Evaluation of SGI Vizserver

James E. Fowler

NSF Engineering Research Center

Mississippi State University

A Report Prepared for the

High Performance Visualization Center Initiative (HPVCI)

March 31, 2000

Technical Report MSSU-COE-ERC-01-01, Engineering Research Center, Mississippi State University, March 2000

Contents

1. Introduction **3**

2. Vizserver Architecture and Operation **3**
2.1 Vizserver Architecture 3
2.2 Vizserver Compression 4
2.2.1 The CCC Algorithm 4
2.2.2 The ICC Algorithm 6
2.3 Vizserver API 7

3. Vizserver Performance **7**
3.1 Frame Rate 7
3.2 Bandwidth 8
3.3 Distortion 9

4. Conclusions **10**

1. Introduction

SGI Vizserver [1] is a product developed by Silicon Graphics, Inc., to enable remote-visualization applications. Specifically, SGI Vizserver is designed to provide users remote access to graphics pipelines of Onyx2 Infinite Reality machines so that they may view rendered output from visualization applications at geographically remote locations while utilizing the powerful pipeline and expansive memory of an Onyx2 machine located at a some centralized place.

The fundamental principle underlying SGI Vizserver is that all rendering takes place on the remote and more powerful Onyx2 instead of on the user's presumably less powerful local machine. Under this paradigm, the user runs a visualization application remotely, while Vizserver captures the rendered imagery output from the application and transmits an image sequence to the local machine for display. An important aspect of the product is that the operation of Vizserver is completely transparent to the application—the application renders visualization imagery as if it were running locally on the Onyx2, with all sophisticated hardware advantages (e.g., memory, graphics pipeline) of the Onyx2 utilized. Vizserver itself does no rendering; it merely controls where rendering takes place and provides a communication path to the local machine for display. Since the image sequences arising in visualization applications can typically have large resolution and frame rate, the bandwidth required of the communication link to the local machine can be particularly large. As a consequence, Vizserver attempts to mitigate the bandwidth burden on the communication link by providing optional compression of the transmitted images.

The goal of this report is to provide an evaluation of SGI Vizserver, identifying the advantages of deploying Vizserver as a solution to remote visualization as well as the disadvantages of this approach. As part of this evaluation, we present an in depth description of the architecture of the Vizserver product and overview its method of operation. Since the optional compression functionality of Vizserver is a key component in its performance, we explore in some depth the compression algorithms employed by Vizserver. We follow with an overview of our observations resulting from a variety of remote-visualization tests we have conducted using Vizserver; in our observations, we include certain measurements we have made regarding visual image quality, frame rate, and communication bandwidth. Finally, we conclude by enumerating the advantages and disadvantages of Vizserver as a solution to the task of remote visualization.

2. Vizserver Architecture and Operation

2.1 Vizserver Architecture

To best understand Vizserver and how it works, it is essential to understand how remote visualization can be accomplished in the absence of Vizserver. The most straightforward approach to remote visualization is to use the remote-display capabilities that have long been fundamental to the operation of the X Window System. This X-based remote-visualization paradigm is shown in Fig. 1. Here, a visualization application runs on the remote machine while the OpenGL software library provides all graphics rendering. The OpenGL library produces, using software rendering routines, a sequence of images which is transmitted to the local machine for display.[1] For this transmission, the OpenGL library acts as a standard X client and communicates with the X server running on the local machine. The standard X Protocol is used for the communication; specifically, the images are transmitted individually, frame-by-frame, using a sequence of XPutImage commands, which simply transmit the image pixels in an uncompressed, raster-scan fashion. Although completely transparent to the application, this X-based approach typically does not represent a reasonable paradigm for remote-visualization applications. Although the application can employ

[1]An alternative paradigm for X-based remote visualization entails the sending of OpenGL commands, rather than rendered images, from the remote to local machine, with rendering taking place on the local machine; for this approach to work, both machines must be equipped with special software to transmit/receive the GLX protocol for OpenGL-command transmission, otherwise operation defaults to as described above. We do not explore this possibility in this report as we assume we are interested in only approaches in which rendering takes place on the remote machine.

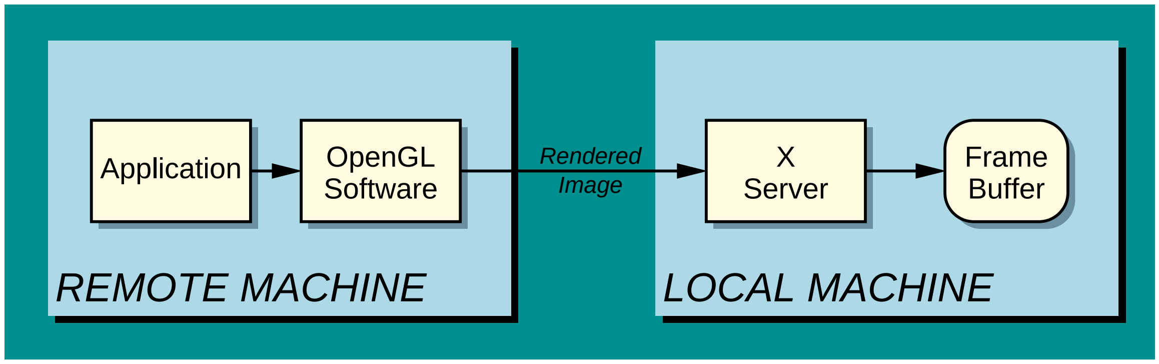


Figure 1: Remote visualization using the remote-display capabilities inherent to the X Window System. Rendering takes place in the OpenGL software library, and resulting images are transmitted, uncompressed, to the X server using the X Protocol.

the presumably large memory resources of the remote machine, all rendering takes place in software in the OpenGL library rather than employing the hardware graphics pipeline. This reliance on software rendering will usually result in a significantly decreased frame rate as compared to hardware rendering. An additional drawback to this X-based approach is that, because communications take place via the X Protocol, no compression is applied to the transmitted images. As a result, the bandwidth required of the communications link is usually prohibitive, unless the frame rate is even further reduced.

SGI Vizserver provides an alternative to X-based remote visualization while retaining the advantage of transparency to the visualization application. A block diagram of the Vizserver system is shown in Figure 2. The application, completely unaware of the presence of Vizserver, runs on the remote machine as if the remote machine were local. That is, the OpenGL commands employed by the application are intercepted by the Vizserver server and directed to the remote-machine graphics pipeline. The Vizserver server then captures the pipeline output from the frame buffer and transmits the resulting sequence of images to the Vizserver client running on the local machine. Finally, the Vizserver client directs the images to the frame buffer of the local machine for display. The Vizserver system can optionally compress the images transmitted from the Vizserver server to the client to reduce the bandwidth required of the network. We overview the compression options provided by Vizserver next.

2.2 Vizserver Compression

The images extracted from the remote-machine frame buffer can be compressed by the Vizserver server before transmission to the local machine. There are three compression options built into Vizserver: 1) no compression, 2) color-cell compression (CCC), and 3) interpolated-cell compression (ICC). Both the CCC and ICC compression schemes are designed to provide very low latency and require very low computation overhead in order to not detrimentally affect the frame rate of the Vizserver system. Since both of these techniques are *lossy* compression methods, some distortion is introduced; i.e., the images displayed on the local machine after transmission are not exactly the same as those extracted from the frame buffer on the remote machine. However, the visual affect of this distortion appears to be minimal in the Vizserver system, as we discuss below. First, however, we overview both the CCC and ICC methods in some detail.

2.2.1 The CCC Algorithm

Color-Cell Compression (CCC) is a simple compression technique based upon color quantization. It originated in a paper by Campbell *et al.* [2] from SIGGRAPH 1986. It has the advantages of requiring very low amounts of computation with a small latency that is fixed (i.e., not image dependent). As employed in

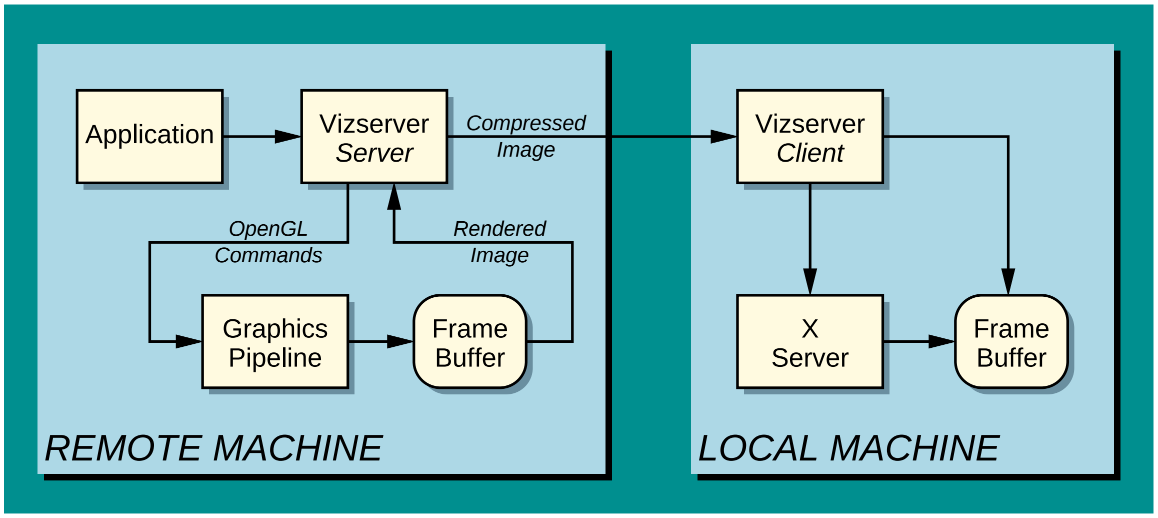


Figure 2: Remote visualization using SGI Vizserver. Rendering takes place in the hardware pipeline of the remote machine, and Vizserver compresses resulting images and transmits them to the local machine.

Vizserver, CCC achieves a fixed compression ratio of 8:1. The operation of CCC is described below.

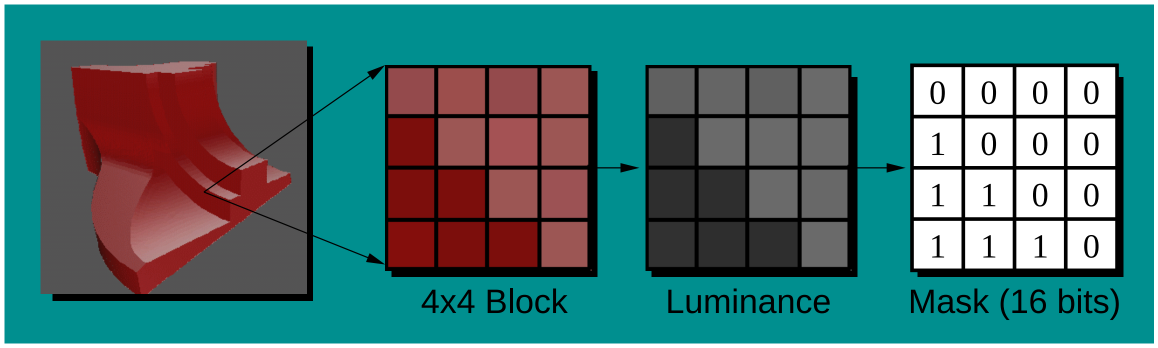
The CCC algorithm works on blocks of color pixels of size 4×4. A 4×4 block is extracted from the image, the CCC algorithm compresses the pixels of that block, and then the algorithm repeats for the next 4×4 block. The operations applied to each 4×4 block are as follows.

Each pixel in the image block is originally represented as an 8-bit red value, an 8-bit green value, and an 8-bit blue value, for a total of 24 bits/pixel. Using these 24 bits/pixel of color information, the luminance of each pixel in the block is calculated. The range of luminance values in the block is found and a threshold within this range is calculated. Each luminance value in the block is then compared to the threshold to create a 4×4 binary mask indicating whether the corresponding luminance value is above or below the threshold. This mask-creation process is illustrated in Fig. 3a.

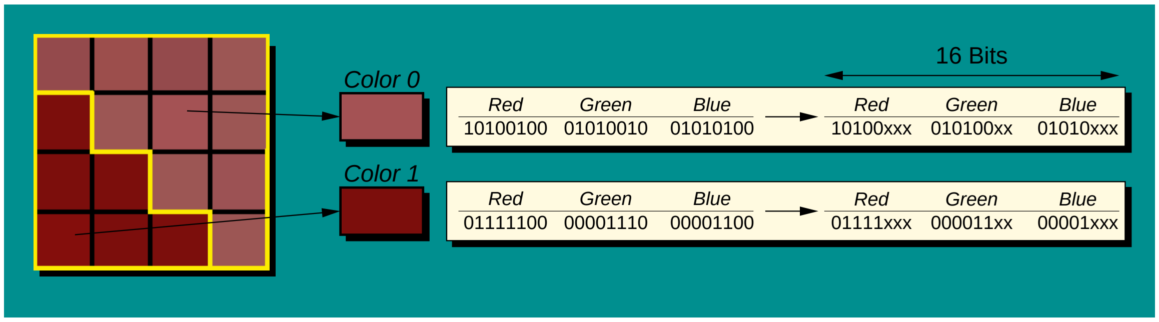
The binary mask is then used to partition the original color image block into two fields—those pixels corresponding to mask value 0 and those corresponding to mask value 1. Using the colors of the pixels corresponding to mask value 0, an “average” color is calculated, shown in Fig. 3b as *Color 0*. A similar process for pixels corresponding to mask value 1 yields *Color 1*. Both *Color 0* and *Color 1* are calculated as 24-bit color values—the final step in the CCC algorithm is to quantize these two colors to 5 bits of red, 6 bits of green, and 5 bits of blue each (this quantization is implemented by preserving only the x most significant bits of the 8-bit color-component value, where $x = 5$ for red and blue, and $x = 6$ for green). Thus, both *Color 0* and *Color 1* are represented using 16 bits each.

On the decoding side, the Vizserver client receives the mask and the 16-bit representations of *Color 0* and *Color 1*. An approximation to the original 4×4 image block is obtained by using the mask to place either *Color 0* or *Color 1* into the block as appropriate. This block is then placed into the image being reconstructed by the Vizserver client, and the process repeats for the next block.

The CCC algorithm is a compression scheme with fixed latency. That is, the amount of time needed to code each 4×4 block is independent of the contents of the block, and is the same for all blocks. As a consequence, the block-coding time depends on only the speed of the remote machine. Likewise, on the client side, the decoding of each block depends only on the speed of the local machine. Since the CCC decoding process is less complicated than the CCC encoding process (decoding does not need to calculate luminance or find *Color 0*/*Color 1*), the latency of the Vizserver client is usually much less than that of the



(a)



(b)

Figure 3: The operation of the CCC compression algorithm. (a) Extraction of the binary mask based on thresholding the luminance values of the pixels in the 4×4 block. (b) Creation and quantization to 16-bits of the two colors used to represent all 16 pixels of the block.

Vizserver server.

The bit rate achieved by the CCC algorithm is as follows. For each 4×4 block of 24-bit color pixels, the CCC algorithm outputs a 16-bit binary mask, 16 bits for *Color 0*, and 16 bits for *Color 1*. Thus, the bit rate for CCC is

$$\frac{16 + 16 + 16}{4 \times 4} = 3 \,\text{bits/pixel}.$$

Since the original image is represented with 24 bits/pixel, the compression ratio is

$$R_{\text{CCC}} = \frac{24}{3} = 8 : 1.$$

2.2.2 The ICC Algorithm

The Interpolated Cell Compression (ICC) algorithm was developed by SGI specifically for the Vizserver application. ICC compression is very similar to the CCC algorithm described above. The sole difference is that the mask specifies four colors rather than two. That is, instead of being a binary mask as in CCC, the ICC mask contains 2 bits for each pixel of the block. This allows the mask to specify a total of 4 possible colors: *Color 0*, *Color 1*, *Color 2*, and *Color 3*. An interpolation strategy based upon luminance values is used to determine which of these four colors is used for each pixel in the block. Like CCC, the ICC algorithm is a fixed-latency compression scheme; since ICC is more complicated than CCC, the ICC latency is more than that of CCC.

The bit rate achieved by the ICC algorithm is as follows. For each 4×4 block of 24-bit color pixels, the ICC algorithm outputs a $4 \times 4 \times 2 = 32$ bit mask, 16 bits for *Color 0*, 16 bits for *Color 1*, 16 bits for *Color 2*, and 16 bits for *Color 3*. Thus, the bit rate for ICC is

$$\frac{32 + 16 + 16 + 16 + 16}{4 \times 4} = 6\,\text{bits/pixel}.$$

Since the original image is represented with 24 bits/pixel, the compression ratio for ICC is

$$R_{\text{ICC}} = \frac{24}{6} = 4 : 1.$$

2.3 Vizserver API

SGI is planning to release to the public an application-program interface (API) for the Vizserver system. Although not available for evaluation at the time of this writing, this forthcoming API is projected to allow the "addition" of custom compression routines to Vizserver. The functionality of the Vizserver API is anticipated to include the following. The API will support the existence of custom encoder and decoder modules, or "plug-ins." On the encoder side, the encoder plug-in will interface to the Vizserver server via the API so to receive an array of color pixels for each frame rendered on the remote machine. The encoder plug-in will be able to dictate the structure of this array (e.g., color depth, colormap, byte alignment, etc.) and will be able to output an arbitrary bitstream back to the API for transmission to the Vizserver client. This flexible functionality should enable users to implement a broad range of image- and image-sequence compression algorithms to endow Vizserver with a large range of performance options. On the client side, a corresponding decoder plug-in will be granted first access to the received bitstream, allowing proper decoding to take place. SGI has planned to make the Vizserver API available with the next release of the Vizserver product, currently scheduled for March, 2000.

3. Vizserver Performance

In order to evaluate the performance of the Vizserver system, we conducted a number of experiments employing a variety of visualization applications. In our experiments, we observed the performance of Vizserver by measuring several key criteria, namely

- the **frame rate** of images rendered at the local machine,
- the **network bandwidth** of the communication occurring between the remote and local machines,
- the **distortion** (image quality) between the images displayed on the local machine and the original images rendered on the remote machine.

Clearly, these criteria are interrelated, with performance for one affecting performance of the others. For example, if distortion is decreased (i.e., image quality improved), one expects to observe a rise in bandwidth. Below, we examine the factors affecting performance for each of these criteria, and present typical measurements from our experimental observations. Table 1 gives the details of the architectures of the remote and local machines we used in the following experiments.

3.1 Frame Rate

One of the key components of the overall performance of Vizserver is the frame rate of the images as displayed on the local machine. Clearly, the local-machine display frame rate is dependent on many factors, including the current conditions of the remote machine, the local machine, and the network in between.

Table 1: Remote and Local Machines Used in Experimental Observations

	Remote Machine	*Local Machine*
Machine Type	Onyx2	O2
Processor Type	R10000	R5000
Processor Speed	250 MHz	200 MHz
Number of Processors	8	1
Primary Cache Size	32 kB	32 kB
Secondary Cache Size	4 MB	1 MB
Main Memory Size	4 GB	256 MB

Table 2: Observed Frame Rate and Network Bandwidth for a Frame Size of 640×480 over a 100 Mbps Switched Ethernet Network

Compression Method	*Frame Rate (frames/sec)*	*Network Bandwidth (Mbps)*
CCC	12	11.1
ICC	10	18.4
None	8	59.0

The visualization application, unaware of the presence of Vizserver, renders images at its normal, application-dependent frame rate using the remote-machine pipeline. For each compression option employed, the encoding of each 4×4 block takes a fixed, machine-dependent amount of time on the remote machine. For the no-compression option, a negligible amount of time is required for encoding; the CCC and ICC compression options require a certain nonnegligible amount of time to perform the compression, with ICC taking longer than CCC. If the Vizserver server encodes frames at a rate that is less than the rate that frames are rendered through the graphics pipeline, frames are in effect rendered but discarded (i.e., not encoded and thus not transmitted to the client). That is, once the Vizserver server finishes encoding a frame, it waits until the next full frame is available from the output of the graphics pipeline and grabs that next frame for encoding; any frames rendered during the encoding of the current frame are lost. Since larger frames take longer to compress and encode, increasing the size of the rendered frame will reduce the rate that frames are transmitted to the client.

Table 2 shows the frame rates as measured on a local O2 machine when Vizserver was employed over a 100Mbps switched Ethernet with a frame size of 640×480. The frame-rate figures in this table are an estimated time-average frame rate as observed by using the SGI IRIX `osview` command on the O2 machine to measure the number of `swapbuf` completes over a one-second interval, which gives a rough estimate the number of frames per second displayed.

3.2 Bandwidth

In addition to depending on the encoder speed, the frame rate observed at the local machine also depends on the bandwidth that can be supported by the network. Specifically, frames may be lost due to network congestion. In Table 2, we present local-machine frame-rate results typical of our experimental observations;

we also show corresponding network bandwidths. We estimate the network bandwidth directly from the observed local-machine frame rate (this is possible since all frames displayed at the local machine must have been transmitted across the network). The estimated network bandwidth, in bits/second, is

$$B = \frac{f \times M \times N \times 24}{R}$$

where f is the observed local-machine frame rate, M and N give the width and height, respectively, of the frame, and R is the compression ratio employed ($R_{\text{CCC}} = 8$, $R_{\text{ICC}} = 4$, $R_{\text{none}} = 1$). In reality, the true bandwidth may differ slightly from this estimate due to inaccuracy in the measuring of frame rate as a time average; however, we have generally found a good agreement between this estimate and direct observation of bandwidth using the IRIX `netstat` command.

In Table 2, the network bandwidth was estimated using the above formula and the observed frame rate. The natural frame rate for the application (i.e., the frame rate observed when the application runs locally on the Onyx2 machine) was 36 frames/sec. We see from these results that, when CCC compression is employed, Vizserver manages to display roughly 12 frames/sec on the local machine, or about $\frac{1}{3}$ of the frames rendered on the remote machine. However, when ICC is used, the frame rate drops slightly to 10 frames/sec. This drop is as expected since, being a slightly more computationally complex compression algorithm, ICC will require slightly more time to encode each frame, resulting in a greater number of frames lost due to encoder latency than when CCC is used. On the other hand, when no compression is used, the encoder speed is actually greater than that when CCC or ICC compression is used. However, rather than resulting in a greater frame rate as might be initially expected, an *slower* frame rate is observed for the no-compression case. The reason for this apparent discrepancy lies in that, when no compression is employed, much greater bandwidth is required of the network. Since an unloaded switched Ethernet can typically achieve throughput of only 30% to 60% of its rated bandwidth (i.e., a 100 Mbps switched Ethernet normally can operate at only 30-60 Mbps in practice due to Ethernet packed overhead and latency within the TCP/IP stack of the OS-level network stack), the network becomes saturated (at roughly 59.0 Mbps according to the results of Table 2) in our experiments unless compression is used. Consequently, for the no-compression case, frames are due to network congestion despite the fact that the increased encoder speed causes fewer frames to be lost due to encoder latency. The net result is a frame rate slower than when CCC or ICC is used. To summarize, the critical factor in frame-rate performance of the Vizserver system is encoding speed when CCC or ICC compression algorithms are used, but the available network bandwidth becomes the critical factor when no compression is used.[2]

3.3 Distortion

As observed above, the ICC and CCC compression algorithms are *lossy* techniques—due to quantization of colors, the images reconstructed on the local machine necessarily differ from the images that were originally produced in the graphics pipeline of the remote machine. However, the amount of this distortion that is visibly perceptible appears to be little or none for the Vizserver system. Of course, when no compression is used, no distortion is introduced, and the Vizserver implements *lossless* transmission of the images to the local machine.

As part of our experimental evaluation of Vizserver, we measured the image distortion quantitatively, as well as investigated the visual affects of this distortion. In short, there was no visible distortion apparent in any of our experimental observations. Quantitatively, we used a signal-to-noise ratio (SNR) derived

[2]During the final stages of the preparation of this report, it was brought to our attention that a bug exists in the current version of the Vizserver client that detrimentally affects frame-rate performance of O2 local machines. Specifically, an inefficient mode of frame-buffer write is employed on O2 machines which results in a significantly slower frame rate observed for these clients. According to SGI, this bug should be fixed in the next release of Vizserver, bringing frame-rate results up to around 30 frames/sec for CCC compression of 640×480-size frames; we have been unable to verify these figures as of this writing.

from the 1964 CIE color-space distance measure [3] (as calculated in the CIE Modified UCS space) as a distortion metric. Quantitative distortion results from our experiments are presented in Fig. 4, where we used a 1238×717 image from an ocean-current visualization. The actual images from which these quantitative figures were generated are also shown. Quantitatively, we see that, as expected, CCC introduces slightly more distortion, resulting in a relatively lower SNR, as compared to ICC; however, no difference in performance is observed visually, as is evidenced by the images of Fig. 4.

4. Conclusions

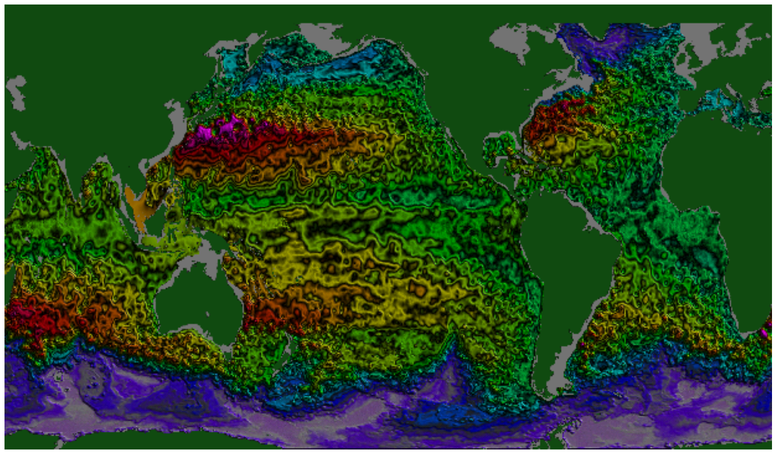
Our overall conclusion on the effectiveness of SGI Vizserver is that, in certain settings, Vizserver can provide useful remote-visualization capability. However, it is not the "perfect" remote-visualization tool, and has several shortcomings that might render it impractical in certain applications. Below we summarize our observations on Vizserver's strengths and shortcomings.

Vizserver Strengths:

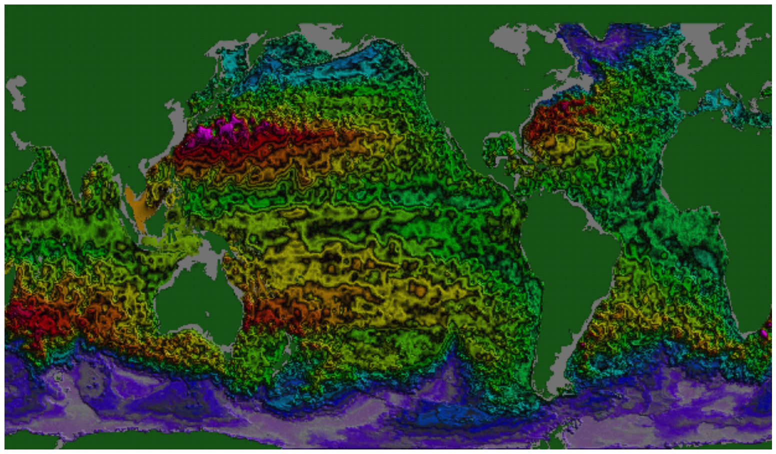
- **Transparent to application**—Vizserver can be deployed without any modification whatsoever to the visualization application.
- **High frame rate achieved**—Because the two compression methods (CCC/ICC) built into Vizserver involve extremely lightweight computation, a minimal number of frames are lost due to encoder latency. It is likely that any other compression algorithm (i.e., custom algorithms implemented via the API), will be computationally more costly and result in a slower frame rate.
- **Visually lossless**—The distortion introduced by CCC/ICC is is not usually perceivable; consequently, Vizserver produces excellent visual quality at the local machine.
- **API**—The flexibility represented by the forthcoming Vizserver API will allow Vizserver to be customized to better fit users' needs.

Vizserver Shortcomings:

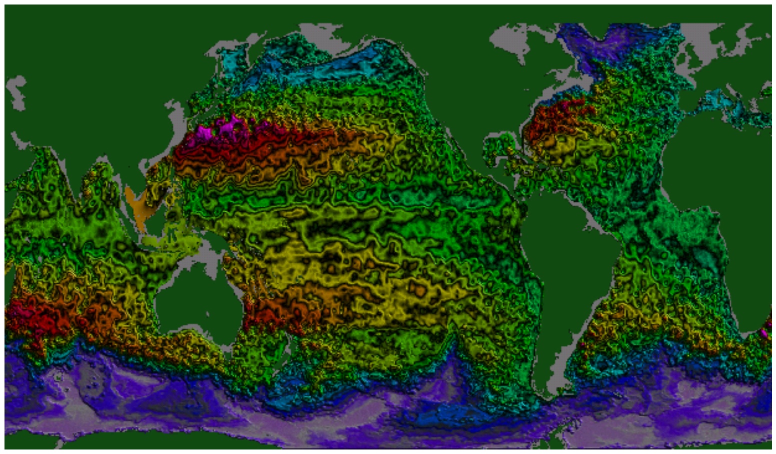
- **Limited user control**—The main weakness of Vizserver is that there exists limited user control over the frame-rate/bandwidth/quality performance tradeoff. This limitation is inherent in that Vizserver supports only two built-in compression routines. Thus, the user can control the frame rate, bandwidth, and image distortion only indirectly—by adjusting the frame size, and/or by selecting one of only three possible compression settings. Additionally, both the ICC and CCC algorithms were observed to provide visually lossless compression performance; therefore, there is little gain to be had in selecting ICC over CCC, since ICC requires larger bandwidth.
- **Bandwidth is too large**—The bandwidth required by Vizserver appears to be too large for truly remote users. For instance, in the results presented in Table 2, we see that, even when using CCC, which had the lightest bandwidth requirement of the three compression choices, the bandwidth was still over 10Mbps for the frame size (640×480) used in our experiment. Although this bandwidth requirement is perhaps feasible for users who are situated on the same LAN as the remote machine, it is too high for truly remote users—those users geographically separated from the remote machine by many miles and connected only through links with bandwidth on the order of T1 ($\sim$1.5 Mbps). Such remote users will likely find that Vizserver requires impractically small frames sizes or unreasonably slow frame rates.



(a)



(b)



(c)

Figure 4: Example images from the Vizserver system, frame size = 1238×717. (a) Original image as rendered at the remote machine. (b) Image reconstructed at the local machine using CCC compression algorithm, CIE SNR = 10.8 dB. (c) Image reconstructed at the local machine using ICC compression algorithm, CIE SNR = 12.6 dB.

- **Proprietary**—Vizserver is proprietary software controlled by SGI. The Vizserver server relies on that fact the SGI controls the underlying OS, permitting Vizserver proprietary access to the frame buffer on the remote machine. Additionally, the Vizserver client systems are limited to those which SGI has chosen to support. Currently, Vizserver clients exist only for SGI machines, although Solaris, Windows, and Linux clients are planned.

In summary, Vizserver appears to be a reasonable choice for remote-visualization applications deployed on SGI architecture connected via 100 Mbps Ethernet LANs. However, truly remote users with significantly less bandwidth will see much less utility, while application portability will be restricted by Vizserver's proprietary nature.

References

[1] C. Ohazama, "OpenGL Vizserver White Paper," White Paper, 1999, Silicon Graphics, Inc.

[2] G. Cambell, T. A. DeFanti, J. Frederiksen, S. A. Joyce, L. A. Leske, J. A. Lindberg, and D. J. Sandin, "Two Bit/Pixel Full Color Encoding," in *Computer Graphics (Proceedings of SIGGRAPH 86)*, D. C. Evans and R. J. Athay, Eds., Dallas, TX, August 1986, vol. 20, pp. 215–223.

[3] A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall, Englewood Cliffs, NJ, 1989.