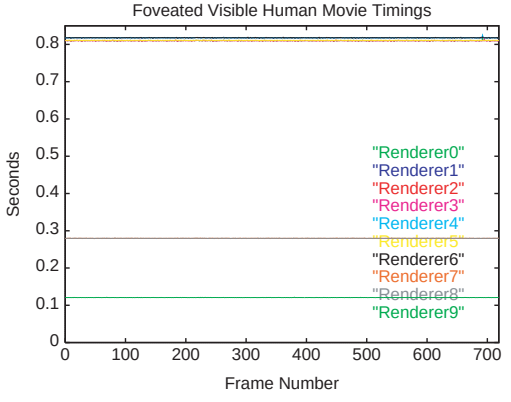

Fig. 12. Rendering times for the frames from the foveated vision movie using the isosurface from the visible human data set. The timing at 0.1 s is from the low-resolution rendering server, the timings at 0.3 s are from the two medium-resolution rendering servers, and the timings at 0.8 s are from the seven high-resolution rendering servers.

5. Wireless user interface

Employing handhelds to control the visualization system opens up many interesting new research areas to explore. Many different groups have studied the use of handheld devices for user interfaces for such things as ubiquitous computing [20], augmented reality and situated information spaces [21], and context-aware applications including memory prostheses [22].

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.

Fig. 13 shows an actual screen shot of the user interface. At the top of the shot is a representation of the 5×2 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

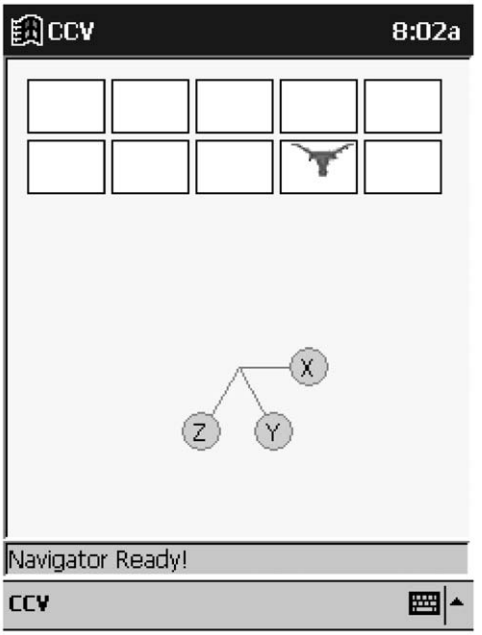

Fig. 13. Wireless visualization device user interface.

Bajaj et al. [23]. 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, time varying data from computational fluids dynamics, and molecular van Der Waal 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 three-dimensional space. As shown by Bajaj et al., for simple 3D case like the one shown in Fig. 13 it is easy to convert these three individual vectors into a more familiar 3D projection matrix [23].

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.

6. Conclusion

High-resolution imagery and frame rates are usually a tradeoff in most visualization applications. High resolution requires more computation time yielding slower frame rates. Low resolution requires less computing power and gives a faster display, but the image quality is not as good. A primary issue is managing the balance between high resolution and frame rate in order to provide the best interactivity for the user. In an active visualization system, multiresolution, in conjunction with the appropriate user interface, can manage this balance effectively resulting in higher levels of user interactivity than possible with other systems that do not exploit this feature.

The Metabuffer's implementation of multiresolution viewports in not entirely seamless. Because the bandwidth requirements of the Metabuffer must be even throughout the entire rasterization of the display, there are limitations on the sizes of the viewports that can be used. Viewports can only be of integer multiples of resolution (twice as large with half the detail, or three times as large with one third the detail, etc.). This is because the bandwidth needs are lessened by using pixel replication on the composer nodes. If the resolution multiples are not integer values, pixel replication will not be as effective and this will mean higher bandwidth needs that in some cases may swamp the bus. Also, if a viewport is in low resolution, it can only be positioned on a location that is a multiple of that resolution. Again, this is to assist pixel replication.

Pixel replication in general is another limitation of the Metabuffer. While fast and simple, it yields blockiness at very low resolutions. To achieve higher quality low resolution images some form of linear interpolation should be used to smooth the replicated pixels. This would add greatly to the complexity of the Metabuffer hardware but would not be impossible to add.

The Metabuffer requires non-trivial custom hardware in order to implement. In this regard, the Princeton project [12] which uses a standard cluster and the Sepia project which uses COTS components would be easier to deploy. However, Lightning-2 shares the same basic architecture as the Metabuffer. By reprogramming the compositing nodes of this board to have an on-board cache and do pixel replication it should be possible to yield the multiresolution features of the Metabuffer.

The benefits of multiresolution techniques vary in usefulness. Certainly for the cases in which image quality is paramount, multiresolution techniques will not be a valid option. However, for situations in which user interactivity is an overriding concern, such as active visualization, and rendering loads are large because of data set size or complexity, multiresolution techniques such as progressive image composition and foveated vision do provide fast, consistent frame rates when used in the context of a parallel, multidisplay image compositing system such as the Metabuffer.

Acknowledgements

This research was supported in part by grants from the National Science Foundation ACI-9982297, CCR-9988357, NPACI-UCSD-10181410, contract from LLNL/SNL BD 4485-MOID-1, and the award from the Texas Higher Education Coordinating Board ARP-BD-781.

References

[1] Bunker M, Economy R. Evolution of GE CIG systems, SCSD Document. 1989.
[2] Fussell DS, Rathi BD. A vlsi-oriented architecture for real-time raster display of shaded polygons. In: Graphics Interface '82. May 1982.

[3] Molnar SE. Combining z-buffer engines for higher-speed rendering. In: Proceedings of the 1988 Eurographics Workshop on Graphics Hardware, Eurographics Seminars, Nice, France, 1988. p. 171–82.
[4] Molnar SE. Image composition architectures for real-time image generation. Ph.D. dissertation, Technical Report tr91-046, University of North Carolina, 1991.
[5] Shaw CD, Green M, Schaeffer J. A vlsi architecture for image composition. In: Proceedings of the 1988 Eurographics Workshop on Graphics Hardware, Eurographics Seminars, Nice, France, 1988. p. 183–99.
[6] Weinberg R. Parallel processing image synthesis and antialiasing. Computer Graphics 1981;15(3):55–61.
[7] Eyles J, Molnar S, Poulton J, Greer T, Lastra A, England N, Westover L. Pixelflow: the realization. In: Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware, Los Angeles, CA, 1997. p. 57–68.
[8] Heirich A, Moll L. Scalable distributed visualization using off-the-shelf components. In: Ahrens J, Chalmers A, Shen H-W, editors. Parallel Visualization and Graphics Symposium—1999, San Francisco, CA, 1999.
[9] Muraki S, Ogata M, Ma K-L, Koshizuka K, Kajihara K, Liu X, Nagano Y, Shimokawa K. Next-generation visual supercomputing using pc clusters with volume graphics hardware devices. In: Supercomputing 2001, Denver, CO, 2001.
[10] Stoll G, Eldridge M, Patterson D, Webb A, Berman S, Levy R, Caywood C, Taveira M, Hunt S, Hanrahan P. Lightning-2: a high performance display subsystem for pc clusters. In: Proceedings of SIGGRAPH '2001, Los Angeles, CA, 2001.
[11] Blanke W, Bajaj C, Fussell D, 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.
[12] Samanta R, Zheng J, Funkhouser T, Li K, Singh JP. Load balancing for multi-projector rendering systems. In: SIGGRAPH/Eurographics Workshop on Graphics Hardware, Los Angeles, CA, 1999.
[13] Humphreys G, Hanrahan P. A distributed graphics system for large tiled displays. In: Proceedings of IEEE Visualization Conference, San Francisco, CA, 1999. p. 215–23.
[14] Kettler KA, Lehoczky JP, Strosnider JK. Modeling bus scheduling policies for real-time systems. In: Proceedings of 16th IEEE Real-Time System Symposium, Silver Springer, MD: IEEE Computer Society Press, 1995. p. 242–53.
[15] Hoppe H. Smooth view-dependent level-of-detail control and its application to terrain rendering. In: IEEE Visualization 1998, Research Triangle Park, NC, 1998. p. 35–42.
[16] Blanke W. Multiresolution techniques on a parallel multidisplay multiresolution image compositing system. Ph.D. thesis, University of Texas at Austin, 2001.
[17] Zhang X, Bajaj C, 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. Silver Spring, MD: IEEE Computer Society Press; 2001. p. 51–8.
[18] Blanke W, Bajaj C. Active visualization in a multidisplay immersive environment. In: Eighth Eurographics Workshop on Virtual Environments, Barcelona, Spain, 2002. p. 103–11.
[19] Geisler W, Perry J. Variable-resolution displays for visual communication and simulation. The Society for Information Display 1999;30:420–3.
[20] Weiser M. Some computer science issues in ubiquitous computing. Communications of the ACM 1993;36(7):65–84.
[21] Fitzmaurice G. Situated information spaces and spatially aware palmtop computers. Communications of the ACM 1993;36(7):39–49.
[22] Lamming M, Brown P, Carter K, Eldridge M, Flynn M, Louie G, Robinson P, Sellen A. The design of a human memory prosthesis. The Computer Journal 1994;37(3):153–63.
[23] Bajaj CL, Pascucci V, Rabbiolo G, Schikore DR. Hypervolume visualization: a challenge in simplicity. In: IEEE Symposium on Volume Visualization, Research Triangle Park, NC, 1998. p. 95–102.