White Paper

# SGI® OpenGL Vizserver™ 3.5 Visualization and Collaboration (Application-Transparent, Remote, Interactive)

## Table of Contents

## 1.0 Introduction

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, thin clients. VAN 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.

OpenGL Vizserver enables users of remote workstations, laptops, and even wireless tablet computers to use existing unmodified applications to access and control the power of Silicon Graphics Prism™ and SGI® Onyx® families of visualization systems and collaborate with one another by selecting from existing visualization applications based on X11 and the Open GL® API or by sharing the entire desktop from the server. The Silicon Graphics Prism and SGI Onyx families of visualization systems can be deployed as shared group, departmental, or enterprise visual servers in the manner that best matches the needs of an organization.

### 1.1 Collaboration

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, arbitrary fashion. This control mechanism provides the flexibility necessary to conduct formal design reviews or unstructured brainstorming sessions.

### 1.2 Application and Desktop Sharing

With OpenGL Vizserver 3.5 and higher, users have two options for the scope of graphics being served: application-based or desktop sharing mode. In application-based mode, users select specific application windows to be served. With desktop sharing mode, users are able to access and share the entire desktop from the server, enabling greater flexibility for a wider range of applications.

### 1.3 Resource Centralization

Centralizing the visualization resources allows for the consolidation of data, additional security of that data, and alleviation of the need for data replication to local desktops. Also, since only the visual servers need to be upgraded to meet increased visualization demands, desktop upgrades are simplified and copy-data demands for network upgrades are eliminated. An added benefit of the OpenGL Vizserver approach is that network bandwidth to clients remains constant while the data set sizes and computational complexity of the visualizations can increase without bound.

### 1.4 Scalability

OpenGL Vizserver leverages the scalability provided by Silicon Graphics® visualization systems. Scalable graphics hardware includes support for multiple graphics pipes, SGI Scalable Graphics Compositors, and SGI Scalable Graphics Capture (SGC) cards. The compositors provide a hardware solution to joining and cascading the video output of two or more graphics pipes and outputting them in a single video output. The SGC cards provide enhanced hardware readback including the ability to ingest external DVI signals from hardware compositors or other computers.

On the software side, SGI provides a full range of toolkits that enable scalable applications. The toolkits include products like OpenGL Performer™ and OpenGL Multipipe™ SDK.

### 1.5 Server-Client Independence

OpenGL Vizserver supports clients that range from a Silicon Graphics Prism 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.

## 2.0 OpenGL Vizserver Architecture

OpenGL Vizserver software has two primary components: a server and a client. The server must be one of the Silicon Graphics visualization systems—for instance, a Silicon Graphics Prism or SGI Onyx system. The OpenGL Vizserver client runs on a variety of operating systems including IRIX®, Linux®, Microsoft® Windows®, Mac OS X®, or Solaris™ operating systems, and the client system need not have extensive graphics or computational power.

The server manages graphics resources (for example, graphics pipelines) and monitors 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 the SGI 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 to create a seamless visualization environment on the client as if the user were interacting locally with the SGI graphics supercomputer.

### 2.1 Implementation Details

Figure 1 illustrates the architecture of OpenGL Vizserver. 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 by 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 transparently 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 (Application Binary Interface) on IRIX and the 64-bit Itanium™ ABI on Linux. Each of these libraries will work with a single-threaded 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.

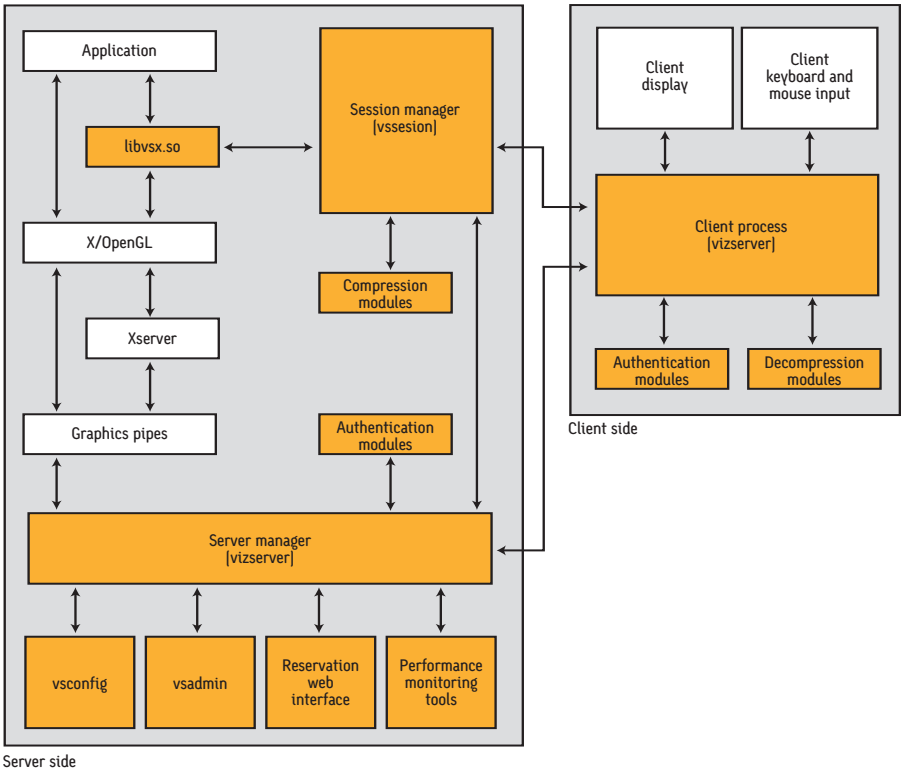


Fig. 1. OpenGL Vizserver Architecture

Recording data from the monitored `libGL.so` and `libX11.so` API entry points is handled by 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 framebuffer, 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 framebuffer, compressed, and shipped to the client.

#### 2.1.1 Hardware Readback

When SGC cards are installed in the system and are connected to the outputs of the graphics pipes being used by the OpenGL Vizserver session, the framebuffer readback will be done using these cards. Instead of performing software readback inside `libvsx.so` (and, thus, within the application process), the framebuffer is read from the SGC cards that are connected to the graphics pipes output. The communication between `libvsx.so` and `vssession` is reduced in this case to the events required for OpenGL Vizserver functionality without the actual pixel data. The readback performance is increased when using this method, and the influence of OpenGL Vizserver on the application's process is reduced to the bare minimum.

#### 2.1.2 Desktop Sharing

Upon startup of an OpenGL Vizserver session, the user can choose to make it a desktop sharing session. In desktop sharing mode, instead of serving specific application windows, OpenGL Vizserver serves the whole server-side desktop to the client. The library `libvsx.so` is not used in this case, and `vssession` reads the entire desktop from the framebuffer independently from the behavior of applications running on the desktop. On the client side, one window per server screen will be displayed.

Desktop sharing mode can be used with both software readback (in which case, `vssession` reads the framebuffer using the READDISPLAY X11 extension or the **glReadPixels()** OpenGL call) and with hardware readback using an SGC card. The difference in readback performance between these two methods is very noticeable in desktop sharing mode, and it is highly recommended to use the SGC card in this case.

### 2.2 OpenGL Vizserver Pipeline

The OpenGL Vizserver pipeline, briefly described in section 2.0, is implemented with an FIFO queue at each stage of the pipeline; each stage runs independently of the others. The frame rate is the number of frames per second processed by the longest stage, and the latency is roughly the number of stages along the pipeline times 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 the individual stages of the pipeline; each section describes both the client and server operation. Figure 2 illustrates the pipeline.

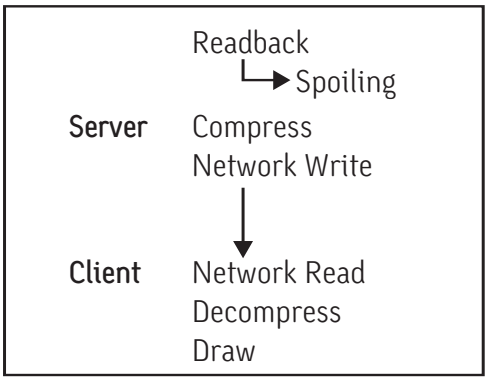


Fig. 2. OpenGL Vizserver Pipeline

### 2.2.1 Readback (Server) and Draw (Client)

The application populates the graphics framebuffer and OpenGL Vizserver reads the completed frame into memory for readback to the client. Only the portion of the framebuffer 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 3. When an SGC card is installed and hardware readback is used, the readback time is completely independent from the application and depends only on the refresh rate of the graphics pipe's output.

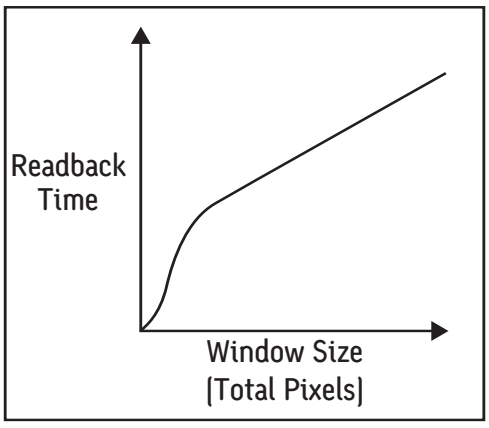


Fig. 3. Readback Performance as a Function of Window Size

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's 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 run-ning 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

OpenGL Vizserver determines that a frame has been completely drawn when an application issues either a **glxSwapBuffers()**, **glFlush()**, or **glFinish()** call. These OpenGL API calls (readback trigger functions) are monitored and, when issued, trigger OpenGL Vizserver to read the completed frame from the graphics framebuffer into main memory. 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 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. Information on the number of frames spoiled can be obtained through the performance characterization tools described in section 3.4.

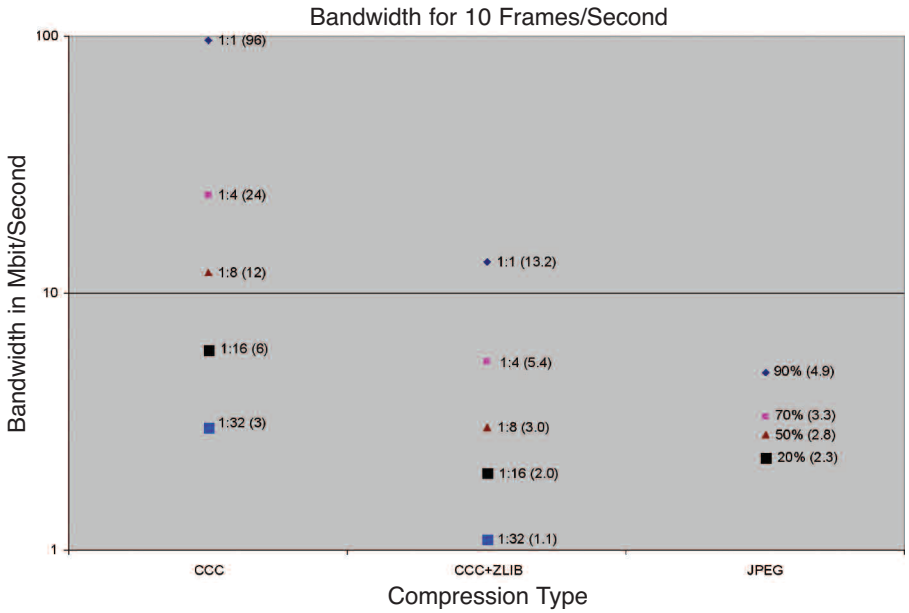


Fig. 4. Bandwidth Requirements for OpenGL Vizserver Compressors

When spoiling is off, 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.

A third option is time-based spoiling, where the server sends a new frame every time the client is ready to handle it, regardless of the application's behavior. This option is useful in special cases where the application being served is incompatible with the OpenGL Vizserver paradigm, which uses readback trigger functions. Typically, such applications do not call a readback trigger function at the end of each frame or draw to the same window with both OpenGL and X drawing functions. Time-based spoiling must be used when the session is in desktop sharing mode.

### 2.2.3 Compress (Server) and Decompress (Client)

OpenGL Vizserver provides a variety of compression schemes to compress/decompress frames of the rendered scene. The OpenGL Vizserver compression model is a tiered approach: a native interframe compression framework and two levels of user-selectable compression schemes. At the first level, users can select one of the following schemes:

- No compression
- Delta Color Cell (CCC, ICC, SCCC, or SICC)
- JPEG™
- Lossless (LLC)

At the second level of the tiered approach, users can select the ZLIB compression scheme. The following subsections describe these schemes along with the creation of custom compressors.

Figure 4 shows the amount of network bandwidth required to drive 10 frames per second of performance over a network with a typical model-viewing application and using compressors supplied with OpenGL Vizserver. The figure shows how network bandwidth can be reduced by increasing the level of image compression used.

#### 2.2.3.1 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 are 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 a 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.

#### 2.2.3.2 Delta Color Cell Compression (CCC, ICC, SCCC, SICC)

The CCC, ICC, SCCC, 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. SCCC 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 "Two Bit/Pixel Full Color Encoding," SIGGRAPH 86 Conference Proceedings, 1986, and "Hardware for Superior Texture Performance," EuroGraphics 95 Conference Proceedings, 1995.

#### 2.2.3.3 JPEG Compression

The other lossy compression scheme is Joint Photographic Experts Group (JPEG) compression, also known as the ISO/IEC IS 10918-1 standard. The implementation is based in part on the work of the Independent JPEG Group, as well as on other implementations. OpenGL Vizserver provides JPEG compression ratios of 20%, 50%, 70%, and 90%. The highest compression level, 90%, provides the best quality but requires the most network bandwidth.

#### 2.2.3.4 Lossless Compression

In addition to lossy compressors, there is also a lossless compression module called LLC. 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.

#### 2.2.3.5 ZLIB Compression

At the second level of the tiered approach, users can select ZLIB compression to be used on top of the level 1 specification, except in the case of JPEG compression. When ZLIB compression is enabled, the OpenGL regions of the image are compressed using the ZLIB algorithm. Regions other than the OpenGL regions (that is, pure X regions) are compressed using the ZLIB algorithm rather than the default RLE algorithm. Enabling ZLIB has the advantage of reducing the network bandwidth required by the OpenGL Vizserver session at no cost to the image quality. The disadvantage is that the ZLIB compression/decompression introduces higher computational loads on the server and client processors and, thus, might result in an overall reduced performance.

#### 2.2.3.6 Custom Compressors

OpenGL Vizserver also includes an API that provides the capability to develop new modules with user-defined functionality. The API allows the creation of compression modules that take into account domain-specific data formats and data structures that cannot be optimally handled by 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.

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 OpenGL Vizserver module development 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 that the slowest client can accept.

## 3.0 Optimizing OpenGL Vizserver Performance

To optimize OpenGL Vizserver performance, you should consider the following items:

- Network bandwidth
- Network latency
- Processor utilization
- Monitoring OpenGL Vizserver performance
- Expected performance

### 3.1 Network Bandwidth

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. In addition to image size, the physical network condition, routing topologies, switch bandwidths, and network congestion leading to packet retransmits must all be considered. Users should keep these overheads in mind when calculating the expected bandwidth or frame rate.

### 3.2 Network Latency

By default, OpenGL Vizserver is generally well-suited for low-latency networks, such as Ethernet-based local area networks (LANs). However, in high-latency environments, such as wide area networks (WANs), the default configuration does not utilize the network's bandwidth well and achieves sub-optimal frame rates. For such case, OpenGL Vizserver allows users on the client side to configure the number of frames that are *in transit* at any given time from the server to the client.

### 3.3 Processor Utilization

The OpenGL Vizserver server is a pipeline wherein the framebuffer readback, image compression, network writes, various pixel conversions, and handling of non-GL regions each run independently. Some of the built-in compressors (namely, the lossless and CCC compressors) are multi-threaded. In most cases, three processors per OpenGL Vizserver session will fully utilize the parallel nature of this pipeline—two processors for image compression and pixel conversion and another one for the rest of the server-side work. This is in addition to the number of processors needed for the application.

### 3.4 Monitoring OpenGL Vizserver Performance

The main purpose of monitoring OpenGL Vizserver performance is to find performance bottlenecks and ensure that an application running under OpenGL Vizserver gives virtually the same performance as the application running locally without using OpenGL Vizserver. To monitor OpenGL Vizserver, a Performance Co-Pilot™ (PCP) OpenGL Vizserver Performance Metric Domain Agent (PMDA) module and a text-based tool, `vsmonitor`, are available. Easy to use, they are also useful in monitoring the performance of each stage in the OpenGL Vizserver pipeline.

The PCP OpenGL Vizserver PMDA collects the performance data from the OpenGL Vizserver server and makes it available for the various performance metrics viewers available with PCP. Some of the OpenGL Vizserver performance metrics are the frame rate, the time consumed by each of the stages in the OpenGL Vizserver pipeline, the amount of pixel data that moves between the stages and is sent to the clients, and more. Figure 5 shows a snapshot of `pmchart`, a PCP graphical viewer, monitoring an active OpenGL Vizserver session.

In addition to the Performance Co-Pilot tools, `vsmonitor` is a text-based tool that is used to display various performance metrics in a text-only display. It reports the current values of all the metrics from the OpenGL Vizserver server every five seconds.

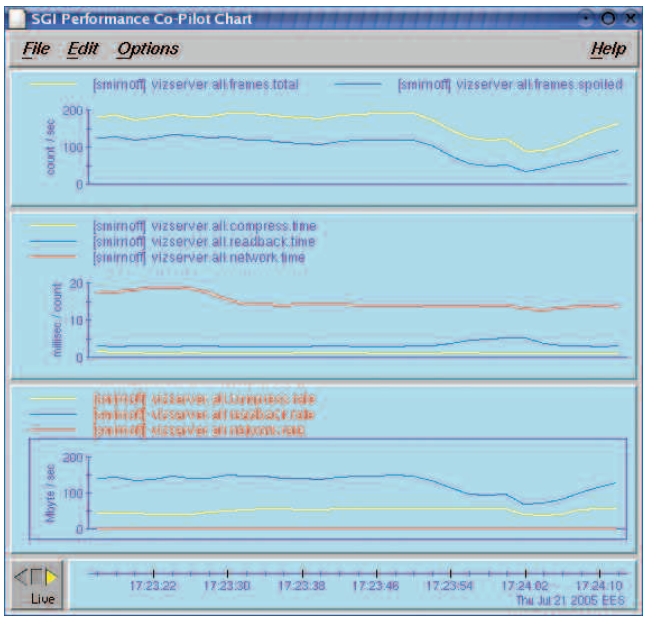


Fig. 5. The PCP Graphical Viewer `pmchart`

### 3.5 Expected Performance

Figures 6, 7, and 8 illustrate sample performance for OpenGL Vizserver 3.5 in various environments. 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 *www.sgi.com/software/vizserver/tech_info.html*.

## 4.0 Application Compatibility

OpenGL Vizserver can be used with today's applications in their current form. However, there are several considerations that are noteworthy regarding application compatibility:

- Ideal application design for OpenGL Vizserver
- Desktop sharing
- Hardware and software composition
- Cross-platform application execution

### 4.1 Ideal Application Design

Application-wise, 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()**

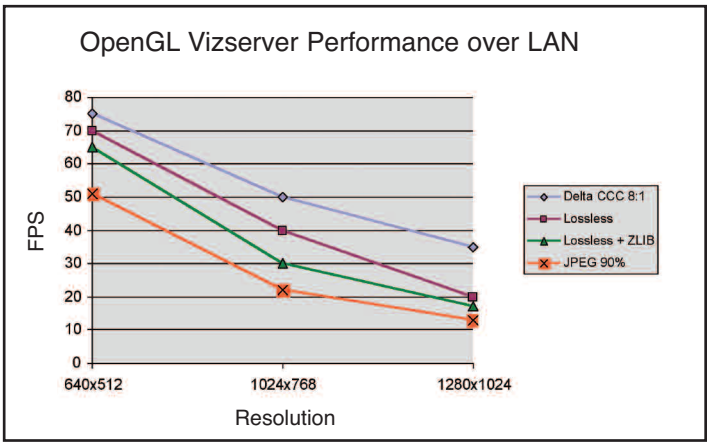


Fig. 6. Performance over LAN, Silicon Graphics Prism Server and Client, 100 Mbit Network, Various Compressors

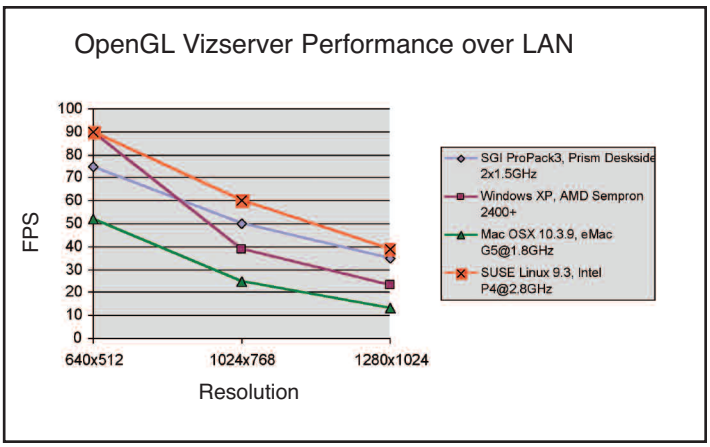


Fig. 7. Performance over LAN, Silicon Graphics Prism Server, Various Clients, 100 Mbit Network, Delta CCC 8:1 Compression

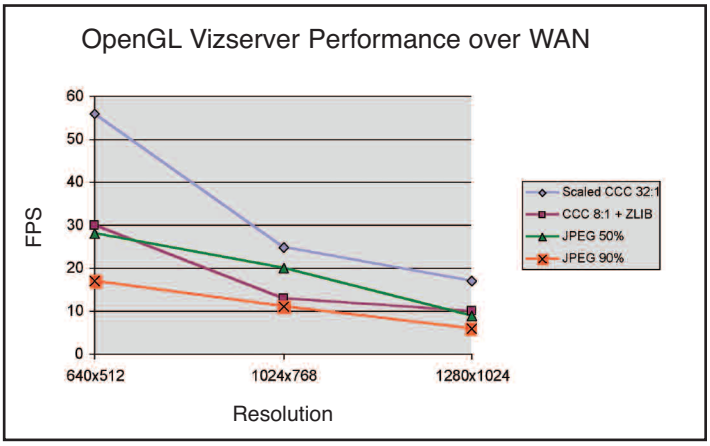


Fig. 8. Performance over WAN, Silicon Graphics Prism Server, SUSE LINUX 9.3 PC Client, 2 Mbit Network, Various Low-Bandwidth Compressors

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 to start 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 glXExtensionString 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.

### 4.2 Desktop Sharing

When starting an OpenGL Vizserver session, the user might choose to start it in desktop sharing mode. In this mode, the whole desktop on the server side is served to the clients rather than specific application windows. All windows, graphics, and other content that appear on the server's desktop will be viewable on the clients.

Starting a session in desktop sharing mode is useful when there is a need to serve applications which are highly incompatible with the OpenGL Vizserver paradigm of OpenGL readback trigger functions. Since OpenGL Vizserver reads the whole desktop from the framebuffer independently of the application's behavior, its functionality will not be harmed by any application-specific implementation.

Another reason for using desktop sharing mode is efficiency when there are many application windows on the server that need to be served to the client. In some cases, if the sum of graphics regions on the desktop is high enough, it might be more efficient for OpenGL Vizserver to serve the desktop as a whole rather than many smaller windows. Note that since the software readback rate in desktop sharing mode can be considerably lower than when serving specific applications, this efficiency improvement will probably exist only when an SGC card is available for hardware readback.

Desktop sharing is also the only way OpenGL Vizserver supports hardware composition applications, which are described in section 4.3.

### 4.3 Hardware and Software Composition

For improved performance, greater display resolution, and other reasons, some applications use graphics composition methods such as DB, 2D, or eye decomposition. The composition can be done using software only or a hardware solution. In the case of software composition, the application renders to several pipes and composes the target image in the CPU. For hardware composition, a hardware compositor can be connected to the output of the rendering graphics pipes to compose the final image.

To support software composition applications, the user can select upon OpenGL Vizserver session startup which of the screens belonging to the session's X server are active. Only application windows that are displayed on these screens will be served to the clients. For example, in a four-pipe OpenGL Vizserver session, a software composition application might use screens 1, 2, and 3 as pure rendering resources and compose the target image on screen 0. Selecting only screen 0 as active upon session startup will allow the clients to see only the final image of the application.

Hardware compositor support in OpenGL Vizserver is done using SGC cards, which are connected to the hardware compositors output. To support hardware composition applications, the user must start the session in desktop sharing mode and select one of the *video groups* available on the server instead of an arbitrary number of pipes. A video group is a set of graphics pipes, hardware compositors, and SGC cards that are connected to one another using external DVI cables. OpenGL Vizserver will start an X server on the video group's graphics pipes and serve to the clients the ingest of the video group's SGC cards (effectively, the hardware compositor's output). It is the job of the application (or the user) to configure the hardware compositors so that they meet the application's composition requirements.

### 4.4 Cross-Platform Application Execution

OpenGL Vizserver can work in conjunction with other products like Transitive® QuickTransit™ to allow cross-platform execution of applications. Many changes to this, and it should read: The QuickTransit software tool allows applications compiled for IRIX MIPS platforms to run transparently on SGI® Altix® and Silicon Graphics Prism systems, which are based on Intel® Itanium® 2 processors. Hence, QuickTransit software greatly augments the

number of graphics applications users can run on Silicon Graphics Prism systems.

### 5.0 Environments

This section describes the following server and client display environments:

- High-resolution servers and low-resolution clients
- Multipipe servers and multipipe clients
- Multipipe servers and single-pipe clients

#### 5.1 High-Resolution Servers and Low-Resolution Clients

OpenGL Vizserver does not resize the dimensions of the image stream when the size of the OpenGL Vizserver client display is different 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, 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, thereby causing the application window to disappear from the remote display.

#### 5.2 Multipipe Servers and Multipipe Clients

When starting a session, the user can select the number of graphics pipes on the server that will be part of the session. Assuming there are enough available pipes on the server, OpenGL Vizserver will start an X server containing the number of pipes requested by the user. If the client has at least as many pipes as requested for the sessions, an application window displayed on a specific screen on the server side will be displayed on the corresponding screen on the client side.

#### 5.3 Multipipe Servers and Single-Pipe Clients

Although OpenGL Vizserver can serve from a multipipe server (for example, an SGI® Reality Center® facility) to a single-pipe OpenGL Vizserver client, care must be taken in placing the windows within the SGI Reality Center environment. Each pipe will map to the same screen space on the OpenGL Vizserver client. Therefore, proper server-side window location is critical to ensure a proper, non-overlapped 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. 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.

### 6.0 Security

The use of OpenGL Vizserver does not require you to compromise system security in any manner. Firewall support and user authentication, both configurable, are integral parts of OpenGL Vizserver.

OpenGL Vizserver provides two authentication mechanisms that allow system administrators to control which users can remotely access the server system and at what times they are allowed to access it:

- `AUTH-PAM`—This scheme uses the Pluggable Authentication Modules (PAM) mechanism. When using the `AUTH-PAM` module on the server, no authentication module is needed on the client.
- `AUTH-PASSWORD`—This scheme is an unencrypted user/password mechanism based on the system's `passwd` database.

The OpenGL Vizserver module development package includes an API that allows customers to develop their own site-specific authentication modules. One can also use the `AUTH-PAM` authentication scheme and use it with custom PAM modules developed using the PAM modules API.

### 7.0 Pipe Allocation/Management

When graphics pipes are allocated to the OpenGL Vizserver sessions by the OpenGL Vizserver server manager, there are two types of pipe allocation methods used: static pipe allocation and dynamic pipe allocation. The terms static and dynamic refer to the mobility of graphics pipes between the X display manager (XDM) for local, interactive users and OpenGL Vizserver. A site administrator configures the following:

- Which graphics pipes are managed by OpenGL Vizserver
- Whether OpenGL Vizserver can use XDM-managed graphics pipes for its sessions (Default is `True`.)
- Whether a graphics pipe reservation by a user is required in order to use a pipe managed by OpenGL Vizserver (Default is `False`.)

#### 7.1 Dynamic Pipe Allocation

In dynamic pipe allocation 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 through OpenGL Vizserver will be told that no resources are available.

Pipe reservation by a user and dynamic pipe allocation are mutually exclusive.

### 7.2 Static Pipe Allocation

In static pipe allocation mode, OpenGL Vizserver can allocate only graphics pipes that it manages. If so configured, a user can open a session using any graphics pipes that are managed by OpenGL Vizserver (subject to availability). Otherwise, 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.



Corporate Office
1500 Crittenden Lane
Mountain View, CA 94043
(650) 960-1980
www.sgi.com

North America +1 800.800.7441
Latin America +55 11.5509.1455
Europe +44 118.912.7500
Japan +81 3.5488.1811
Asia Pacific +1 650.933.3000


3263 [10.10.2005]
J15002