# Chapter 2

# Giga-Stack: A Method for Visualizing Giga-pixel Layered Imagery on Massively Tiled Displays

## 2.1 Introduction

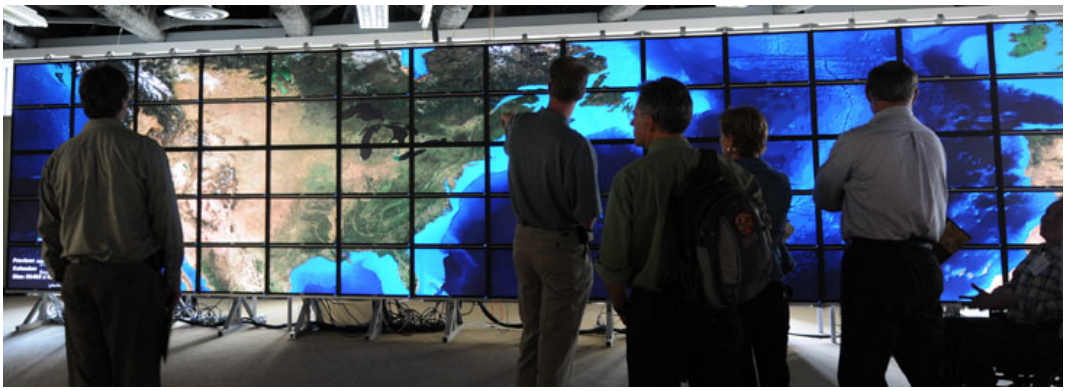

**Figure 2.1**: Users viewing a time varying version of NASAs 44 giga pixel Blue Marble Next Generation data set [SVS+05] on a 286 million pixel resolution display

The goal of the presented research in this chapter is to enable a paradigm

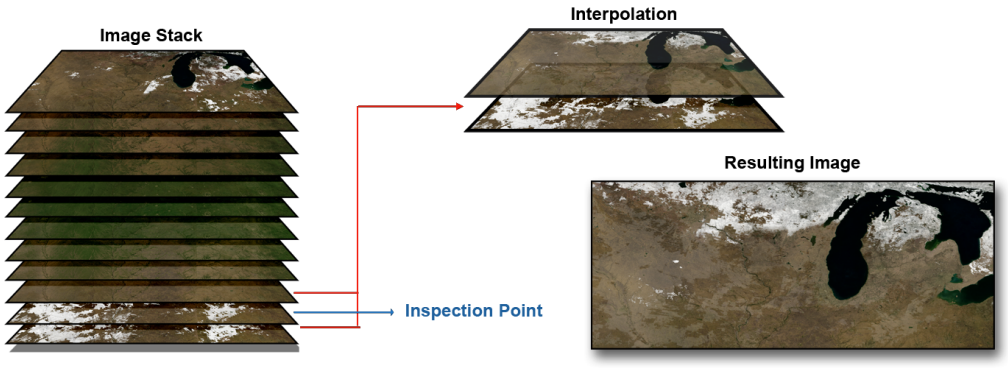


**Figure 2.2**: A conceptual diagram of the system, illustrated for the Blue Marble Next Generation data set, consisting of 12 distinct layers. The inspection point signifies which images will be interpolated to create the final visual.

where users can seamlessly zoom and translate through large, multi-dimensional image data sets, allowing users to inspect differences between time-varying and/or multi-spectral data layers at a visual complexity of hundreds of mega-pixels at a time, as shown in Figures 2.1 and 2.2. For this process to be effective, data loading must be swift, transparent and scalable. With many data sets being excessively large, making it impractical for them to be fully loaded into main memory, a tunable, resource aware management scheme is needed. The Blue Marble Next Generation data [SVS+05] was selected as a case study to demonstrate the challenges at hand.

The Blue Marble Next Generation data set consists of cloud-free satellite images taken during each of the twelve months of 2004. The images were created using NASAs Terra MODerate resolution Imaging Spectroradiometer (MODIS). Each image is 3.7 giga-pixels in size (86,400 by 43,200 pixels in dimension), with a resolution of 500 m per pixel in length. With over 44 giga-pixels worth of information, in-core approaches for data analysis as well as local data replication, are undesirable.

Several techniques are presented that allow interactive analysis of massive image sources on multi-tile displays, including tiled pyramidal image represen-

tations, global texture pools, smart replacement schemes, hardware shaders for boundary condition management and software-level synchronization.

## 2.2 Massive Tiled Displays

While current displays are limited to approximately 4 mega-pixels, large-scale data sets are orders of magnitudes higher resolution. To analyze a data set such as Blue Marble Next Generation, users could only view one one-thousandth of the data at any given time on conventional hardware, either by viewing a massively down sampled version or a tiny fragment of the overall image at native resolution. One approach towards increasing the visual real estate, is to tile displays together, as shown by the OptiPortal project [SBdL09], [DDS+09], [DLR+09], [SW06], and [CT09]. The added advantage of these systems is that the associated render and display cluster provides significant compute cycles that can be used to analyze hundreds of mega pixels worth of data simultaneously. OptIPortal-type systems have various display to node configurations, most commonly ranging from a one-to-one, two-to-one and all the way to four-to-one mapping. For the methods described in this Chapter, the assumption will be a suboptimal scenario, with each node driving a quad display configuration with a total of 16 mega-pixels worth of display real estate. As a back-end, the tiled display system uses CGLX (Cross-Platform Cluster Graphic Library) [DK10] as the middleware for communication between display nodes and synchronization of the display context.

In order to determine the appropriate position to display image layers within the tiled display environment, information about the arrangement and position of each display tile in the visualization grid, can be requested from CGLX. CGLX in turn determines the correct projection and transformation matrices needed to create a continuous visual, compensating for any bezels that may exist. Since CGLX allows applications to run natively on each of the rendering nodes, shaders can be directly added to the display loop. CGLX also provides support for the synchronization of user events, such as mouse and keyboard I/O, which are propagated reliably and efficiently throughout the tiled display environment.

## 2.3 Resource Management

Since brute-force data loading is neither practical nor desirable at the gigapixel scale, a resource management system is used to control loading, display, and replacement of image data (see Figure 2.3).

### 2.3.1 Tiling Images

Image layers are first broken up into tiles, a method also used in [FAJ07], [KUDC07], and [Sre08] By tiling the image, sections of the image can be laid out sequentially in memory, allowing sub-sections of the image to be loaded without massive cache penalties. Tiling images also allows for pre-generation of tiles containing different resolutions, analogous to mip-mapping [Wil83a]. This in turn supports a resource-aware approach coupled with the actual display device, which has a clearly defined native resolution. In other words, only information that will actually be mapped to the existing physical pixel real estate will be loaded.

A lookup table is created for each resolution of the image and each entry in the table contains pointers to locations to texture memory for a given tile. At startup this table entry defaults to zero, indicating that initially data for this tile does not exist in RAM or texture memory. When data for this tile is requested from disk, a texture pointer finds an open or stale section of pre-allocated texture memory while the data is loaded into RAM. Once the data is loaded into main memory, the table entry is changed to point to the proper main memory address. Whenever the drawing thread encounters a texture that is ready to be uploaded, it locks the section of RAM and uploads the data onto the GPU. Once data has been uploaded, the table entry changes to the OpenGL texture identification number and the RAM section is unlocked. Consequently, when this tile is drawn, this texture identification number simply needs to be referenced. When a new texture is ready to occupy this spot, the table entry is set back to zero and a new texture is uploaded.

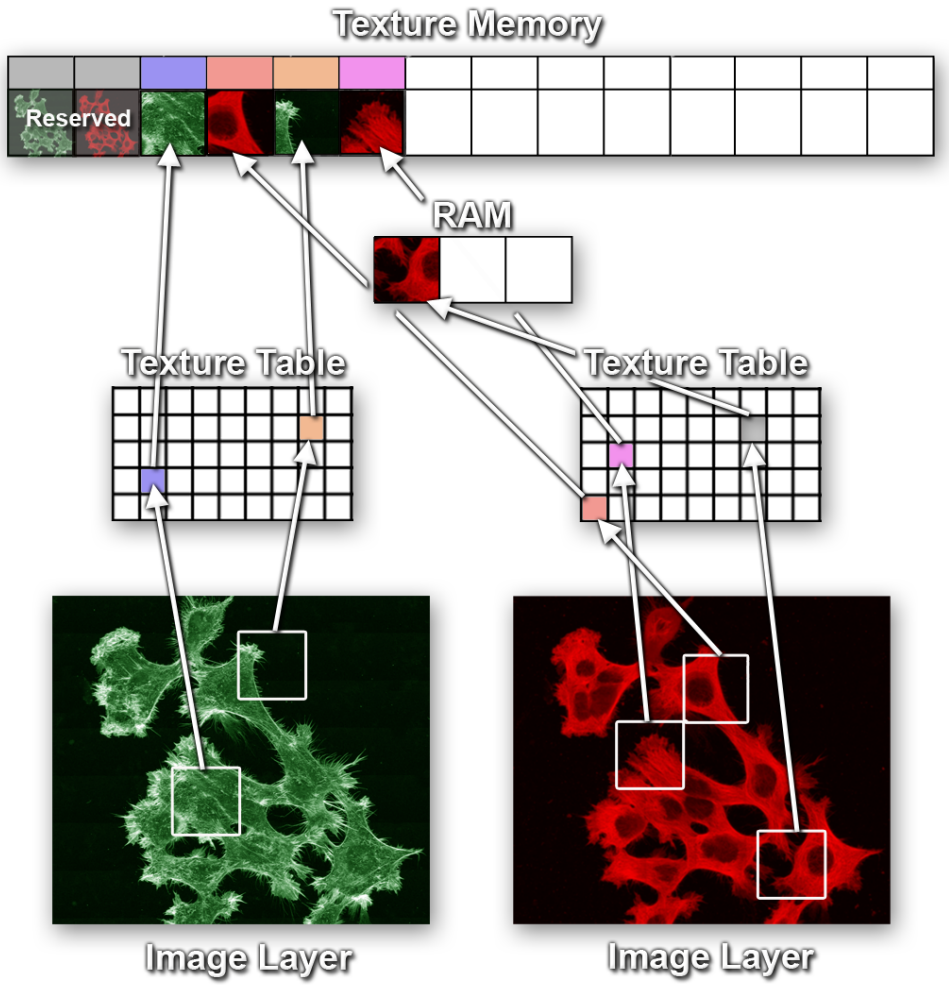


**Figure 2.3**: Illustration of resource management strategy for the example of two image layers that are being processed. Each image layer contains its own texture table which references how texture tiles are mapped to memory. The image layer on the right contains a texture tile that has been loaded from disk into RAM, but has not yet been transferred to the GPU texture memory. The global texture pool also contains references back to the locations in the texture table as shown by the colors in the diagram.

### 2.3.2 Replacement Scheme

Giga-Stack uses its own data replacement scheme to operate within a tunable memory footprint, suitable for the used hardware. A natural approach is the use of a round robin pointer that proceeds around the global texture pool, replacing the textures in the order added. However, this method needs an improvement to guarantee that tiles that are being currently shown on screen are not replaced. This will potentially occur when first panning in one direction and then reversing the opposite direction, causing the tiles currently being displayed to be at the top of the replacement list.

One solution would be to sort tiles either by the "distance" a tile is from current screen space or by the amount of time since a tile was last used. Unfortunately, sorting a list takes worst-case $O(nlog(n))$ time. It is far more important for the replacement algorithm in the presented application to simply guarantee that a tile that is replaced is not currently on screen, rather than producing an optimal result.

A second method was created which used a semi-sorted list. Whenever a tile was drawn, it was moved from its current position in the list to the front of the list. Based on this approach, textures at the back of the list would be known to be stale and, even though significantly faster than the sorted list approach, a time penalty is incurred when manipulating the list.

A modified round robin technique, as shown in Figure 2.4, was finally used. Every time a texture tile is drawn on screen, the current frame number is stored inside of the tile container. When a new texture spot is requested, the pointer moves through the global texture pool until a stale texture is found. The worst case for this method would be for the pointer to increment through the number of tiles that could be displayed on screen, but this is rare due to the order in which tiles are loaded into memory. On average, the method was able to find an open spot in less than two hops. The results section below charts the effectiveness of each of these approaches.

### 2.3.3 Texture Loading

As stated above, loading the entire giga-pixel image layer into texture memory is an unfeasible solution. Applications that use texture compression to load the entire data set in memory prove to only slightly extend the size of data sets that they can load. When dealing with giga-pixels worth of information, out-of-core techniques are preferable, fetching data.

To make these data sets load as quickly as possible, the data load stage is multi-threaded, allowing data to be loaded, processed, rendered and analyzed concurrently. The caveat is that the loading thread does not have an OpenGL context in which the data can be uploaded to the GPU. This means that the loading thread first maps data to main memory from where the render thread uploads it to the GPU.

The texture management pipeline is illustrated in Figure 2.4. First the loading thread determines which tiles have to be uploaded for the corresponding nodes viewpoint at the appropriate resolution. Each tile in the list of tiles to load is first checked to verify that it is not already loaded in the GPU or in RAM. The loading thread reads the tile from disk and loads it into RAM. Once the tile is loaded, the lookup table is changed to add a placement pointer to the tiles location in RAM. As stated previously, when the display loop is ready to draw, it checks to see if a tile is in the GPU, in RAM, or is not yet available. If the tile is in RAM, it is uploaded to the GPU using pixel-buffer-objects, to increase overall transmission speed.

Unfortunately, this upload may still take a relatively large amount of time during one draw cycle, degenerating frame rates and harming interactivity. This is especially noticeable when doing slow pans while simultaneously moving the inspection point through layered data. To mitigate this problem, the drawing thread only allows data uploads for short periods of time (i.e. 100 microseconds). After that point, any remaining data is loaded during the subsequent draw cycle(s). This approach proves to be useful for maintaining a smooth frame rate but increases the time from request to upload for a given tile. A watchdog timer is activated on periodic intervals in order to check if new tiles need to be added to the list. During

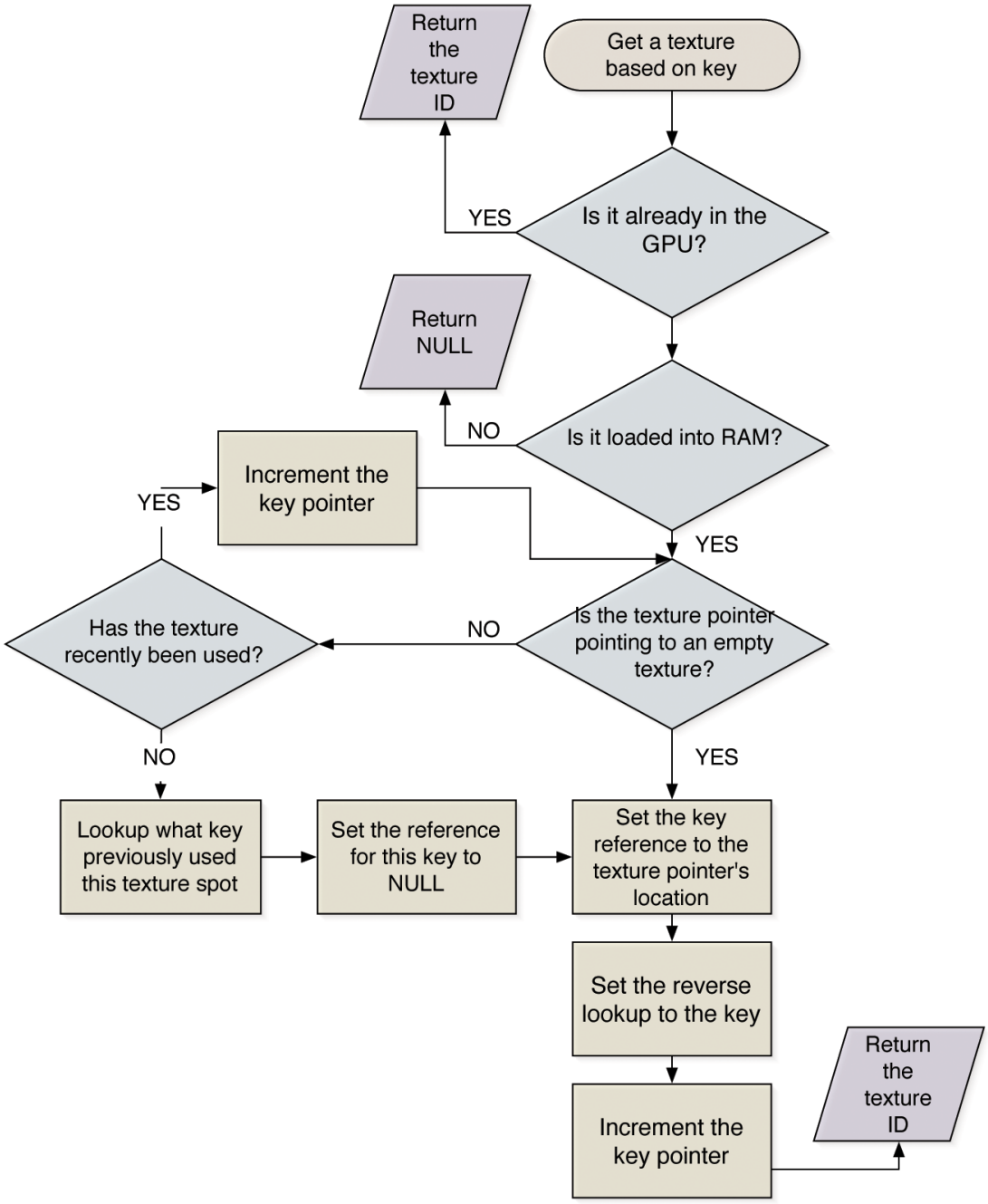


**Figure 2.4**: Flowchart of texture uploading

these periods, tiles that are no longer being used are also removed from the list.

### 2.3.4 Preview Loading

There is still a chance that the loading thread might not be able to load all desired information to the screen per display cycle. This problem is rather benign when using a single display, but is greatly exacerbated when viewing complex content on a tiled display system.

An example illustrating these additional challenges is demonstrated in Figure 2.5 for the processes of zooming into an image on a tiled display. On a single display, the image being zoomed in on can simply be super-sampled while loading higher resolution data. The result will be a slightly blurry image, which then appears to sharpen when the higher resolution is finally loaded. Unfortunately, on a tiled display system, this method no longer works because the center point of the zoom is no longer occurring in screen space, but is taking place somewhere outside of the nodes viewing frustum. The result is that much more information must be loaded, as tiles must be loaded to fill in gaps at the current resolution while at the same time loading data for the desired higher resolution representation.

The set-up for viewing images on the tiled display system also adds extra challenges for the image loading process itself. For the case-study-scenario, each computer utilizes four thirty-inch display tiles operating at 2,560x1,600 pixels resolution each, for a combined resolution of over 16 mega-pixels, which each node has to load to satisfy native resolution pixel-to-pixel mapping. Since the objective is to support continuous transformations on the data (translation, zoom), images are thresholded, such that they support a size range from 0.75 times smaller to 1.5 times larger than the actual image dimensions. This means that, for the worst case scenario, each node will have to load slightly more than 21 mega-pixels to fill the quad-display-tile configuration. Since the objective is to analyze and compare multi-spectral and/or temporal data using two images at a time, 43 mega-pixels worth of data must be loaded per node and transferred to the GPU to fully render a given viewpoint.

The challenge is that this massive amount of data has to be loaded without

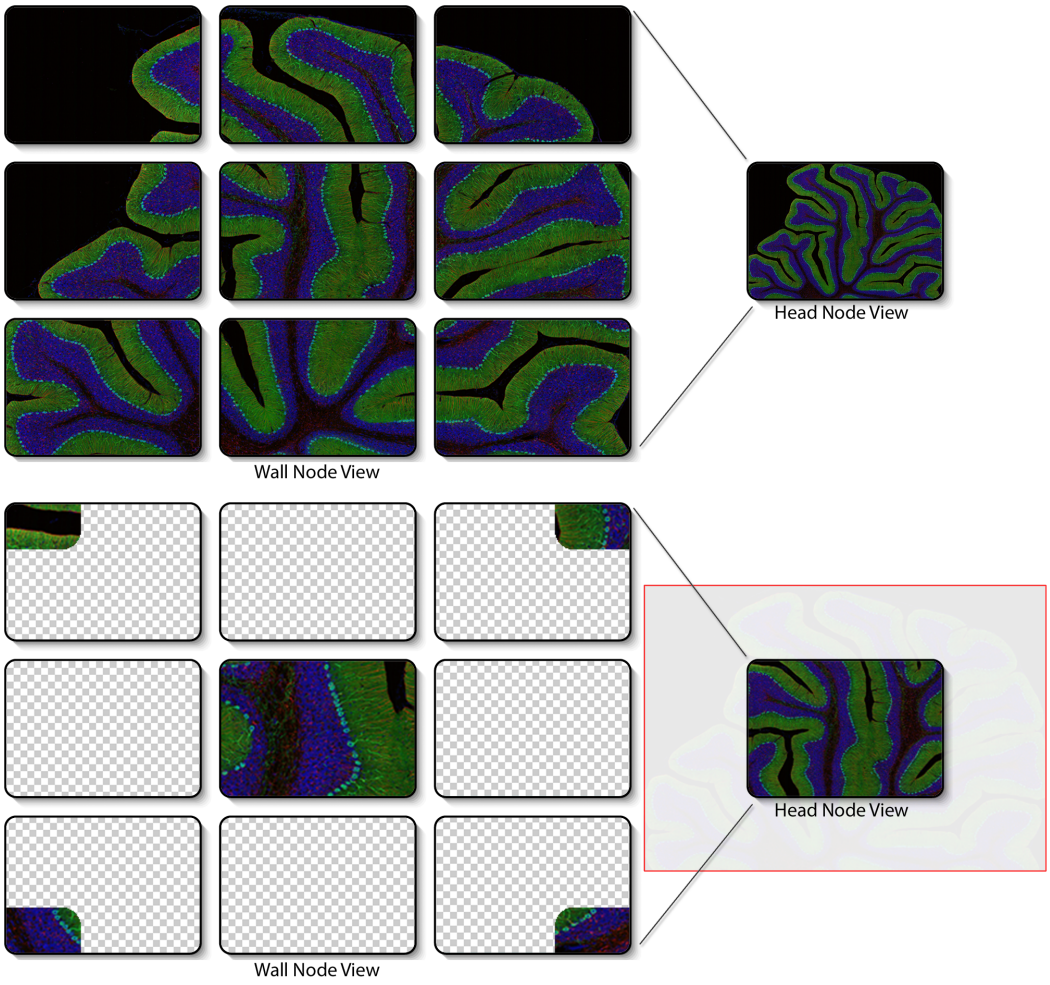


**Figure 2.5**: Problems with zooming on a tiled display. The top image shows the original image on the head node and tiled display. The bottom image shows the result of a 200 percent scale in the center of the image. The tiled display loses much of its data coherency.

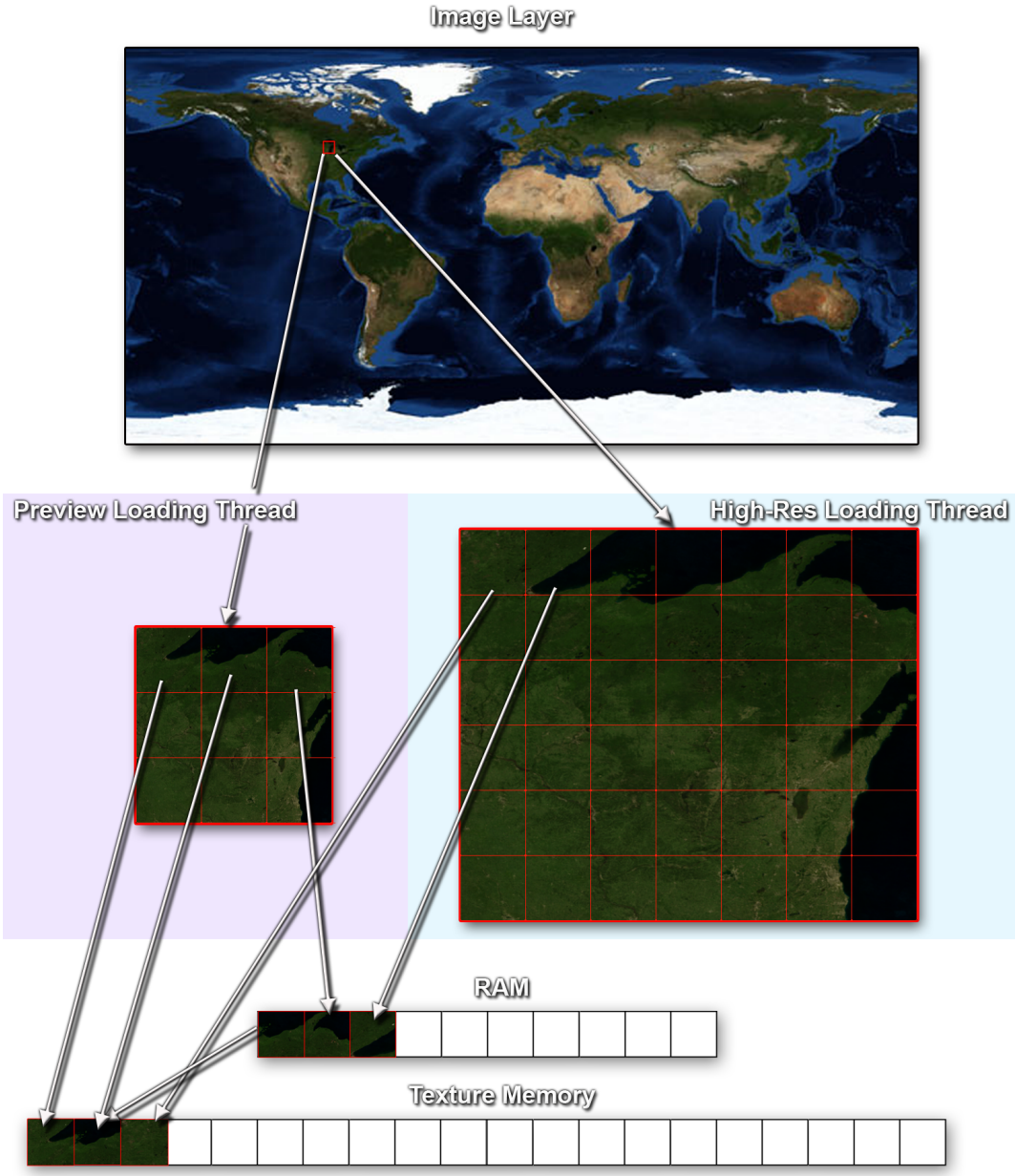


**Figure 2.6**: Diagram showing the mechanism for multithreaded texture loading.

blocking the display loop or displaying blank image tiles, to retain interactivity and full control over the environment. Two alternatives were developed to address the requirement. The first uses a protected section of texture memory, which is guaranteed to contain very low-resolution tiles for any region of the image. The preview resolution tiles can then be loaded and filled at startup. This approach in turn produces a progressive refinement step whereby images initially are rendered at low resolution and then snap into focus as the high-resolution tiles become available. To increase the quality of the preview, higher resolution samples can then simply be loaded into the protected texture memory. The trade-off with this technique is that higher resolution previews increase the memory footprint, decreasing overall effectiveness.

The second technique produces better quality results and optimizes memory efficiency via the use of a second loading thread, dedicated to fetching "preview resolution" tiles, as shown in Figure 2.6. This "preview resolution" is determined by profiling past loading performance and determining the number of tiles, which can be loaded at the targeted refresh rate. Optimally, this thread will load data from a different disk than the first thread, since disk thrashing may otherwise occur. In the setup, the first thread is set to read from local disks, while the second thread loads from network attached storage. The preview resolution tiles are only drawn when needed, allowing the preview to be swapped out of texture memory, once the full resolution image has been loaded.

As there is still a possibility that the "preview resolution" may not load fast enough for the display cycle, a single tile version of the image is stored in the protected section of texture memory. While this very low-resolution image is only used as a last resort, it is much more useful than showing an empty tile region. Because there is only a single tile protected for each image, the memory cost of this lowest resolution image is minimal. Using this method, "preview resolution" tiles and full resolution tiles can be swiftly blended, largely removing visual artifacts.

## 2.4 Display Loop

The display loop is responsible for aggregating the needed tiles into one seamless visual that can span multiple tiles per node at their native resolution. Given that high-resolution images may regularly consist of tens-of-thousands of tiles and that image size ideally should be unbounded, strategies such as culling are need. Culling geometry is trivial for the head node application, but is somewhat more demanding for the render nodes, as each nodes geometry inside of the wall must be considered. At startup, each render node queries the wall dimensions and determines the area of the wall for which it is accountable. Next, the node queries all displays connected to it to establish a pixel resource inventory and the corresponding canvas size. This canvas size is subsequently used for all decisions related to loading and displaying content. The advantage is that repeated calculation of display tile placement is no longer needed in combination with the ability of the loading loops to more easily load blocks of information all at once. Overall, this works well, even if display tiles are arranged in discontinuous blocks. It should be noted that the mapping between the head nodes display and the walls aspect ratio need to be taken into consideration, even if identical display tiles are used. For instance, the tiled display system is nearly twice as wide as the head nodes screen. This means that the head node either must be letter-boxed or images on the head node need to be stretched. It was found that full screen rendering on the head-node in combination with an overlay showing the actual wall dimension (semi-transparent letter box) tended to work best. The added benefit is that data currently not visible on the wall may still be viewable on the head node, further aiding in the analysis process. Since all of the needed tiles of an extremely high-resolution tiled image cannot be loaded simultaneously, some tiles may not yet be available when drawing timeslice has completed. If a tile is not yet in memory, the space for which the tile is to be displayed is left blank. If the drawing thread is unable to draw all of the desired tiles in the viewing area, the preview resolution is drawn to fill in the gaps. Because calculating how the holes from one resolution map to another is relatively time consuming, it was found to be much more effective to simply draw the entire screen area with the lower resolution behind

the higher resolution image plane. This process is rather fast as the GPU can do early Z-termination and simply occlude the pixels that had been rendered by the first image drawing. In the rare event that the preview resolution is also not in memory, this process is repeated, and the lowest resolution is drawn behind the preview resolution image.

To accomplish trilinear interpolation, the alpha blending internal to the OpenGL pipeline is utilized. The two image containers immediately behind and in front of the point of inspection are selected and loaded. The image in the back is drawn first, at full opacity. Subsequently, the top image is drawn in front with its opacity proportional to the distance from the inspection point to that image containers location. This method is problematic when the drawing of multiple resolutions is required. Specifically, although the back image could simply have gaps filled by drawing subsequent resolutions behind it, transparency in the top image would create blending artifacts. For example, if the point of inspection is midway between two image containers, then the top image container would be drawn with 50% transparency. But if subsequent resolutions would be drawn behind the higher resolutions, undesirable blending between the resolutions would occur. Hence, it is important for the blending to occur only between the first and second image container and not between various resolutions in an image container.

This problem can be solved by utilizing the mechanisms of OpenGLs blending and early z-termination, as shown in Figure 2.7. First the highest resolution variation of the lower image layer is drawn. All tiles in the current viewpoint, which are not loaded for this resolution, are left as gaps. Underneath, the preview resolution is drawn in a similar fashion, leaving holes for unloaded data. Finally, the lowest resolution is drawn behind the preview resolution. As each of the subsequent draws are rendered behind the higher resolution version, only areas which are unloaded on the higher resolution add to the final result due to early z-termination. If a resolution is fully loaded for a given viewpoint, lower resolution versions do not need to be rendered as they will not add anything to the final scene.

After the lower image container is fully rendered, the higher resolution image is rendered in a similar fashion above it. Each of the necessary resolutions

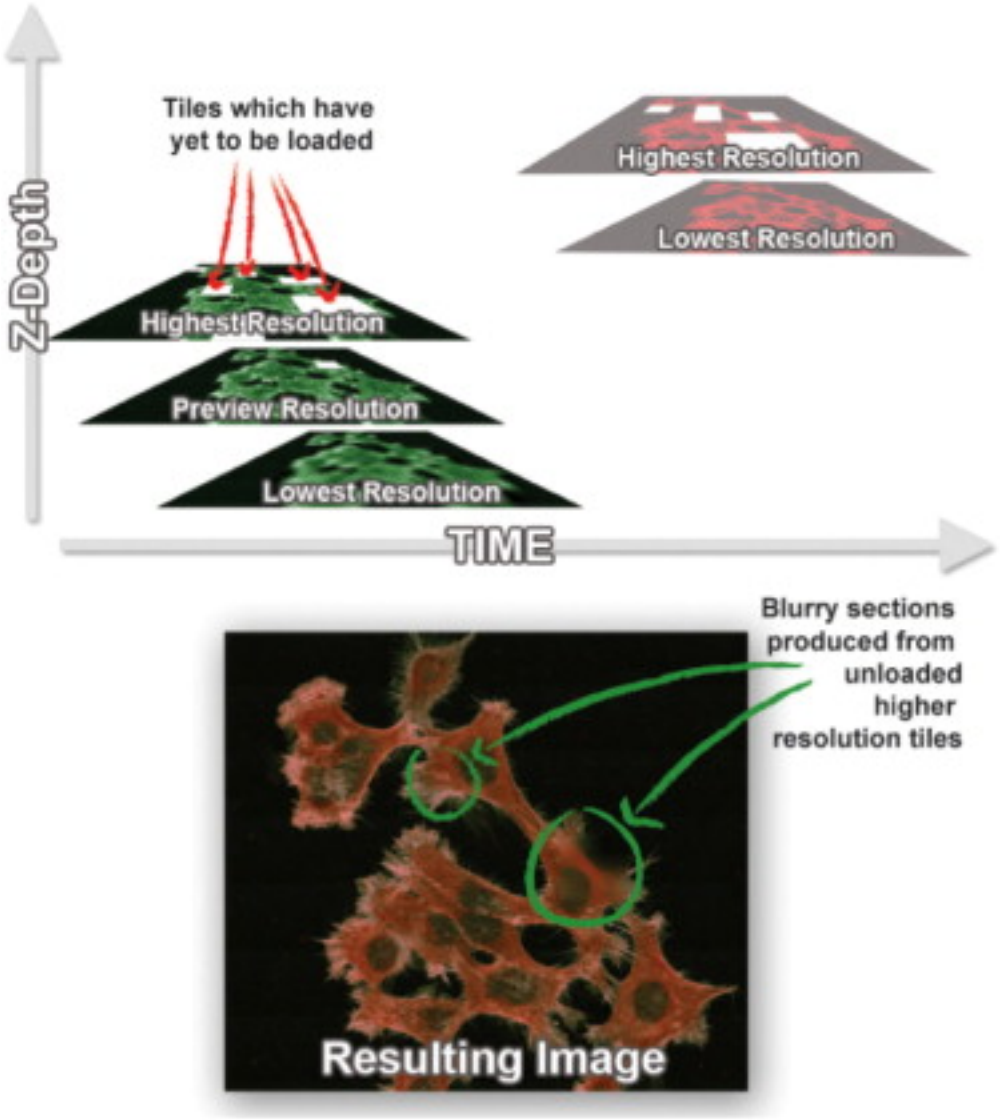


**Figure 2.7**: Demonstrates the drawing order and z-depth in order to achieve the desired blending. All three resolutions of the lower image container are drawn before the various resolution of the top image container. As shown in the resulting image, blurry sections can be caused by higher resolution tiles that have not yet been unloaded.

is rendered behind the highest resolution, but above all of the resolutions of the lower image container. Again, pixels that fall behind higher resolution rendered data will be terminated, allowing correct inspection point blending to be achieved.

## 2.5 Method for Generating Tiled Images

As stated previously, tiled images are used to facilitate rapid loading of subsections of images. To do this, the TIFF file format was selected since it provides a simple container that allows multiple images to be stored in a single TIFF file. This feature can be used to store multiple resolutions for a given image container inside of a single TIFF image container. The TIFF format also allows each of these images to be tiled, breaking up sections of the image into individually accessible pieces. With this setup, a tile from any resolution at any location inside of an image can be fetched without the need for other data fetches.

Tools such as VIPS [MC05] and [KUDC07] can readily be used to create tiled images. For the presented approach, images are first converted to the VIPS image format and subsequently to a tiled TIFF format using a tile size of 256x256 pixels and deflate compression providing fast and lossless compression.

## 2.6 Issues with Tiled Images

Tiling of images introduces other challenges such as border padding and tile boarder interpolation that have to be addressed.

### 2.6.1 Border Padding

Since images may not be a multiple of the tile size, extra image space, termed padding, is required to tile an image, as shown in Figure 2.8. Since different resolutions map to different sizes, different amounts of extra padding are needed for each resolution.

For example, tiling an image with a size of 600x600 pixels would require nine 256x256 sized tiles with a border of 168 pixels in each dimension. Tiling of its

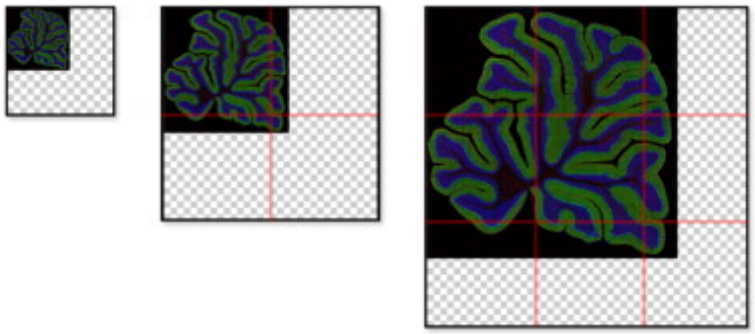

**Figure 2.8**: Example of the tiling for microscopy image of a rat brain. The red lines indicate tile boundaries.

lower resolution counterpart, a 300x300 pixel image, would require four 256x256 sized tiles with a border of 212 pixels in each dimension. Tiling of an even lower resolution counterpart, a 150x150 pixel image, would require one 256x256 sized tile with a border of 106 pixels in each dimension.

This means that whenever one image container switches from one resolution to the next, the extra padding required to fill in the image tiles will change. To solve this issue, the image is shaped such that extra space will fall outside of the image container. This way, the edge tiles will only draw the part of the tile containing actual image data.

Two items are worth noting in this setup. First, this method of tiling does not produce a quad-tree unless the tile size is a divisor of the image size. This is important for the drawing loop, as tiles do not overlap between resolutions in a regular fashion. For the example above, a pixel for the middle resolution will be the combination of 4 pixels from the higher resolution. But a group of four pixels in the center tile of the 3x3 tiled image could map to any of the four tiles in the 2x2 tiled image depending on their exact location in the image. In the opposite case, a single pixel on one resolution, when expanded to four pixels in the higher resolution, may require the loading of four tiles in order to be visualized correctly.

The other issue with image padding is the interpolation on the edges. The padding on the outside of the image is set to be transparent. This allows images to be layered without introducing a visible border. Since this extra padded section

is not drawn, this trait is not useful for us. Unfortunately, this transparent color does cause problems for the interpolation on the edge of the image since colors are interpolated using both the edge color and transparency. This is especially problematic for projected images for which opposing edges in reality are connected (e.g. spherical or cylindrical topology). The discussed Blue Marble Next Generation data set is a good example for this, where the farthest right pixel connects to the farthest left pixel. When two copies of the same image are placed side-by-side, border transparency would be considered during blending, revealing the background. To address this undesirable artifact, special treatment for tile border interpolation is required.

### 2.6.2 Tile Border Interpolation

Another issue with tiling images is the interpolation between tiles, as shown in Figure 2.9. Since these images reside in separate textures, interpolation between two tiles will not work automatically. Many tiled image systems get around this by simply using nearest-neighbor interpolation. For the presented technique, this short cut is not always effective since layered images may have drastically different resolutions. In practice it turned out that users preferred smoothened ("blurry") image transitions over pixelated ones.

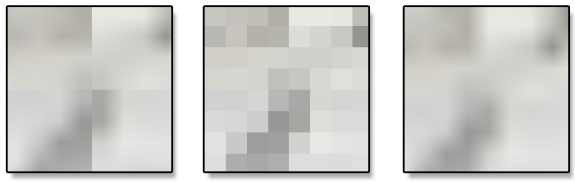

**Figure 2.9**: Three different interpolation techniques between tiles. The figure on the left shows GL_LINEAR, the figure in the center shows GL_NEAREST and the figure on the right shows the interpolation implemented in the Chapter.

To provide flexibility with tile border interpolation, an OpenGL shader with TEXTURE_RECT_ARB was created that operates as follows: four texture tiles are loaded into memory; texture 0 contains the tile currently being used, texture

1 contains the tile to the right of the current one, texture 2 contains the tile above the current texture, and finally texture 3 contains the tile to the right and above the current tile. Pseudo code is shown in the Appendix A.1.

This method effectively fixes problems of interpolation between tiles and allows blending of edge pixels. Since the tiled TIFF format creates tiles of equal size, the image will generally not completely fill the outer tiles. As outlined above, the border section in the TIFF will be encoded as transparent, allowing stacked images to appear correctly. Unfortunately if linear blending is used, the edge pixels will blend with the transparent border, causing fuzzy edges. To address this problem, the values for tilewidth and tileheight in the code above can simply be changed to the values of the image edge, allowing interpolation between opposing image edges.

## 2.7 Interaction

Intuitive and natural interaction was a primary design consideration and users have access to the multi-layered image data via a regular node with dual 30 in. displays, using the large-form-factor wall display wall as an extended display. In addition, a wireless gyroscopic mouse may be used to freely interact with the wall, allowing users to translate the image layers by clicking and dragging with the left mouse button, zoom into image layers by clicking and dragging with the right mouse button, or move the inspection point by scrolling through the layers using the mouse wheel. The system can also be set to automatically change viewing attributes without the need for continuous user input. Slowly panning across the image while the layers are automatically blended following user defined timing characteristics, turned out to be one of the most powerful visual analytics tools to extract correlations between layers, while establishing the “big picture”.

## 2.8 Applications

This project was specially designed to permit manipulation of ultra high-resolution multi-layered data sets at interactive rates. The Blue Marble Next Generation data set was selected as a primary case-study example. It provides whole-earth coverage on a per month basis for 2004, resulting in twelve 3.7 giga-pixel images (44.4 giga-pixels total) and a thereby an image stack encoding a broad set of temporal and environmental characteristics. On the HIPerSpace system, users then can inspect 286 million pixels simultaneously and blend between layers instantaneously. To put this in perspective, the created visuals are displayed at a resolution two-orders of magnitude higher than next-generation, high-definition television (1080p).

At this resolution and overall display canvas size, users can combine digital and physical zoom, by resizing image layers digitally or simply varying the physical distance from the wall; i.e. by standing further back, users can view the entire display at once, while walking closer allows for the interrogation of selected regions of interest without loss of perceived resolution. This paradigm becomes even more powerful when dozens of users collaborate face-to-face on data analysis tasks.

For example, using the Blue Marble Next Generation data set [SVS+05], users can clearly follow the seasons changing by setting the system on a continuously moving inspection point. This immediately exposes receding snow banks, rising rivers, and greening fields, providing a powerful tool for the analysis of local, regional and global climate variations throughout a given year.

Microscopy imagery also proves to be an interesting source for multi-spectral data. For example, scientists at the National Center for Microscopy Research (NCMIR) have developed ultra high-resolution imaging techniques for organic tissue. Using their confocal microscopes, a rat brain may be imaged at hundreds of mega pixel resolution following the staining of cells with different dyes [CEM01]. Dyes in this case are useful to help identify distributions of glial cell intermediate filament protein or calcium channel enriched Purkinje cells, and to show different DNA attributes. Generally, these dyed layers are treated as individual color channels and subsequently merged into a single high-resolution image foranalysis.

**Table 2.1**: Time required per frame for the three different methods of tile replacement as described in this Chapter.

| Method | Average Time (ms) | Worst Time (ms) |
|---|---|---|
| Sorted List | 0.598167 | 49.243556 |
| Semi Sorted List | 0.00694 | 0.21322 |
| Modified Round Robin | 0.00389 | 0.07739 |

With the presented approach, it is possible to keep the individual image layers and to composite them on the fly as they are being analyzed. This allows for novel, targeted and interactive, concurrent interrogation of raw and synthesized data, with the ability to add extra image layers on the fly.

## 2.9 Results

It is difficult to quantitatively measure system performance due to a broad mix of quality-of-service parameters, such as data caching across the network, network latency, jitter and packet loss. As one would expect, overall performance greatly depends on the amount of data that each node has to load. Test results are based on the presented worst-case-scenario of a quad-display setup, with a total of 16 mega-pixels being served per node. No other system which runs natively at the resolution of the tiled display system is known to perform the operations described in this Chapter for baseline measurements.

Performance tests were broken up into minute long intervals, targeting core image analysis tasks. First, the image stack was panned from left to right while a series of sinusoidal zooms, from the highest resolution to the lowest resolution, ran on six second intervals. Next, the inspection point was continuously progressed through the stack, such that each image layer was interpolated over for three-second intervals. For these tests, the global texture pool was set to handle 4000 256x256-sized tiles.

Each of the methods described in the replacement section above were tested and timed for operations such as sorting, reordering, and selection of the removal

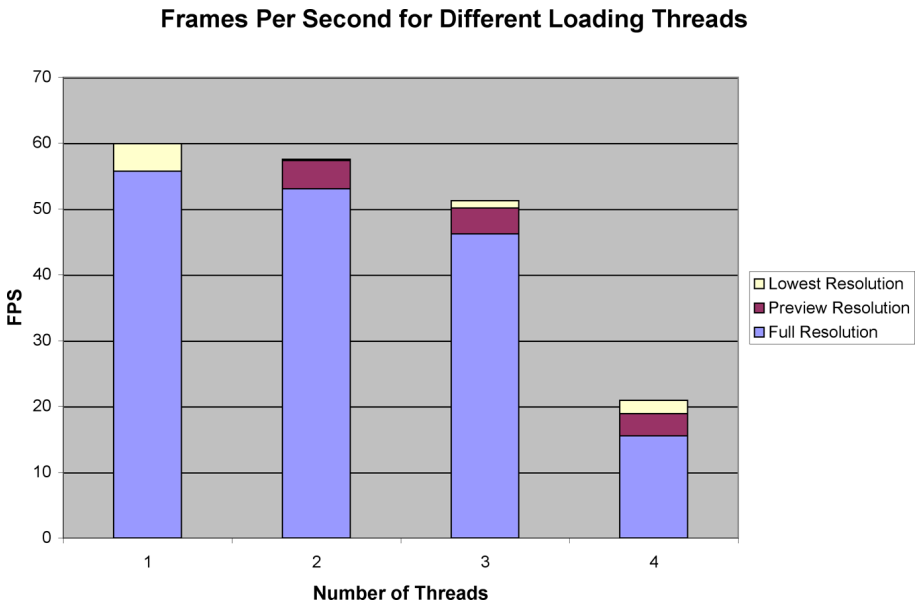


**Figure 2.10**: Graph showing frame rates divided into the number of frames which were drawn using only the full resolution texture, the full and preview resolution textures, and the full, preview and lowest resolution textures.

tile via the system clock. The test was repeated 10,000 times, and average and worst-case times were recorded. As shown in Table 2.1, the sorted list produced the worst average and worst-case times, while the modified round robin method produced the best average and worst-case times.

The system was also tested using the threaded loading approach as stated above. The respective frame rates for different numbers of loading threads are shown in Figure 2.10. By adding the second loading thread, the frame rate decreased slightly, but, as opposed to the single thread loading approach where the lowest resolution texture tiles were needed 7% of the time, the lowest resolution tiles were needed less than 0.5% of the time (see Table 2.2). The addition of more than two threads created contention on the system and reduced its overall effectiveness.

**Table 2.2**: Table showing the percentage of frames which were drawn using only the full resolution texture, the full and preview resolution textures, and the full, preview and lowest resolution textures.

| # Threads | % Highest | % Preview | % Lowest |
|---|---|---|---|
| 1 | 93.0 | 0.0 | 6.9 |
| 2 | 92.2 | 7.4 | 0.3 |
| 3 | 90.1 | 7.7 | 2.1 |
| 4 | 74.3 | 16.0 | 9.5 |

## 2.10 Conclusion

This chapter presents a technique for the interactive and intuitive visualization of large multi-dimensional data. While the primary focus of this Chapter caters to large tiled displays, the introduced methods work equally well for single display computers and laptops, while scaling gracefully as nodes are being added. In the context of multi-tile or distributed display environments, this approach allows for multiple users to analyze large data sets simultaneously. Beyond the shown examples of the robust applications that these methods provide for geoscientists and biologists, the usefulness of these methods transcends into other fields and applications.

## 2.11 Acknowledgments

This Chapter, is a reprint of "Giga-Stack: A Method for Visualizing Gigapixel Layered Imagery on Massively Tiled Display" as it appears in Future Generation Computer Systems 2010, Volume 26, Number 5. Ponto, K., Doerr, K., and Kuester, F. with permission from Elsevier. The dissertation author was the primary investigator and author of this paper.

# Bibliography


[Ake93] Kurt Akeley. Reality engine graphics. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 109–116, New York, NY, USA, 1993. ACM.

[bbc] http://www.apple.com/quicktime/guide/hd/bbc-cfb.html.

[BJH+08] Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira. Vr juggler: a virtual platform for virtual reality application development. In *SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 courses*, pages 1–8, New York, NY, USA, 2008. ACM.

[BN05] Robert Ball and Chris North. Effects of tiled high-resolution display on basic visualization and navigation tasks. In *CHI '05 extended abstracts on Human factors in computing systems*, pages 1196–1199, New York, NY, USA, 2005. ACM.

[BR98] Uwe Behrens and Ralf Ratering. Adding shadows to a texture-based volume renderer. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 39–46, New York, NY, USA, 1998. ACM.

[BVG05] Stefan Bruckner, Ivan Viola, and M. Eduard Gröller. Volumeshop: interactive direct volume illustration. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, page 60, New York, NY, USA, 2005. ACM.

[BXH+09] Leonardo Bonanni, Xiao Xiao, Matthew Hockenberry, Praveen Subramani, Hiroshi Ishii, Maurizio Seracini, and Jurgen Schulze. Wetpaint: scraping through multi-layered images. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 571–574, New York, NY, USA, 2009. ACM.

[car] http://www.apple.com/trailers/disney/cars/.

[CB04] Xiang Cao and Ravin Balakrishnan. Visionwand: interaction techniques for large displays using a passive wand tracked in 3d. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 729–729, New York, NY, USA, 2004. ACM.

[CCF94] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS '94: Proceedings of the 1994 symposium on Volume visualization*, pages 91–98, New York, NY, USA, 1994. ACM.

[CEM01] F. Capani, M.H. Ellisman, and M.E. Martone. Filamentous actin is concentrated in specific subpopulations of neuronal and glial structures in rat central nervous system. *Brain Research*, 923(1-2):1–11, 2001.

[Che02] Han Chen. A parallel ultra-high resolution mpeg-2 video decoder for pc cluster based tiled display system. to appear. In *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS), IEEE CS*, page 30. Press, 2002.

[Che03] Han Chen. *Scalable and Ultra-High Resolution MPEG Video Delivery on Tiled Displays*. PhD thesis, Princeton University, 2003.

[CHS04] Ian Creighton and Chris Ho-Stuart. A sense of touch in online sculpting. In *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 118–122, New York, NY, USA, 2004. ACM.

[CI05] Alvaro Cassinelli and Masatoshi Ishikawa. Khronos projector. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Emerging technologies*, page 10, New York, NY, USA, 2005. ACM.

[Cor09] Carlos D. Correa. Visualizing what lies inside. *SIGGRAPH Comput. Graph.*, 43(2):1–6, 2009.

[CS02] Hui Chen and Hanqiu Sun. Real-time haptic sculpting in virtual volume space. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 81–88, New York, NY, USA, 2002. ACM.

[CSC06] Carlos Correa, Deborah Silver, and Min Chen. Feature aligned volume manipulation for illustration and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1069–1076, 2006.

[CSM02] E.F. Churchill, D.N. Snowdon, and A.J. Munro. Collaborative virtual environments: digital places and spaces for interaction. *Educational Technology & Society*, 5(4), 2002.

[CT09] Andrew A. Chien and Nut Taesombut. Integrated resource management for lambda-grids: The distributed virtual computer (dvc). *Future Generation Computer Systems*, 25(2):147 – 152, 2009.

[DC02] James Davis and Xing Chen. Lumipoint: multi-user laser-based interaction on large tiled displays. *Displays*, 23(5):205 – 211, 2002.

[DDS+09] Thomas A. DeFanti, Gregory Dawe, Daniel J. Sandin, Jurgen P. Schulze, Peter Otto, Javier Girado, Falko Kuester, Larry Smarr, and Ramesh Rao. The starcave, a third-generation cave and virtual reality optiportal. *Future Generation Computer Systems*, 25(2):169 – 178, 2009.

[DK10] Kai-Uwe Doerr and Falko Kuester. CGLX: A Scalable, High-performance Visualization Framework for Networked Display Environments. *IEEE Transactions on Visualization and Computer Graphics*, 99(PrePrints), 2010.

[DL01] P. Dietz and D. Leigh. Diamondtouch: a multi-user touch technology. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 219–226. ACM New York, NY, USA, 2001.

[DLR+09] Thomas A. DeFanti, Jason Leigh, Luc Renambot, Byungil Jeong, Alan Verlo, Lance Long, Maxine Brown, Daniel J. Sandin, Venkatram Vishwanath, Qian Liu, Mason J. Katz, Philip Papadopoulos, Joseph P. Keefe, Gregory R. Hidley, Gregory L. Dawe, Ian Kaufman, Bryan Glogowski, Kai-Uwe Doerr, Rajvikram Singh, Javier Girado, Jurgen P. Schulze, Falko Kuester, and Larry Smarr. The optiportal, a scalable visualization, storage, and computing interface device for the optiputer. *Future Generation Computer Systems*, 25(2):114 – 123, 2009.

[EKCB03] Jr. Easton, R.L., K.T. Knox, and W.A. Christens-Barry. Multispectral imaging of the archimedes palimpsest. *Applied Imagery Pattern Recognition Workshop, 2003. Proceedings. 32nd*, pages 111–116, Oct. 2003.

[EKE01] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel

shading. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 9–16, New York, NY, USA, 2001. ACM.

[Elv92] T. Todd Elvins. A survey of algorithms for volume visualization. *SIGGRAPH Comput. Graph.*, 26(3):194–201, 1992.

[FAJ07] G. Flint, C. Aves, and MT Jones. The gigapxl project. http://www.gigapxl.org, 2007.

[GH91] Tinsley A. Galyean and John F. Hughes. Sculpting: an interactive volumetric modeling technique. *SIGGRAPH Comput. Graph.*, 25(4):267–274, 1991.

[Gra72] R. L. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132 – 133, 1972.

[GRC+07] J.F. Gantz, D. Reinsel, C. Chute, W. Schlichting, J. McArthur, S. Minton, I. Xheneti, A. Toncheva, and A. Manfrediz. The expanding digital universe: A forecast of worldwide information growth through 2010. *IDC white paper*, 2007.

[GSW01] François Guimbretière, Maureen Stone, and Terry Winograd. Fluid interaction with high-resolution wall-size displays. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 21–30, New York, NY, USA, 2001. ACM.

[HA08] J. Heer and M. Agrawala. Design considerations for collaborative visual analytics. *Information Visualization*, 7(1):49–62, 2008.

[Han05] J.Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 115–118. ACM New York, NY, USA, 2005.

[Har90] Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335 – 346, 1990.

[HEB+01] Greg Humphreys, Matthew Eldridge, Ian Buck, Gordan Stoll, Matthew Everett, and Pat Hanrahan. Wiregl: a scalable graphics system for clusters. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 129–140, New York, NY, USA, 2001. ACM.

[Her08] L. Herr. Creation and Distribution of 4 K Content. *Television Goes Digital*, page 99, 2008.

[HKSB06] M. Hadwiger, A. Kratz, C. Sigg, and K. Bühler. Gpu-accelerated deep shadow maps for direct volume rendering. In *Graphics Hardware 2006: Eurographics Symposium Proceedings, Vienna, Austria, September 3-4, 2006*, pages 49–52. Eurographics Association, 2006.

[HLSR08] Markus Hadwiger, Patric Ljung, Christof Rezk Salama, and Timo Ropinski. Advanced illumination techniques for gpu volume raycasting. In *SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 courses*, pages 1–166, New York, NY, USA, 2008. ACM.

[HYB02] T. Hansen, P. Yalamanchili, and H.W. Braun. Wireless measurement and analysis on HPWREN. In *Proceedings of Passive and Active Measurement Workshop, Fort Collins, Co*, pages 222–229, 2002.

[Ini06] Digital Cinema Initiatives. Standard evaluation material (stem), 2006.

[JC06] G. Johansson and H. Carr. Accelerating marching cubes with graphics hardware. In *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*, page 39. ACM New York, NY, USA, 2006.

[JJR+05] B. Jeong, R. Jagodic, L. Renambot, R. Singh, A. Johnson, and J. Leigh. Scalable graphics architecture for high-resolution displays. In *IEEE Information Visualization Workshop*, 2005.

[JRJ+06] Byungil Jeong, Luc Renambot, Ratko Jagodic, Rajvikram Singh, Julieta Aguilera, Andrew Johnson, and Jason Leigh. High-performance dynamic graphics streaming for scalable adaptive graphics environment. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 108, New York, NY, USA, 2006. ACM.

[KCC08] D. Kim, K. Cha, and S.I. Chae. A high-performance openvg accelerator with dual-scanline filling rendering. *IEEE Transactions on Consumer Electronics*, 54(3):1303–1311, 2008.

[KKH01] J. Kniss, G. Kindlmann, and C. Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of the conference on Visualization'01*, pages 255–262. IEEE Computer Society Washington, DC, USA, 2001.

[KSR+06] Matthias Koenig, Wolf Spindler, Jan Rexilius, Julien Jomier, Florian Link, and Heinz-Otto Peitgen. Embedding vtk and itk into a visual programming and rapid prototyping platform. *Medical Imaging 2006: Visualization, Image-Guided Procedures, and Display*, 6141(1):61412O, 2006.

[KUDC07] Johannes Kopf, Matt Uyttendaele, Oliver Deussen, and Michael F. Cohen. Capturing and viewing gigapixel images. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 93, New York, NY, USA, 2007. ACM.

[KVV+04] NK Krishnaprasad, V. Vishwanath, S. Venkataraman, AG Rao, L. Renambot, J. Leigh, AE Johnson, and B. Davis. JuxtaView-a tool for interactive visualization of large imagery on scalable tiled displays. In *Cluster Computing, 2004 IEEE International Conference on*, pages 411–420, 2004.

[LBS85] SK Lee, W. Buxton, and KC Smith. A multi-touch three dimensional touch-sensitive tablet. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 21–25. ACM New York, NY, USA, 1985.

[LC87] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169. ACM New York, NY, USA, 1987.

[Lee84] S. Lee. A fast multiple-touch-sensitive input device. *Master's thesis, University of Toronto*, 1984.

[Leh97] Roy S. Lehrle. Forensics, fakes, and failures: Pyrolysis is one part in the overall armoury. *Journal of Analytical and Applied Pyrolysis*, 40-41:3 – 19, 1997. PYROLYSIS '96.

[LM04] Eric B. Lum and Kwan-Liu Ma. Lighting transfer functions using gradient aligned sampling. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 289–296, Washington, DC, USA, 2004. IEEE Computer Society.

[Mar91] K. Martinez. High resolution digital imaging of paintings: The vasari project. *Microcomputers for Information Management*, 8(4):277–83, 1991.

[MC05] Kirk Martinez and John Cupitt. Vips - a highly tuned image processing software architecture. In *ICIP (2)*, pages 574–577, 2005.

[MCSP02] K. Martinez, J. Cupitt, D. Saunders, and R. Pillay. Ten years of art imaging research. *Proceedings of the IEEE*, 90(1):28–41, 2002.

[MDH+03] A. MacEachren, X. Dai, F. Hardisty, D. Guo, and G. Lengerich. Exploring high-D spaces with multiform matrices and small multiples. In *IEEE Symposium on Information Visualization, 2003 (INFOVIS 2003); 19–21 Oct. 2003; Seattle, Washington*, pages 31–38. Citeseer, 2003.

[Mit97] J.L. Mitchell. *MPEG video compression standard*. Kluwer Academic Publishers, 1997.

[Mor98] H. Moravec. When will computer hardware match the human brain. *Journal of Evolution and Technology*, 1:1–14, 1998.

[MRB05] Shahzad Malik, Abhishek Ranjan, and Ravin Balakrishnan. Interacting with large displays from a distance with vision-tracked multi-finger gestural input. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 43–52, New York, NY, USA, 2005. ACM.

[MTB03] Michael J. McGuffin, Liviu Tancau, and Ravin Balakrishnan. Using deformations for browsing volumetric data. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 53, Washington, DC, USA, 2003. IEEE Computer Society.

[NR02] S. Navrud and R.C. Ready. *Valuing cultural heritage*. Elgar, 2002.

[PDMDRP08] A. Pelagotti, A. Del Mastio, A. De Rosa, and A. Piva. Multispectral imaging of paintings. *Signal Processing Magazine, IEEE*, 25(4):27–36, July 2008.

[PKS+08] Peter Peltonen, Esko Kurvinen, Antti Salovaara, Giulio Jacucci, Tommi Ilmonen, John Evans, Antti Oulasvirta, and Petri Saarikko. It's mine, don't touch!: interactions at a large multi-touch display in a city centre. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1285–1294, New York, NY, USA, 2008. ACM.

[Ple08] L. Plesea. The design, implementation and operation of the JPL OnEarth WMS server. In *Geospatial Services and Applications for the Internet*, pages 93–109. Springer US, 2008.

[PSH97] Vladimir I. Pavlovic, Rajeev Sharma, and Thomas S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. 1997.

[PWFO01] KL PERNG, WT WANG, M. FLANAGAN, and M. OUHYOUNG. A Real-time 3D Virtual Sculpting Tool Based on Modified Marching Cubes. In *Int Conf Artif Real Telexistence*, volume 11, pages 64–72, 2001.

[RBJW01] Meredith Ringel, Henry Berg, Yuhui Jin, and Terry Winograd. Barehands: implement-free interaction with a wall-mounted display. In *CHI '01: CHI '01 extended abstracts on Human factors in computing systems*, pages 367–368, New York, NY, USA, 2001. ACM.

[Rek98] Jun Rekimoto. A multiple device approach for supporting whiteboard-based interactions. In *CHI '98: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 344–351, New York, NY, USA, 1998. ACM Press/Addison-Wesley Publishing Co.

[RJJ+06] L. Renambot, B. Jeong, R. Jagodic, A. Johnson, J. Leigh, and J. Aguilera. Collaborative visualization using high-resolution tiled displays. In *ACM CHI Workshop on Information Visualization Interaction Techniques for Collaboration Across Multiple Displays*, 2006.

[RJL05] L. Renambot, A. Johnson, and J. Leigh. Lambdavision: Building a 100 megapixel display. In *NSF CISE/CNS Infrastructure Experience Workshop, Champaign, IL*, 2005.

[RP00] M. Riesenhuber and T. Poggio. Models of object recognition. *Nature Neuroscience*, 3:1199–1204, 2000.

[Ryd] Thomas Rydell. Virtual autopsy table. https://www.tii.se/projects/autopsy.

[SBD+] J. Schöning, P. Brandl, F. Daiber, F. Echtler, O. Hilliges, J. Hook, M. Löchtefeld, N. Motamedi, L. Muller, P. Olivier, et al. Multi-touch surfaces: A technical guide.

[SBdL09] Larry Smarr, Maxine Brown, and Cees de Laat. Special section: Optiplanet – the optiputer global collaboratory. *Future Generation Computer Systems*, 25(2):109 – 113, 2009.

[SC93] D. Saunders and J. Cupitt. Image processing at the national gallery: The vasari project. 1993.

[SGHB07] J.D. Smith, TC Graham, D. Holman, and J. Borchers. Low-cost malleable surfaces with multi-touch pressure sensitivity. In *Horizontal Interactive Human-Computer Systems, 2007. TABLETOP'07.*

*Second Annual IEEE International Workshop on*, pages 205–208, 2007.

[SGM03] Stacey D. Scott, Karen D. Grant, and Regan L. Mandryk. System guidelines for co-located, collaborative work on a tabletop display. In *ECSCW'03: Proceedings of the eighth conference on European Conference on Computer Supported Cooperative Work*, pages 159–178, Norwell, MA, USA, 2003. Kluwer Academic Publishers.

[SHP+96] Rajeev Sharma, Thomas S. Huang, Vladimir I. Pavlovi'c, Yunxin Zhao, Zion Lo, Stephen Chu, Klaus Schulten, Andrew Dalke, Jim Phillips, Michael Zeller, and William Humphrey. Speech/gesture interface to a visual computing environment for molecular biologists. In *IEEE Computer Graphics and Applications*, pages 30–35, 1996.

[SLJM08] D. Svistula, J. Leigh, A. Johnson, and P. Morin. MagicCarpet: a high-resolution image viewer for tiled displays, 2008.

[SPS48] C. Shannon, N. Petigara, and S. Seshasai. The Mathematical Theory of Communication . *Communication, Bell System Technical Journal*, 1948.

[Sre08] M. Sreenivasan. Microsoft silverlight. 2008.

[STA] C. STANDARD. THE MPEG VIDEO COMPRESSION STANDARD.

[SVFR04] Chia Shen, Frédéric D. Vernier, Clifton Forlines, and Meredith Ringel. Diamondspin: an extensible toolkit for around-the-table interaction. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 167–174, New York, NY, USA, 2004. ACM.

[SVS+05] R. Stockli, E. Vermote, N. Saleous, R. Simmon, and D. Herring. he blue marble next generation – a true color earth dataset including seasonal dynamics from modis. *Published by the NASA Earth Observatory*, 2005.

[SW06] Bram Stolk and Paul Wielinga. Building a 100 mpixel graphics device for the optiputer. *Future Generation Computer Systems*, 22(8):972 – 975, 2006.

[SYK+05] H. Shimamoto, T. Yamashita, N. Koga, K. Mitani, M. Sugawara, F. Okano, M. Matsuoka, J. Shimura, I. Yamamoto, T. Tsukamoto, et al. An Ultrahigh-Definition Color Video Camera With 1.25-inch

Optics and 8k x 4k Pixels. *SMPTE Motion Imaging Journal*, pages 3–11, 2005.

[SYS+06] D. Shirai, T. Yamaguchi, T. Shimizu, T. Murooka, and T. Fujii. 4k shd real-time video streaming system with jpeg 2000 parallel codec. In *Circuits and Systems, 2006. APCCAS 2006. IEEE Asia Pacific Conference on*, pages 1855–1858, Dec. 2006.

[TC05] James J. Thomas and Kristin A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Ctr, 2005.

[TFM96] S. Thorpe, D. Fize, and C. Marlot. Speed of processing in the human visual system. *Nature*, 381(6582):520–522, 1996.

[Tuf91] E.R. Tufte. Envisioning information. *Optometry and Vision Science*, 68(4):322, 1991.

[TWC+06] Nut Taesombut, Xinran (Ryan) Wu, Andrew A. Chien, Atul Nayak, Bridget Smith, Debi Kilb, Thomas Im, Dane Samilo, Graham Kent, and John Orcutt. Collaborative data visualization for earth sciences with the optiputer. *Future Generation Computer Systems*, 22(8):955 – 963, 2006.

[VBRR02] G. Voß, J. Behr, D. Reiners, and M. Roth. A multi-thread safe foundation for scene graphs and its extension to clusters. In *EGPGV '02: Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, pages 33–37, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[VL03] H.R. Varian and P. Lyman. How much information. *University of California at Berkeley, School of Information Management & Systems (SIMS)*, 2003.

[VOT] http://www.nationalgeographic.com/field/projects/valley-khans-project.html.

[WAB+05] G. Wallace, O.J. Anshus, P. Bi, H. Chen, Y. Chen, D. Clark, P. Cook, A. Finkelstein, T. Funkhouser, Anoop Gupta, M. Hibbs, K. Li, Z. Liu, Rudrajit Samanta, Rahul Sukthankar, and O. Troyanskaya. Tools and applications for large-scale display walls. *Computer Graphics and Applications, IEEE*, 25(4):24–33, 2005.

[WE98] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *Proceedings of SIGGRAPH*, volume 98, pages 169–178, 1998.

[WEH01] W. Westerman, J. Elias, and A. Hedge. Multi-touch: A new tactile 2-d gesture interface for human-computer interaction. In *Proceedings of the Human Factors and Ergonomics Society 45th Annual Meeting*, volume 1, pages 632–636, Minneapolis/St. Paul, MN, 2001.

[Wil83a] Lance Williams. Pyramidal parametrics. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 1–11, New York, NY, USA, 1983. ACM.

[Wil83b] Lance Williams. Pyramidal parametrics. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 1–11, New York, NY, USA, 1983. ACM.

[Wil04] A.D. Wilson. Touchlight: an imaging touch screen and display for gesture-based interaction. In *Proceedings of the 6th international conference on Multimodal interfaces*, pages 69–76. ACM New York, NY, USA, 2004.

[WK95] Sidney W. Wang and Arie E. Kaufman. Volume sculpting. In *I3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 151–ff., New York, NY, USA, 1995. ACM.

[wms] http://www.opengeospatial.org/standards/wms.

[Zha09] Jian-Feng Zhang. Gpu-based direct volume rendering with advanced illumination and deep attenuation shadows. *Computer-Aided Design and Computer Graphics, 2009. CAD/Graphics '09. 11th IEEE International Conference on*, pages 536 –539, 2009.