

## **Scalable visual networking: using OpenGL overloads**

*By Davide Pasetto and John Hamill  
IBM Systems & Technology Group*

---

Contents

---

- 2 Introduction to immersive visualization technology**
- 2 Deep Computing Visualization Scalable Visual Networking**
- 3 SVN overload mechanisim**
- 4 Customer pain: stereo output from mono application**
- 5 Customer pain: points are too small**
- 6 Customer pain: Doom3 does not display correctly**
- 7 Other SVN overload examples**
- 7 Conclusion**

## **Introduction to immersive visualization technology**

Visualization is used to aid the understanding of complex data and to support users in their decision making processes. Many industrial and scientific sectors are seeing a constantly increasing demand for high-end visualization capabilities as they deal with ever larger datasets. As high performance computing systems become more widely used for running high fidelity simulations, such datasets are becoming so big that their understanding is extremely difficult without the aid of novel tools.

Visualization of extremely complex datasets requires ultra high resolution to be able to see the smallest details, and an immersive environment, which allow the scientist and engineer to interactively explore complex data by literally putting oneself inside the data. The actual experience allows data exploration in first person, in contrast with the typical third person experience of the standard desktop interface.

Current software tools, display and workstations hardware cannot cope with the increasing data size and higher resolution demands, so a scalable and economically feasible solution is needed.

## **Deep Computing Visualization scalable visual networking**

Deep Computing Visualization (DCV) is an IBM suite of visualization products designed to address current customer pain when they must deal with computer aided visualization. The DCV product suite approaches the scalable visualization and the remote visualization of unmodified 3D application in an innovative way. In this whitepaper we'll show how the scalable visual networking (SVN) product can be used to enable cost effective and high performance immersive visualization environments.

DCV SVN is a middleware product designed to work with any unmodified OpenGL application. The product replaces the system OpenGL library (supported library vendor is NVIDIA) and intercepts and modifies the OpenGL instruction stream to scale and distribute rendering on a cluster of workstations, each containing one or more graphics cards. The individual output of each workstation can then be arranged physically on a tiled display, either built by standard CRT or LCD monitors or by using projectors or an active (or passive) stereo projector setup.

---

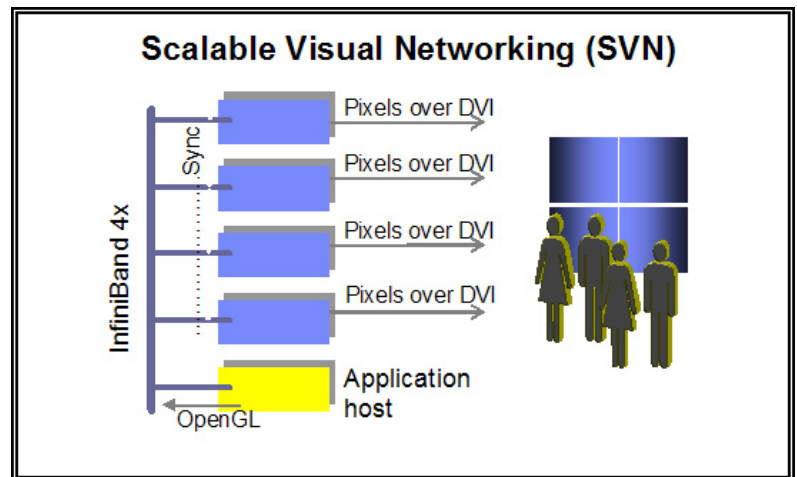
Highlights

---

*DCV SVN is a middleware product designed to work with any unmodified OpenGL application.*

*The effect of using SVN is to effectively turn a single rendering pipeline application into a multiple rendering pipeline application, by parallelizing rendering using a sort first and early clipping techniques in the OpenGL pipeline to speedup rendering.*

DCV SVN system architecture is shown in the following picture:



DCV SVN allows the user to configure one or more display walls showing the same application output. The ability to work with an unmodified application enables the user to continue working in a well known environment and, at the same time, enhances user experience by higher rendering performance and display pixel count. The effect of using SVN is to effectively turn a single rendering pipeline application into a multiple rendering pipeline application, by parallelizing rendering using a sort first and early clipping techniques in the OpenGL pipeline to speedup rendering. SVN relies on very high speed Infiniband network interconnection for distributing the OpenGL rendering instructions from the client application machine to the rendering servers.

**SVN overload mechanism**

The DCV SVN product contains an innovative user programming mechanism, which allows an experienced user to modify application rendering behaviour. DCV SVN intercepts every OpenGL function call and distributes these calls on each rendering server. The system provides a straight-forward mechanism which allows the user to replace the implementation of any OpenGL function and to modify its behaviour. This allows for “on the fly” changes to rendering parameters of the application, and these changes may be different for each rendering server output. This system enables SVN to solve a number of problems, related to showing existing application in a higher resolution display, in an elegant and efficient way.

***A passive stereo video wall with linear or circular polarization is a wonderful immersive environment which allows users to dive deep into the data and obtain a better insight and understanding.***

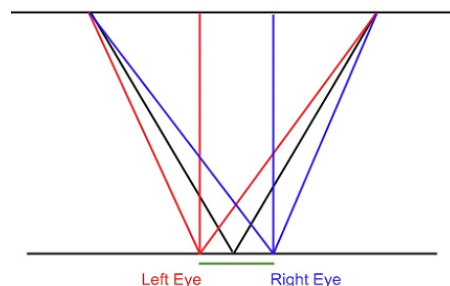
**Customer pain: stereo output from mono application**

A common customer problem is that the application they are using does not support passive stereo output. A passive stereo video wall with linear or circular polarization is a wonderful immersive environment which allows users to dive deep into the data and obtain a better insight and understanding. SVN allows the application to be displayed on a tiled video wall and, through the use of the overload mechanism, allows any application to be converted to passive stereo.

Passive stereo operates by superimposing two images (one for the left eye and one for the right) on a display. These images are normally superimposed through the use of polarizing filters and glasses to allow each eye to only see the appropriate image. Using a projected video wall and pairs of projectors for each region, a convincing full colour 3D display can be obtained. This kind of stereo output requires producing two slightly different images at one time.

The SVN passive stereo overload functions by dividing the set of SVN servers into two groups, the left group and the right group. Essentially two full video walls are used, with appropriate projector layout and filters to superimpose the final images. The overload needs to adjust the camera (or eye) position for each wall to the left or right, to simulate binocular eye separation, and then perform the rendering. When the images are combined viewed through polarized glasses, each eye views the 3D scene as it would in the real world.

To enable this camera positioning we must apply a 3D transformation to the scene (to the projection matrix in OpenGL terminology). This transformation must skew the camera's projection matrix so that the axes of projection align with the eyes, while also maintaining parallel projection planes to prevent visual distortion (see below).



Overloading those OpenGL functions which alter the projection matrix allows us to perform the stereo transformation. An additional matrix

stack entry, containing the stereo transform, is pushed and popped during any operation that alters the ‘real’ projection matrix. Thus the stereo transform can be applied transparently, without adverse effects on any existing rendering performed by the application.

All parameters used by the overload, including which render nodes reside in which grouping, and the level of the stereoscopic effect to be applied can be controlled through a simple text configuration file.

#### **Customer pain: points are too small**

Another common user problem is that some applications display their output without considering the pixel size of a large display wall. For example, take an application which displays its dataset as a set of coloured points. On a regular desktop display these points will be small, but visible. However, such single pixel points will result in an image which would barely be seen on a 3x4 metre video wall. The pixels remain singleton, but the area they are spread over becomes vast as each portion of the display wall retains full native resolution.

Furthermore, when this problem is combined with a passive or active stereo technique, the stereographic effect is nullified due to the inability for the eye to focus on the small points.

***A simple usage of the SVN overload mechanism allows the user to change how OpenGL points are displayed on the display wall, and tailor the rendering size to their needs.***

A simple usage of the SVN overload mechanism allows the user to change how OpenGL points are displayed on the display wall, and tailor the rendering size to their needs. Since overloads can be applied on a case by case basis the overload changes can be used as needed.

In order to change the size of rendered points in OpenGL the function `glPointSize()` is used. By default, SVN will scale values passed to this function by an appropriate amount appropriate to the display wall. However, should an application specify a size value much less than unity, the scaled size could still be very small resulting in tiny points on a large wall.

The SVN overload to compensate for this allows alternative fixed sizes to be specified in a simple text configuration file. These values can be tuned to override the default behaviour of the SVN system and render the points at an arbitrary size. With minimal tuning the passive stereo effect that is lost when the points are too small may be restored. A similar effect can be applied to other primitives (such as lines) should it be desired.

***Current computer graphics hardware is programmable, that is rendering techniques can be devised by the developer rather than being fixed by the hardware manufacturer.***

### **Customer pain: Doom3 does not display correctly**

Current computer graphics hardware is programmable, that is rendering techniques may be devised by the developer rather than being fixed by the hardware manufacturer. This programmability is needed for many of the visual effects employed in modern software, especially in the field of computer games.

An example of usage of a programmable pipeline is the game Doom3 by id Software Inc. This game makes use of hardware programmability to provide “heat-haze” ripples and “glass distortions” as the user moves through the game world. These graphics programs are known as shaders.

When this game application is rendered on a large tiled display massive visual distortions appear wherever the ripple effect is employed due to the mismatch between the resolution of the application host machine and the video wall.

The ripple effect uses the screen resolution as part of its rendering equations, and this resolution is captured from the client display. These values are not appropriate for the tiled video wall and the equations produce the visual distortions observed.

To compensate for this distortion problem there are two solutions. Either rewrite the programmable graphics hardware code or correct the values for the screen resolution to values appropriate to the render nodes. The second method requires less intervention and does not require access to application source code.

By tracking which shader (among the many used by the game) is responsible for the ripple effect we can also intercept and alter the parameters that the application passes to that programmable shader. This interception is implemented in an SVN overload that alters the values received from the client node when appropriate to ones better suited to the local display.

Through the use of the SVN overload system, the full impressive visuals of this game may be combined with the extreme resolution an SVN display wall can muster.

***Overloads can be used for implementing edge blending for projector walls in software: it is possible to configure the logical screen with overlapping borders and modify the implementation of SwapBuffers to apply a suitable post processing operations to buffer borders to smooth pixel colors towards the buffer edges.***

## **Other SVN overload examples**

Several other customer problems are suitable for being solved by developing a specific overload. Overloads can be used for implementing edge blending for projector walls in software: it is possible to configure the logical screen with overlapping borders and modify the implementation of SwapBuffers to apply a suitable post processing operations to buffer borders to smooth pixel colors towards the buffer edges.

Using overloads it is also possible to implement geometry correction overloads for curved projection walls or domes, turn the application output to a color coded stereo image or apply any picture post processing algorithm that may be needed for a better immersive display,

## **Conclusion**

The described overloads demonstrate the power of a customisable rendering pipeline when performing high resolution tiled rendering for unmodified applications. This powerful mechanism allows the user to change how the application displays without changing application code.

The customisable rendering pipeline can tune the application for the connected display hardware, such as enabling a stereo display, performing edge blending in multiple projector setups and handling dome distortion or curved displays.

The same mechanism can be used to enhance the quality of specific application display features (for example by enlarging the points drawn on the screen) or apply other image post-processing.

Finally the customisable rendering pipeline may be used to enhance specific application compatibility in a tiled display environment, by changing how the application uses OpenGL extensions and functionalities, such as shaders, vertex programs and others.

Deep Computing Visualization (DCV) scalable visual networking offers next-generation performance designed to handle increasing data size and higher resolution demands.



**IBM United Kingdom Limited**

emea marketing and publishing services (emaps)  
Normandy House  
PO Box 32  
Bunnian Place  
Basingstoke  
RG21 7EJ  
United Kingdom

The IBM home page can be found on the Internet at  
**ibm.com**

IBM is a registered trademark of International Business  
Machines Corporation.

\*\* OpenGL is a trademark of SGI.

\*\* Doom3 is a trademark of id Software Inc.

Other company, product and service names may be  
trademarks, or service marks of others.

References in this publication to IBM products, programs  
or services do not imply that IBM intends to make these  
available in all countries in which IBM operates. Any  
reference to an IBM product, program or service is not  
intended to imply that only IBM's product, program  
or service may be used. Any functionally equivalent  
product, program or service may be used instead.

IBM hardware products are manufactured from new  
parts, or new and used parts. In some cases, the  
hardware product may not be new and may have been  
previously installed. Regardless, IBM warranty terms  
apply.

This publication is for general guidance only.

Photographs may show design models.

© Copyright IBM Corporation 2008.

