OpenGL Vizserver 3.1
Application Transparent Remote Interactive Visualization and Collaboration

Michael Brown, Yochai Shefi-Simchon und Allan Commike

sgi Hannover
Ahrensburger Str. 5, 30659 Hannover
christianm@sgi.com

Abstract: The OpenGL Vizserver computing solution is one of the key enabling technologies in a Visual Area Network (VAN). These solutions distribute the interactive graphics generated by powerful visual servers to remote, light-weight clients. Visual Area Networking enables remote users and distributed teams to manipulate and visualize large data sets at rates that are tens to thousands of times faster than is possible with desktop systems.

1 Introduction

OpenGL Vizserver allows users of remote workstations, laptops, and even wireless tablet computers to use existing unmodified applications to access and control the power of SGI Onyx family visualization systems and collaborate with one another using existing visualization applications based on X11 and the OpenGL API. The SGI Onyx family systems can be deployed as shared group, departmental, or enterprise visual servers in the manner that best matches the needs of an organization.

An added benefit of this approach is that network bandwidth to clients remains constant while the data set sizes and computational complexity of the visualizations can increase without bound. Since only the visual servers need to be upgraded to meet increased visualization demands, desktop upgrades are simplified, and data-copy-driven demands for network upgrades are eliminated. Centralizing the rendering resources also allows a centralization of data and an enhancement of security and alleviates the need for data replication to local desktops. OpenGL Vizserver can be used with today's applications without change, allowing users to immediately use VAN technology and see results right away.

2 OpenGL Vizserver Architecture

OpenGL Vizserver software has two primary components: a server and a client. The server runs on SGI graphics supercomputers managing graphics resources (e.g., graphics pipelines) and monitoring the visualization application activity. Once a visualization application is started, the OpenGL Vizserver server will assign the application the requested graphics resources and begin serving the application-rendered frames to the

OpenGL Vizserver client. This visual serving is the basis of the OpenGL Vizserver technology and SGI's VAN strategy. Only after a visual application has rendered a frame will OpenGL Vizserver intercede and capture that frame. The captured frame can be a small fraction of the original data set size and orders of magnitude less complex, because only the pixels associated with the screen representation of the data are captured.

Each frame captured is compressed using either lossy or lossless data compressors that take advantage of interframe coherency to minimize the amount of data sent to the OpenGL Vizserver clients. Once compressed, the image stream is sent to the client. An OpenGL Vizserver client is a lightweight application that reads the image stream from the OpenGL Vizserver server, uncompresses the stream, and displays the uncompressed image on the client computer. The OpenGL Vizserver client directs all user interaction back to the OpenGL Vizserver server, creating a seamless visualization environment on the client as if the user were interacting locally with the SGI graphics supercomputer. The OpenGL Vizserver client runs on a variety of operating systems including IRIX, Linux, Windows, or Solaris operating systems, and the client system need not have extensive graphics or computational power.

The OpenGL Vizserver architecture is designed to allow multiple clients in a geographically dispersed network to create a fully collaborative work session. This architecture provides the capability to gather input from any OpenGL Vizserver client and treat that input as if it were the local input that has direct control of the visualization application. Passing control from client to client can be strictly moderated by a session master or allowed to happen in an unmoderated, ad hoc fashion. The moderator's responsibility is to explicitly determine which users can obtain application control. Without a moderator, application control is passed from user to user as requested. This control mechanism provides the flexibility necessary to conduct formal design reviews or unstructured brainstorming sessions. Multiple users can collaborate on the same visualization, each instantly seeing the newly updated visualization take form as the clients interact with the visualization application. Because the application is always working with local input and output, no changes are required to use existing standalone applications in a collaborative OpenGL Vizserver environment. Each user can join or leave a collaborative work session at any time allowing remote users to participate in the session as needed.

2.1 Implementation Details

The philosophy behind OpenGL Vizserver is one of Execute and Monitor. To that end, it monitors all calls to the OpenGL and X11 application libraries, relying on the native OpenGL and X11 implementations as if OpenGL Vizserver were not running. Information gathered via monitoring OpenGL and X11 is passed to the OpenGL Vizserver run-time system that handles all aspects of running OpenGL Vizserver. All API calls into the OpenGL and X11 libraries occur as expected, in the order expected, without side effects. An application that runs correctly against the native libraries will also run correctly in an OpenGL Vizserver installation.

In order for OpenGL Vizserver to effectively monitor OpenGL and X11, it inserts a wrapper around the libGL.so and libX11.so API entry points. These wrapper libraries are trans-

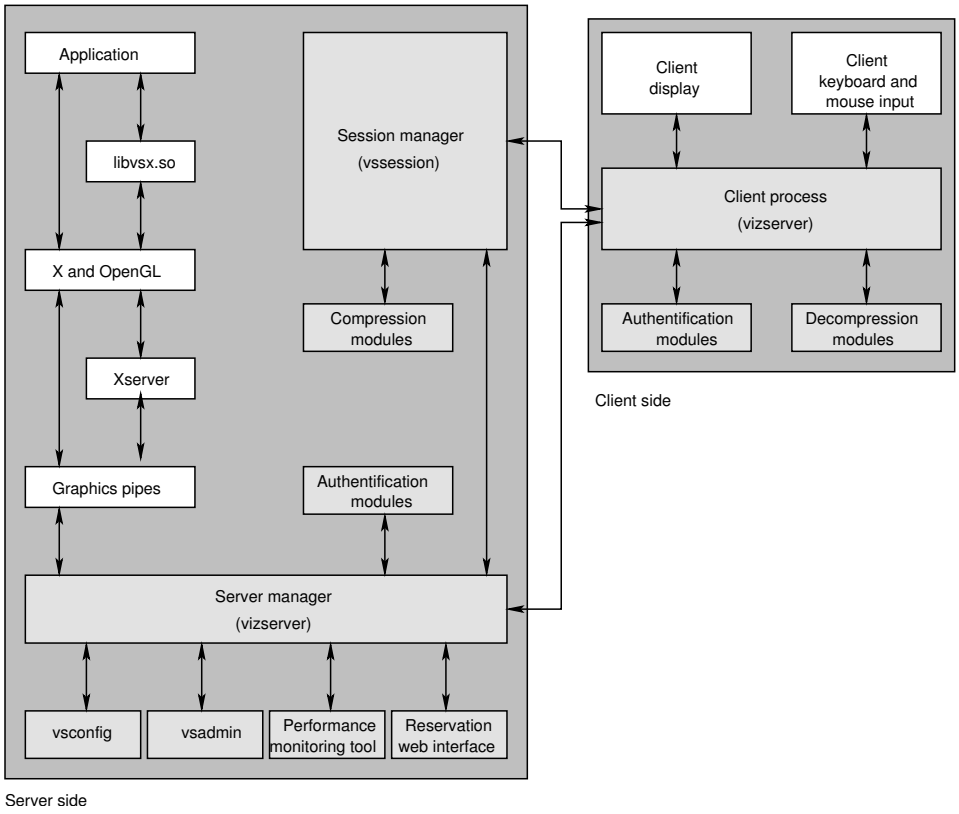


Figure 1: OpenGL Vizserver Architecture

parently used when an application is started in the OpenGL Vizserver environment. Applications that are statically linked or those that dynamically load libGL.so and libX11.so will not transparently use the wrapped libraries and will not function correctly in the OpenGL Vizserver environment. There are wrapper libraries for OpenGL and X11 that target o32, n32, and 64-bit MIPS ABIs. Each of these libraries will work with a singlethreaded application or a multithreaded application using sproc, pthread, or fork thread programming models. An application running in the OpenGL Vizserver environment will use these wrapper libraries for libGL.so and libX11.so in place of the native libraries. The wrapper libraries enable OpenGL Vizserver to monitor select OpenGL and X11 API calls by augmenting the API entry points with additional code that records the information needed by OpenGL Vizserver to effectively process the rendered scenes. Those API entry points that do not need to be monitored by OpenGL Vizserver are called directly by the application as if OpenGL Vizserver were not in use. The monitored API entry points first call the native API entry points and then record the information that is needed by OpenGL Vizserver.

Recording data from the monitored libGL.so and libX11.so API entry points is handled via functions within the libvsx.so OpenGL Vizserver run-time library. This run-time library

encodes a protocol that allows the library to communicate with the OpenGL Vizserver run-time system, vssession, which provides all of the OpenGL Vizserver functionality. The run-time system runs as a separate process in the system in order not to introduce side effects into the monitored application. The protocol used between the libvsx.so and the OpenGL Vizserver run-time system communicates through a shared-memory segment that is shared between the run-time library and server. This approach ensures a minimal performance impact of monitoring API entry points. Each monitored entry point first calls the native API entry point, allowing native calls to process data as normal. It then writes data to the shared-memory segment using the OpenGL Vizserver communication protocol and signals the run-time server that new data is available. The monitored API entry point will not return to the application until the OpenGL Vizserver run-time system acknowledges that it has recorded the new information. The monitored OpenGL and X11 API entry points allow OpenGL Vizserver to record events of interest in the lifetime of an application. These events of interest include window creation, window movement, mouse movement, buffer swaps, window destruction, application exit, and others. As each event occurs, OpenGL Vizserver takes the appropriate action based on the number and types of clients that are connected across the VAN. One of the primary events of interest is a buffer swap in a window that OpenGL Vizserver is monitoring. This event triggers OpenGL Vizserver to read back the application's frame buffer, compress the resulting image, and write that image out to the network for each client. This pipeline ensures that the readback and compression steps are performed only once for all clients. The readback is done inside the monitored API entry point that triggered the event, not in the OpenGL Vizserver run-time system. This ensures that the graphics pipeline will not have to context switch between the run-time system and application each time a readback is triggered.

On the client side, the data is read in from the network, uncompressed, and then drawn to the local graphics display. OpenGL Vizserver monitors the region of interest associated with each window in use by an application running in the OpenGL Vizserver environment. This region of interest includes the OpenGL rendering area and X11 renderings including menus, icons, and dialog boxes. The entire area of interest is read back from the frame buffer, compressed, and shipped to the client. Figure 1 shows the architecture of this system.

Scalability

OpenGL Vizserver has been designed from the beginning to be scalable, supporting clients that range from an SGI Onyx family-class graphics supercomputer to clients with little computational and graphics power. When collaborating over a VAN, a heterogeneous mix of clients running various operating systems may be in use. To ensure that client rendering does not become a bottleneck, each client utilizes native drawing mechanisms that deliver the highest performance for that particular client platform.

2.2 OpenGL Vizserver Pipeline

As briefly described above, the basic OpenGL Vizserver pipeline is:

The OpenGL Vizserver pipline is implemented with an FIFO queue at each stage of the pipeline, allowing each to run independently of the others. The latency between the server

and client is the sum of the time for each stage, and the frame rate is the time taken by the longest stage.

The client and server are symmetric with respect to the network. Each operation on the server has a corresponding opposite operation on the client that happens in reverse order. The last operation performed by the server is a network write, while the first operation performed by the client is a network read. The following sections describe individual stages of the pipeline; each section describes both the client and server operation.

2.2.1 Readback (Server) and Draw (Client)

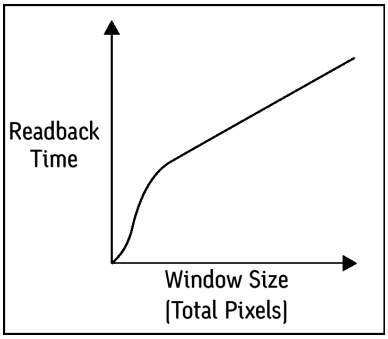


Figure 2: Readback Performance as a Function of Window Size

OpenGL Vizserver determines a frame has been completely drawn when an application issues one of glxSwapBuffers(), glFlush(), or glFinish(). These OpenGL API calls are monitored and, when issued, trigger OpenGL Vizserver to read the completed frame from the graphics frame buffer into main memory. Only the portion of the frame buffer bounded by the region of interest that is used for rendering will be read back. This region of interest includes the window manager frame along with other X11 GUI items and the OpenGL rendering area. A readback of both the left and right buffers is performed if the visual server running OpenGL Vizserver supports stereo and the OpenGL rendering area that the application is using corresponds to a stereo visual. The time taken for this readback depends on the size of the application window, not the complexity of the scene being rendered. The time taken for a readback can be seen in figure 2.

The draw operation on a client is a simple operation that uses the native windowing system to render the pixels to the appropriate region of the client screen. This operation reproduces the image that is displayed on the OpenGL Vizserver server without the need for the client graphics system to have high-performance graphics capabilities. Stereo rendering is reproduced if the client supports stereo. Clients that do not support stereo will only see one of the buffers provided by the visual server running OpenGL Vizserver. The draw operation on the client uses the native client's hardware-rendering architecture to

achieve best performance. Since both the GUI and rendering area are read back on the OpenGL Vizserver server, the client does not need to perform window management or GUI tasks other than opening a window in which to render the pixels provided by the OpenGL Vizserver server. The look and feel of an application on the client will be exactly the same as on the server.

2.2.2 Spoiling

The frame rate seen by OpenGL Vizserver clients depends not only on the rendering speed of the OpenGL Vizserver server but also on the readback/compression pipeline on the OpenGL Vizserver server and the decompression/ draw pipeline on the OpenGL Vizserver client. These two pipelines can introduce delays into the system when the rendering performance of the OpenGL Vizserver server generates frames of data faster than either of the two pipelines can process them. The OpenGL Vizserver server contains a queue of frames that are ready to be sent to an OpenGL Vizserver client. Spoiling is a mechanism that allows the OpenGL Vizserver server to remove a frame from the queue and replace that frame with the next one available from the application. In the case of stereo rendering, a consistent stereo image is preserved by removing frames from both the left and right buffers.

When spoiling is on, pipeline bottlenecks do not affect the frame rate of the application running under the OpenGL Vizserver environment on the server machine. In this case a full pipeline will cause the first frame in the queue to be dropped and replaced with the current frame generated by the application. Without spoiling, OpenGL Vizserver will pause the application until the OpenGL Vizserver pipeline drains enough to fit the next frame into the OpenGL Vizserver output queue. An OpenGL Vizserver client running with spoiling turned on will not see the frames that are spoiled but instead will always see the most current frame generated by the application. Frame rate on the client side in this case will be gated by the bandwidth of the OpenGL Vizserver pipeline. That is, the frame rate seen on the client will be that of the latency in the OpenGL Vizserver pipeline or that of the client application, whichever is greater.

2.2.3 Compress (Server) and Decompress (Client)

OpenGL Vizserver uses programmable compression modules to compress/decompress frames of the rendered scene. There are five standard modules (CCC, ICC, SCC, SICC, LCC) and an API that provides the capability to develop new modules with user-defined functionality. Each compression module has the capability of taking advantage of frame-to-frame coherency inherent in most visualizations by implementing an interframe compression scheme where only the changing portions of each frame are compressed and sent to the clients.

The compression module API allows the creation of compression modules that take into account domain-specific data formats and data structure that cannot be inferred from

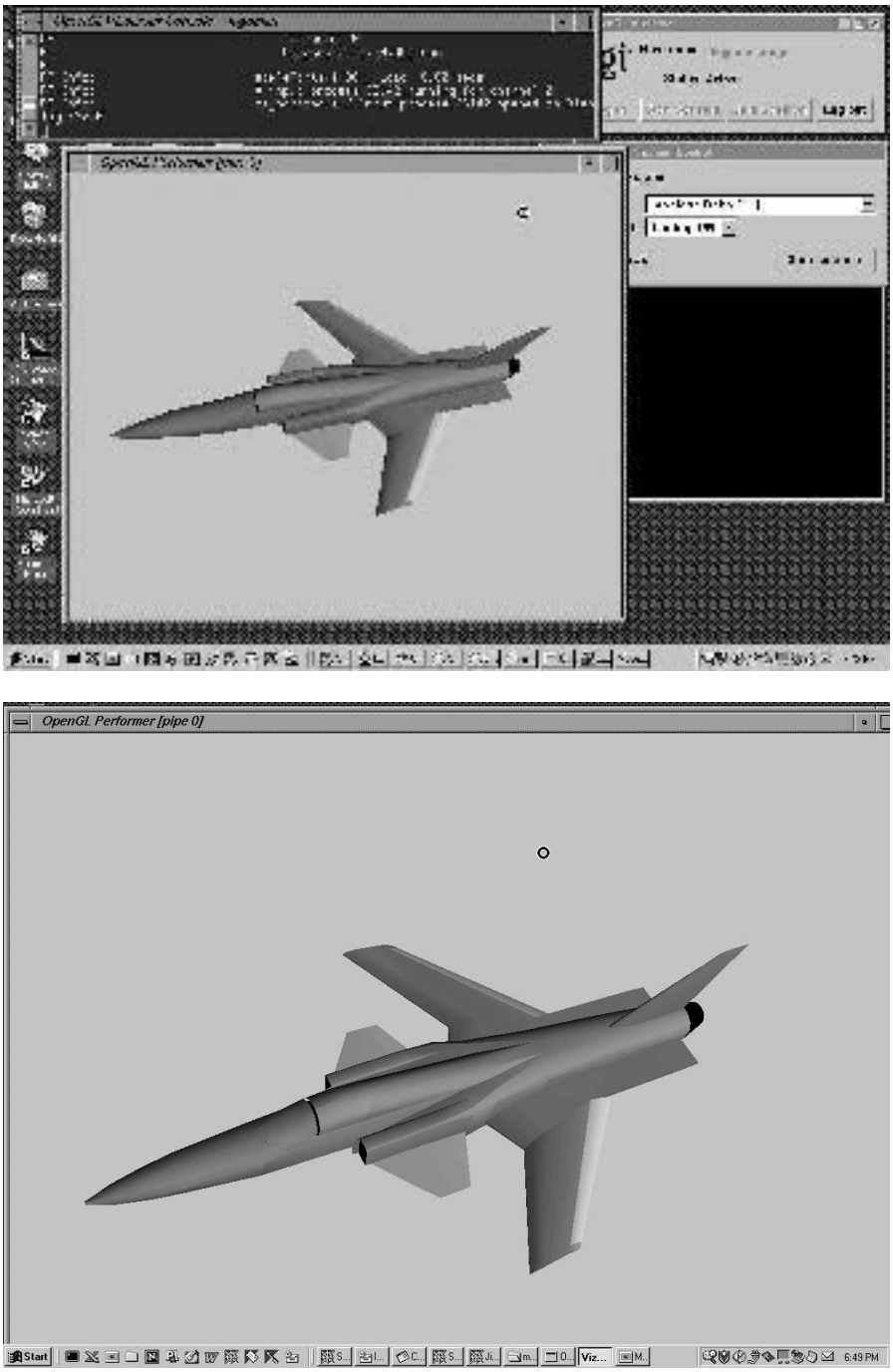


Figure 3: Image Capture of Application Used to Generate Performance Results. The upper image shows a 640x512 interactive window on a standard PC laptop display, and the lower image shows a 1024x768 window on a standard PC laptop display.

general-purpose compression algorithms. This knowledge of the data can enable a custom compression module to attain higher compression rates and quality than the general-purpose compressors supplied with OpenGL Vizserver.

The CCC, ICC, SCC, and SICC OpenGL Vizserver compression modules implement lossy compression algorithms. These four schemes are derived from the Block Truncation Coding (BTC) algorithm that compresses a 4x4 pixel block down to two colors plus a 4x4 pixel mask. The variation between the CCC and ICC compressors lies in the number of colors identified by the pixel mask. In the case of CCC, the mask identifies two colors. The ICC mask identifies four colors. In both cases, the color components are converted to 5 bits of red, 6 bits of green, and 5 bits of blue, totaling 16 bits per pixel. The compression ratio for these two schemes is at least 8:1 for CCC and at least 4:1 for ICC. SCC and SICC are scaled versions of CCC and ICC, respectively. Scaling reduces the image data 75% by scaling the data before applying the CCC or ICC algorithms. Scaled CCC and scaled ICC algorithms have compression ratios of at least 32:1 and at least 16:1, respectively. More information on these compression algorithms can be found in [1] and [2].

In addition to lossy compressors, there is also a lossless compression module called LCC.This preserves the original image quality while still saving bandwidth. In many cases the savings is as high as 4x without any reduction in image quality.

Each compression module uses two threads to perform the compression. On machines with higher CPU counts, each thread will run on a separate processor, speeding up the compression time by almost 2x.

2.2.4 Interframe Compression

The interframe compression framework of OpenGL Vizserver adds frame-differencing calculations to the built-in compression modules. Doing so allows redundancy in the image stream to be quantified and analyzed. Each compression module implements a selective block compression/decompression algorithm compressing only those portions of the image that changed since the last frame was rendered. After frame differencing, only blocks that have changed are sent to the compression modules. If the new frame is exactly the same as the old one, only four bytes of information is sent to the OpenGL Vizserver client. This saves processing time and bandwidth on both client and server. Interframe compression is performed separately on the OpenGL rendering data and the X11 GUI data. The X11 GUI data is sent through the frame-differencing engine and then to the lossless compression module, not to the user-selected module. Since an application's GUI is relatively static, very little bandwidth is needed for the GUI portions of an application in an OpenGL Vizserver environment.

The interframe compressors cache the image stream, allowing frame differencing between current and prior frames in addition to reducing network bandwidth by not sending data that remains unchanged. A cached image stream exists on both the OpenGL Vizserver server and OpenGL Vizserver client, increasing memory usage by a factor of the number of drawable windows being visually served and the size of each window. A stereo window is compressed as two separate images, one for the left eye and the other for the right,

increasing memory usage for stereo windows to twice that of non-stereo windows. The frame-differencing engine is run separately on each of the left and right stereo images. When combing compression techniques with inter-frame coherency in applications that have mostly static images, compression ratios can increase an order of magnitude or more. The theoretical highest compression ration is 1536:1.

A good rule of thumb is that the bandwidth savings is 4x for the interframe component OpenGL Vizserver compression, with a reduction of the net bandwidth needed between OpenGL Vizserver server and client. However, applications that change every pixel every frame, such as an image-roaming application, will see limited benefits from the use of interframe compressors. The following compressor performance and bandwidth figures were generated by rotating a simple, solid model using the OpenGL Performer real-time graphics API and displaying the results on remote systems over a private 100Base-TX network.

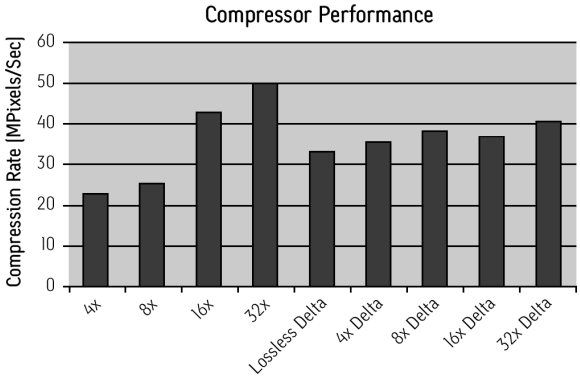


Figure 4: OpenGL Vizserver 3.1 Compression Rates.

Figure 3 shows images of the 640x512 and 1024x768 test cases and shows that there are areas of constant background color that benefit from interframe compression. Because the application requires very little drawing time, these frame rates represent maximum numbers that can be achieved with OpenGL Vizserver 3.1. Applications that run at less than maximum obtainable speeds will generate lower compressor performance because there is less information to compress.

Figure 4 shows the overall compression rates achieved with different standard and interframe compressors.

Figure 5 shows the amount of network bandwidth required to drive 10 frames per second of performance over a network using compressors supplied with OpenGL Vizserver 3.1. These figures show how network bandwidth can be reduced by increasing the level of image compression used. The potential impact of interframe compressors is also shown.

2.2.5 Custom Compressors

It is possible to add custom compression modules to OpenGL Vizserver. A new compression module must be written in C++ and derived from the vsCompressor class that is shipped as part of OpenGL Vizserver. Some of the core methods of the new class that must be implemented include the compress(), expand(), and getMaxCompressedSize() methods. There are several macros that ease the implementation task. The resulting compression module must be built into a shared library for each operating system platform that will be supported. The new module appears to the end-user OpenGL Vizserver GUI once the module is placed in the /usr/vizserver/compress/lib32 directory on both the OpenGL Vizserver server and on each OpenGL Vizserver client that will use the custom compression module. Examples of custom compression modules ship with the vizserver_dev package.

2.3 Collaboration

OpenGL Vizserver enables multiple users in geographically diverse locations to collaborate on the same data set as if they were all in the same room. Since collaboration is an integral part of OpenGL Vizserver, there is little additional configuration needed to configure such a session. The first client to connect to OpenGL Vizserver is considered the session master. This session master provides initial compression settings, validates remote users joining the session, and determines the application control policy used for remote users.

Users can join or leave a collaborative session at any time, although the session ends when the session master exits. Application control is dictated by the policy set by the session master. The session master can either moderate the collaboration by approving all requests for application control or allow an unmoderated session where approval for application control is not needed. Only a single user can have application control at any one time. All users in a collaborative session will have a synchronized view of the scene being visualized. In order to achieve this synchronization, OpenGL Vizserver must serve frames at the rate of that the slowest client can accept.

3 Configuring OpenGL Vizserver

When determining the network bandwidth requirements of OpenGL Vizserver, many aspects of the network must be considered. Primarily, the size of the image that the OpenGL Vizserver server will send to OpenGL Vizserver clients determines the network bandwidth needed. The uncompressed network bandwidth can be calculated by multiplying the size of the window in pixels (width times height) by three bytes for each pixel (one byte each for the red, green, and blue components of the pixel). Each compression module will reduce the bandwidth by a different amount, which will always vary with the specific scene being rendered, although the average compression ratio for each compression module is useful for sizing purposes (see figure 4).

In addition to image size, the physical network condition, routing topologies, switch bandwidths, and network congestion leading to packet retransmits must all be considered. The

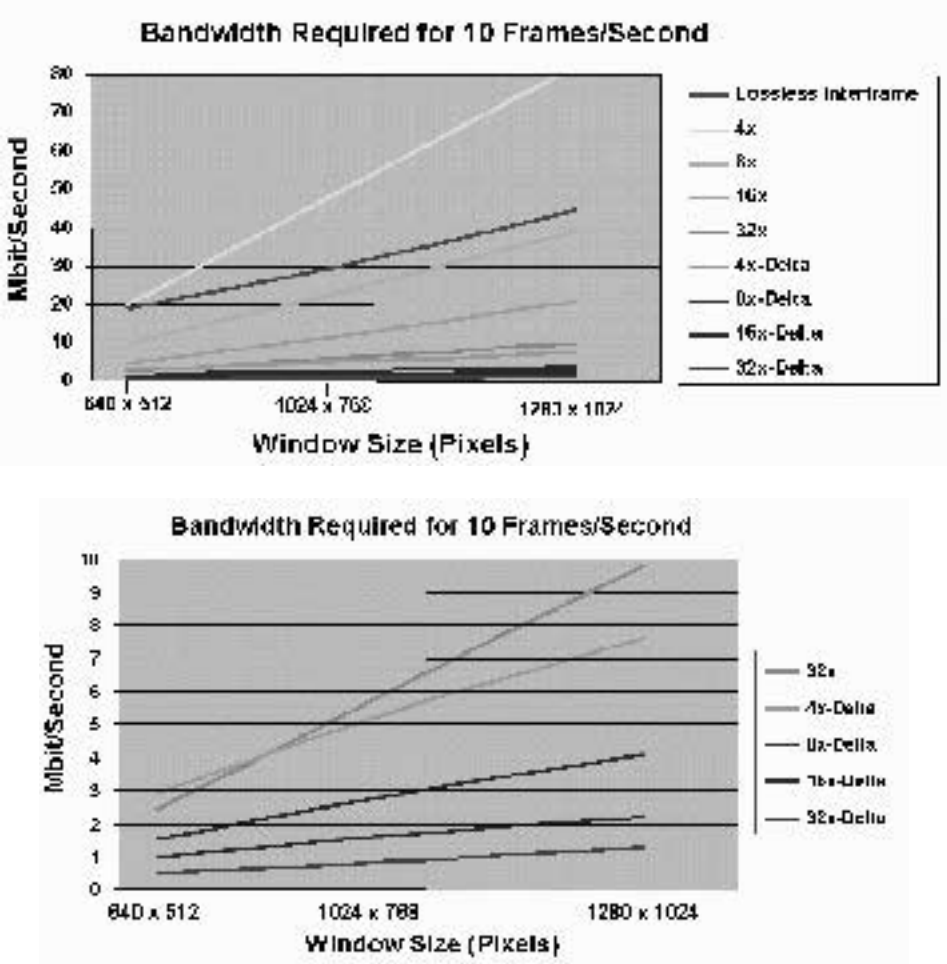


Figure 5: Bandwidth Requirements.
Top: OpenGL Vizserver Compressiors delivering 10 frames per second.
Bottom: Larger scale of low-bandwidth range.

user should keep these overheads in mind when calculating expected bandwidth or frame rate.

Processor Utilization

The OpenGL Vizserver server is a pipeline wherein the frame buffer readback, image compression, and network write each run independently. This architecture means that up to three frames may be in-flight at once. Optimal configuration should allow for a CPU to be utilized for the readback, another for the network write, and a CPU for each of the two threads of the compression module, for a total of four processors. Additionally, a

collaborative session requires that a network write be performed for each collaborative user. Optimally, a CPU can be dedicated to each remote user. This is in addition to the number of processors needed for the application.

4 Application Compatibility

OpenGL Vizserver is a transparent technology, meaning that an application is unaware if it is being visually served. Even though OpenGL Vizserver is transparent, there are a few areas where application design can affect the compatibility with OpenGL Vizserver. Since the OpenGL Vizserver libraries are run-time bound, applications must directly link against the libGL.so and libX11.so libraries. Applications that dynamically load graphics libraries using the dlopen() system call will disable the ability of OpenGL Vizserver to monitor OpenGL and X11 API calls, and thus the application will not perform as if it were executing under OpenGL Vizserver. In this case, an application could dynamically open the libvsx.so library and call the appropriately monitored functions. Additionally, applications that run suid will not function under OpenGL Vizserver, because of security reasons.

The OpenGL Vizserver server monitors the application's use of OpenGL by watching for a signal denoting the end of a frame. This signal is denoted by the application issuing one of the following OpenGL API calls:

- glXSwapBuffers()
- glFlush()
- glFinish()

Applications that do not issue these calls, such as those that draw only to the front OpenGL buffer, will not trigger OpenGL Vizserver to read back the rendered scene starting the flow of data to the OpenGL Vizserver client. By inserting a glFlush() statement in the application where OpenGL Vizserver should read back the scene, applications that utilize such techniques as front buffer OpenGL rendering will be fully compatible with OpenGL Vizserver.

Applications that are based only on X11 will not have calls to any of the above functions that trigger an OpenGL Vizserver readback. OpenGL Vizserver detects this case and uses the SGI_CAPTURE X11 extension to capture X11 rendering. An application can detect if OpenGL Vizserver is running by looking at the glXExtension-String of the client GLX extensions for the SGI_vizserver string. OpenGL Vizserver creates shared-memory segments for use in communicating between libvsx.so in the application's address space and the OpenGL Vizserver run-time system, vssession. Applications cannot assume that all shared-memory segments and queues that exist are owned by the application. In addition, the maximum size and number of shared-memory segments are reduced by OpenGL Vizserver usage of these resources. OpenGL Vizserver allocates approximately 4MB of shared memory and an 8-16MB memory mapped file per graphics pipe.

5 Environments

5.1 Multichannel and Single-Channel Environments

OpenGL Vizserver does not resize the dimensions of the image stream when the size of the OpenGL Vizserver client display is a different size compared with the OpenGL Vizserver server display. The OpenGL Vizserver client display must have at least as many pixels available for display as the OpenGL Vizserver server display. When the OpenGL Vizserver server runs on a system with a large display area, such as a machine from the SGI Onyx family, with multiple display channels, it is difficult to ensure that the client display matches the server display. In these cases the results will be correct as long as the application running on the OpenGL Vizserver server is restricted to the smaller of the client and server display areas. Movement of the application window outside of the proper area is not an issue for an OpenGL Vizserver client on the VAN, but a user sitting at the OpenGL Vizserver server host machine may move the application window without knowledge of the OpenGL Vizserver client configuration, causing it to disappear from the remote display.

5.2 Multipipe Servers and Multipipe Clients

OpenGL Vizserver can serve from a multipipe server (e.g., an SGI Reality Center facility) to a multipipe pipe client. The session must be initiated with the appropriate number of pipes requested.

5.3 Multipipe Servers and Single-Pipe Clients

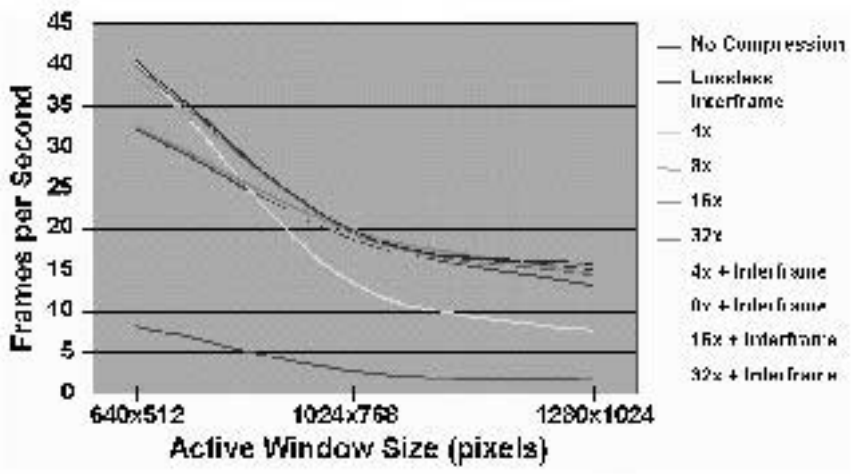
Although OpenGL Vizserver can serve from a multipipe server (e.g., an SGI Reality Center facility) to a single-pipe OpenGL Vizserver client, care must be taken in placing the windows within the Reality Center environment. Each pipe will map to the same screen space on the OpenGL Vizserver client, so proper server-side window location is critical to ensure a proper, nonoverlapped display on the client; window location on each server pipe must be unique. This ensures that the resulting client images do not overlap when the server pipes are mapped to the same screen space on the OpenGL Vizserver client.

In addition, pipes that render user interface elements in small popup windows will generally render correctly without occluding important information rendered on the main pipe. Generally, multiple pipes rendering full-screen or near-full-screen images will occlude one another when rendered on the OpenGL Vizserver client and will look incorrect to the user.

6 Authentication and Firewalls

6.1 Firewalls

Firewall support is an integral part of OpenGL Vizserver. Vizserver*SessionBasePort (in /var/vizserver/config, also accessible through the configuration panel) specifies the base port that OpenGL Vizserver should use. Each pipe will consume three additional ports, so the firewall needs to open ports Vizserver*SessionBasePort + 3 * PipeCount. The default range for the client connections is 0x2000-0x2fff. The server is on port 7051.



on a private 100Base-TX full duplex switched network.

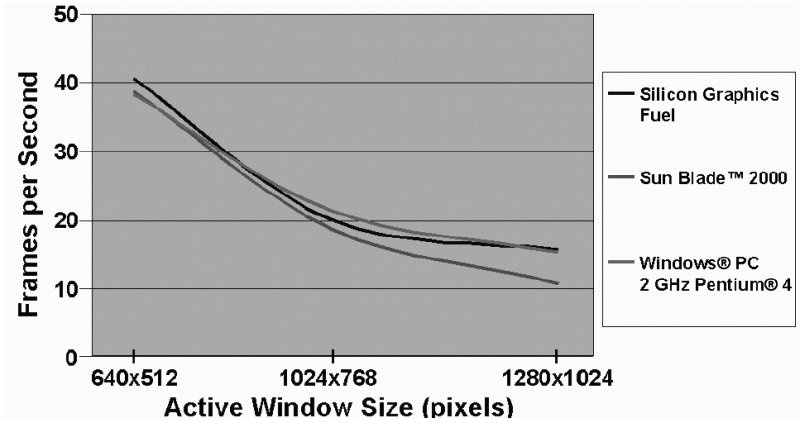


Figure 6: Frame rates for OpenGL Vizserver 3.1.
Top: Performance tested from an SGI Onyx 3000 series system to a Silicon Graphics Fuel on a private 100Base-TX full duplex switched network.
Bottom: Frame Rates as a function of window size and client type. Performance tested from an SGI Onyx 3000 series systems on a private 100Base-TX full duplex switched network.

6.2 Authentication

OpenGL Vizserver comes complete with an authentication module that allows system administrators to control which users can remotely access the Onyx family system and at what times they are allowed to access it.

7 Pipe Allocation/Management

When graphics pipes are allocated to the OpenGL Vizserver sessions by the OpenGL Vizserver server manager, there are two types of allocation methods used: static pipe allocation and dynamic pipe allocation. The terms static and dynamic refer to the mobility of graphics pipes between xdm for local, interactive users and OpenGL Vizserver.

7.1 Dynamic Pipe Allocation

In this mode OpenGL Vizserver can allocate the graphics pipes that it manages, as well as the graphics pipes managed by xdm. OpenGL Vizserver allocates xdm-managed pipes for a session's use only if the X server that currently uses the graphics pipes is not logged in. In other words, if the X server does not have an active user and is in the login stage with the login screen displayed, then OpenGL Vizserver can dynamically allocate that pipe to support a remote user. If, however, that pipe is being used and there are no unused pipes, then the remote user requesting access via OpenGL Vizserver will be told that no resources are available. In order for the server to know which X servers are logged in and which are not, three scripts used by xdm must be changed to record the X server's state in the system's utmpx database. This change is made automatically when installing OpenGL Vizserver. The changes include installing new scripts on the system (in /var/X11/xdm) and modifying xdm's configuration file (/var/X11/xdm/xdm-config). These scripts are the following:

- Xlogin: This script starts the login process of the X server. On installation of OpenGL Vizserver, this script is replaced by Xlogin.vizserver.
- Xstartup: This script is run after a user has logged into the X server. On installation of OpenGL Vizserver, this script is replaced by Xstartup.vizserver.
- Xreset: This script is run after a user has logged out of the X server. On installation of OpenGL Vizserver, this script is replaced by Xreset.vizserver.

The OpenGL Vizserver server reads the xdm X server file (/var/X11/xdm/Xservers) to understand the current state of the system graphics pipes. It also changes the file every time xdm-managed graphics pipes are allocated dynamically or returned to xdm. In OpenGL Vizserver, reservation and dynamic pipe allocation are mutually exclusive.

7.2 Static Pipe Allocation

In this mode OpenGL Vizserver can allocate only graphics pipes that it manages. This is the default behavior of OpenGL Vizserver.

If the Vizserver*ReservationActive parameter's value is False, a user can open a session using any graphics pipes that are managed by OpenGL Vizserver (subject to availability). If the Vizserver*ReservationActive parameter's value is True, a user cannot have a session using more than the maximum number of graphics pipes reserved. If no reservation was made by a user, the user cannot open a session at all.

8 OpenGL Vizserver Performance

Figure 6 summarizes key performance metrics for OpenGL Vizserver 3.1. Most interactive visualization applications in the sciences and engineering run with interactive window sizes of between 640x512 pixels and 1024x768 pixels and with frame rates between 10 and 20 frames per second. These figures show that OpenGL Vizserver is able to exceed these rates and deliver high-quality interactive visual results to a wide variety of desktop devices at high rates of speed. More recent and more complete performance information can be found at [3].

References

[1] Two Bit/Pixel Full Color Encoding, SIGGRAPH 86 Conference Proceedings, 1986

[2] Hardware for Superior Texture Performance, EuroGraphics 95 Conference Proceedings, 1995

[3] Internet: www.sgi.com/software/vizserver/tech_info.html