Remote visualisation using open source software & commodity hardware

Dell/Cambridge HPC Solution Centre

Dr Stuart Rankin, Dr Paul Calleja, Dr James Coomer

Abstract

It is commonplace today that HPC users produce large scale multi-gigabyte data sets on a daily basis and that these data sets may require interactive post processing with some form of real time 3D or 2D visualisation in order to help gain insight from the data. The traditional HPC workflow process requires that these data sets be transferred back to the user's workstation, remote from the HPC data centre over the wide area network. This process has several disadvantages, firstly it requires large I/O transfers out of the HPC data centre which is time consuming, also it requires that the user has significant local disk storage and a workstation setup with the appropriate visualisation software and hardware.

The remote visualisation procedures described here removes the need to transfer data out of the HPC data centre. The procedure allows the user to logon interactively to the Dell | NVIDIA remote visualisation server within the HPC data centre and access their data sets directly from the HPC file system and then run the visualisation software on the remote visualisation server in the machine room, sending the visual output over the network to the users remote PC. The visualisation server consists of a T5500 Dell Precision Workstation equipped with a NVIDIA Quadro FX 5800 configured with an open source software stack facilitating sending of the visual output to the remote user.

The method described in this whitepaper is an OS-neutral extension of familiar remote desktop techniques using open-source software and it imposes only modest demands on the customer machine and network connection. In particular, no 3D graphics card is required, the client is a single application available for common Windows and Macintosh machines in addition to Linux, and interactive visualisation of complex and dynamic images can be achieved in a usable way through a home broadband connection.

Whitepaper Structure

1.0 Background

The computing power and storage capacity now available to high performance computing (HPC) facilities can easily produce terabytes of data daily, and individual data sets of tens of gigabytes are commonplace. These data sets are typically not the final product but require analysis. This post-processing may often involve an interactive stage in which the data set is "browsed", and interesting features, such as a particular region of fluid flow around an obstacle, or perhaps a volume of collapsing, galaxy-forming dust and gas, identified and examined in more detail. In such varied scientific areas as Computational Fluid Dynamics (CFD), Cosmology/Astrophysics, Molecular Modelling and Medical Imaging, this browsing of data is frequently performed by generating and manipulating three-dimensional images from the data set; the necessary interactivity, which requires complex graphics rendering in real time, is achieved through the hardware-accelerated 3D rendering capability of a 3D graphics card.

In practice, the owner of the data set is usually physically remote from the HPC facility and reliant on a network connection from a workstation (or PC/laptop) on their local area network (LAN) - indeed, even the HPC system administrators may be such remote users 95% of the time. This physical separation over the network presents the user with a problem when wanting interactive analysis of large scale data and the question arises: should the analysis application be run on the local machine, or on the remote HPC facility?

Run visualisation on local user machine

Running a 3D graphical application on the local machine requires a 3D graphics card capable of rendering the images in real time (in addition to adequate compute performance and main memory). Depending on the application, an ordinary "gaming" graphics card may be insufficient in terms of video memory or performance, making this a potentially costly item. Assuming for the moment that these requirements are satisfied, a second question arises – does the data set need to be copied to the local workstation?

The need to transfer large, multi-gigabyte files clearly introduces a delay, but one which may not be too onerous given sufficient local storage, network bandwidth and organisation. Locally attached disks with multi-terabyte capacities are now readily available (and may be less expensive by a factor of ten than the graphics card). An uncongested end-to-end 100Mbit/s network connection on a university network could be expected to transfer 35GB per hour. The level of inconvenience rapidly increases with the number of long transfers needed per day, and if it becomes necessary to send modified data sets back to function as initial data for new HPC jobs. Could one avoid moving the data, and instead analyse it in-situ on the HPC storage? One might consider mounting the remote storage on the local workstation as a network filesystem, using e.g. NFS. Alternatively, some applications (e.g. VisIt) can work as a (graphical) client front end running on the workstation, receiving data from a (non-graphical and parallel) back-end running on the HPC server. Both of these approaches introduce security concerns (e.g. firewall exceptions); this and the low network latency required for interactive performance confine such methods in practice to trusted front end nodes located on the HPC LAN. Thus 3D post-processing on the user's local workstation would seem to require a sufficiently high specification in terms of graphics card, CPU, memory and storage, reasonable (and reliable) network bandwidth to the HPC facility for data transfers, and a well organised user. The user should also be prepared to install and configure the necessary analysis software themselves, as their system generally lies outside the domain of the HPC support personnel. Such a well provisioned workstation is probably not mobile - should the user go travelling, all the problems of remote working return.

Run visualisation within HPC machine room

Relocating the analysis application to the HPC server has a number of immediate advantages. Firstly, no data transfers are required - data can be viewed and modified immediately on the (fast) HPC local filesystem. Secondly, the pressure on CPU, memory and storage switch to the HPC machine, which is normally amply provisioned. Thirdly, the application can be installed and configured centrally at the HPC end, saving the user from this complication. From now on we will assume that the HPC machine uses Linux as its operating system, that graphical applications running on it use the X windows system (X11), and that the 3D code of 3D graphical applications is written using OpenGL (integrated with X windows through the GLX extension to the X11 protocol). The problem becomes one of running a 3D application which is built using X11 and GLX on the HPC server, remotely from the user workstation.

We shall see in the next section that the need for good graphics and network performance unfortunately still applies to the user workstation when using traditional (X11/GLX) approaches to running remote applications. These traditional methods rely on the user workstation to perform all graphical rendering, and may work slowly with 3D graphics, or not at all; they also require the installation of additional X11 software if the workstation operating system is other than Linux/Unix. The third section describes the well known remote desktop software VNC which replaces these traditional methods and amounts to a complete solution for two-dimensional applications (i.e. applications using X windows but not OpenGL). We then introduce the open source software VirtualGL which is the key to moving 3D rendering to the HPC side of the network connection, where it can be accelerated by investment in a suitable graphics card in the server room, thereby extending the advantages of the remote desktop from 2D to real-world 3D applications.

2.0 Traditional approaches to remote graphical usage

2.1 2D applications

Basic to the design of the X Windows System (X11) is the feature that the graphical user interface of an application running on one machine can be drawn on (and controlled from) the screen of another – i.e. it is easy in principle to run X11 applications on a remote machine if you have a network connection to it and a local X11 display for it to draw on. Indeed this used to be so easy to do that it created severe security issues and nowadays (up to date!) Linux/Unix machines should refuse to accept requests from remote applications to access their X11 displays “out of the box”. However a secure (provided the remote machine is trustworthy), firewall-friendly and easy way to still do it is to use the X11 forwarding feature of SSH which pulls remote application connections back through the encrypted channel of a SSH login to present them as local connections to the local X display. The upshot is that provided a user has an X windows display running on their local workstation, starting a remote graphical application (e.g. nedit) on a remote (Linux) HPC server should be as simple as the following:

```
workstation$ ssh –Y username@hpcserver.domain.name
Password:
hpcserver$ nedit&
```

where this assumes that the terminal containing the workstation$ prompt is running inside an X session, and that the workstation has the OpenSSH implementation of SSH installed (the -Y option demands trusted X11 forwarding which is only safe if hpcserver is trustworthy; the weaker -X option, which requests restricted X11 forwarding, is intended for situations where the server may be hostile, but with it some applications may fail). Here only the X11 protocol is important, the workstation and hpcserver need have neither the same architecture nor the same operating system – e.g. hpcserver could be a 64-bit Intel (x86_64), Scientific Linux 5 system, whereas the workstation could be a 32-bit Windows Vista PC using the Linux-like environment provided by Cygwin (which includes OpenSSH and X11), or alternatively a MacOSX machine with the optional Apple X11 package installed. (Note that these are not the only ways to add X11 and SSH functionality to Windows and MacOSX. Cygwin is a freely available option for Windows providing many other software packages familiar from the Linux/Unix world, and both it and Apple X11 provide a GLX capable X display, see the next subsection.)

Assuming that the remote application contains no OpenGL, this simple approach involves the transmission of 2D drawing commands only from the server to the workstation. The rate these are transmitted, and therefore the network quality required, is application dependent, thus a visually simple xterm window might tolerate a 56kbps modem connection, but for more decorative applications a 10Mbit/s broadband connection would be far preferable for usability. (Software based on NX improves this situation by using caching and differential X11 protocol encoding.) Different font availabilities on the local and remote sides can be a nuisance and lead to ugly font substitutions; also on occasion incompatibilities between the X11 software on either side can cause application failures; however for 2D applications, this method is a tried and tested means of working on a remote Linux/Unix system. For 3D applications, an additional feature is required to handle the 3D drawing commands, but again this basic method should work (with caveats, as described in the next section).

2.2 3D applications

2.2.1 Indirect rendering

By 3D application we mean an application containing OpenGL code used to draw a three-dimensional image. The capability for an application to include 3D (as well as 2D) drawing commands in its instructions to an X display is provided through the GLX extension to X11, which must be supported by the X11 software on the workstation running the display. Current native X11 on Linux/Unix/MacOSX machines supports GLX, as does the X11 component of Cygwin for Windows. It is not however universal, e.g. the VNC “fake” X display does not support GLX, although the NX “fake” display does.

A remote 3D application sends both 2D and 3D drawing commands (plus any necessary data) to the local display. Assuming the latter supports GLX, this requires no change to the procedure described in the previous section (i.e. simply ssh to the remote system with X11 forwarding enabled and run the application). The X display receives both types of drawing command and must render the complete image: in the context of 3D images, this is referred to as indirect rendering (because the rendering is performed by a program other than the original application), however as described above this is just how ordinary 2D X windows applications have always worked. It should be noted that some features of OpenGL require direct access from the application to the graphics hardware – applications using such features cannot be indirectly rendered.

The need for the client workstation to perform the rendering implies a similar local 3D graphics capability to that required to run the application locally. Assuming the X display can itself make use of the hardware acceleration offered by such a card (which is actually a non-trivial requirement, as indirect rendering in the past has frequently used only software rendering), then the ability to use the remote 3D application relies on the drawing commands and associated data reaching the workstation quickly enough to maintain a decent frame rate. Unfortunately, in realistic cases requiring the transmission of many vertex operations and large textures per frame, gigabit network communication end-to-end may be necessary in order to achieve this in practice. Thus the traditional method by itself is insufficient for users to remotely operate 3D analysis tools effectively over network connections of ordinary quality.

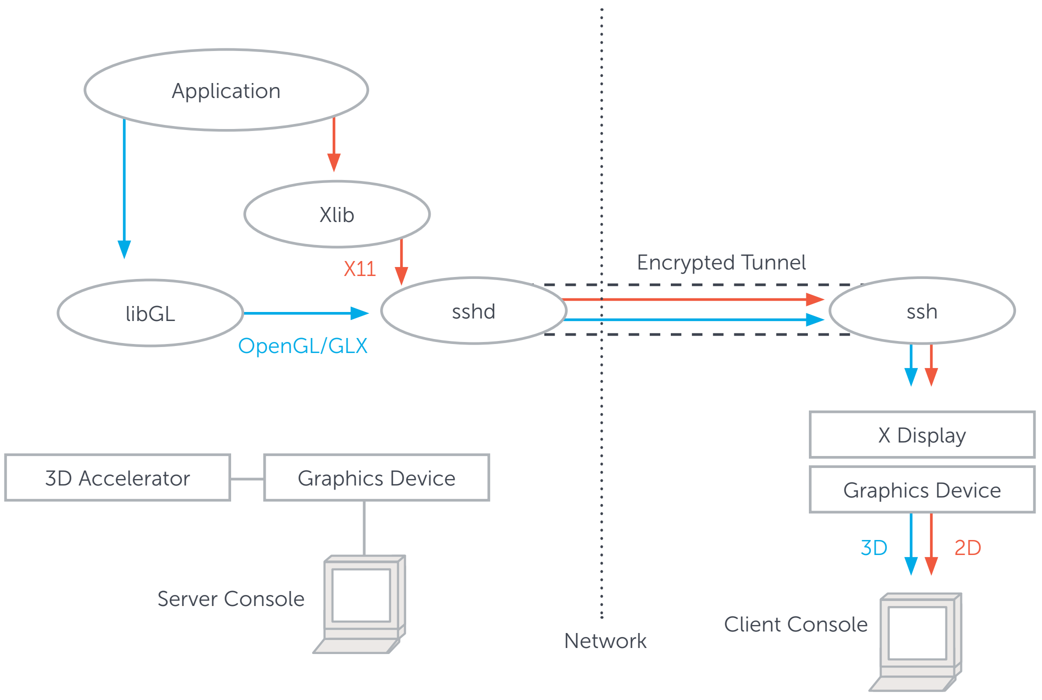

Figure 1. Indirect Rendering: a 3D application displaying on a remote workstation via GLX/X11 over an encrypted SSH tunnel. All the rendering is performed by the client X display.

2.2.2 Direct rendering

In contrast to indirect rendering, direct rendering involves the application issuing 3D commands directly to the 3D graphics hardware; the 2D commands are sent to the X display as usual. This mode of rendering is faster than indirect rendering (because the X11 software middle man is bypassed) but requires that the application and graphics hardware are both located on the same machine. In the usual case, this implies that the X display is also on this machine. Direct rendering is usually employed when the application and graphics display are local to each other.

This is not obviously relevant to “remote graphical usage” but we mention it here for completeness and because it will appear in a remote usage context later (when there will be a direct rendering display local to the 3D hardware at the HPC server, and a separate X display for the remote user’s desktop).

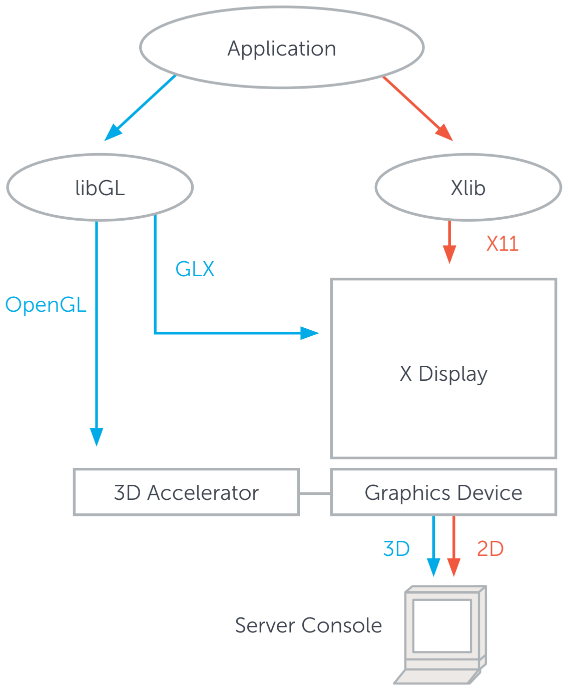

Figure 2. Direct Rendering: 3D drawing commands are sent directly to the graphics device where the rendering is performed with hardware acceleration and the results output to the local console.

In this section we have seen that although in principle using 3D applications which are running on a remote HPC server is straightforward using standard X windows methodology, in practice successful results place non-trivial requirements on network connectivity, graphics hardware and software at the local workstation, and on the application itself.

3.0 Remote desktops

3.1 Software for Linux/Unix

Remote desktop (RD) software is a familiar and widely used tool in both the Windows and the Linux worlds. Commonly used examples in the context of Linux/Unix servers are varieties of the open source VNC, and the proprietary NX software from NoMachine, both of which provide clients for a range of popular operating systems. Other proprietary solutions exist which are tied to particular operating systems.

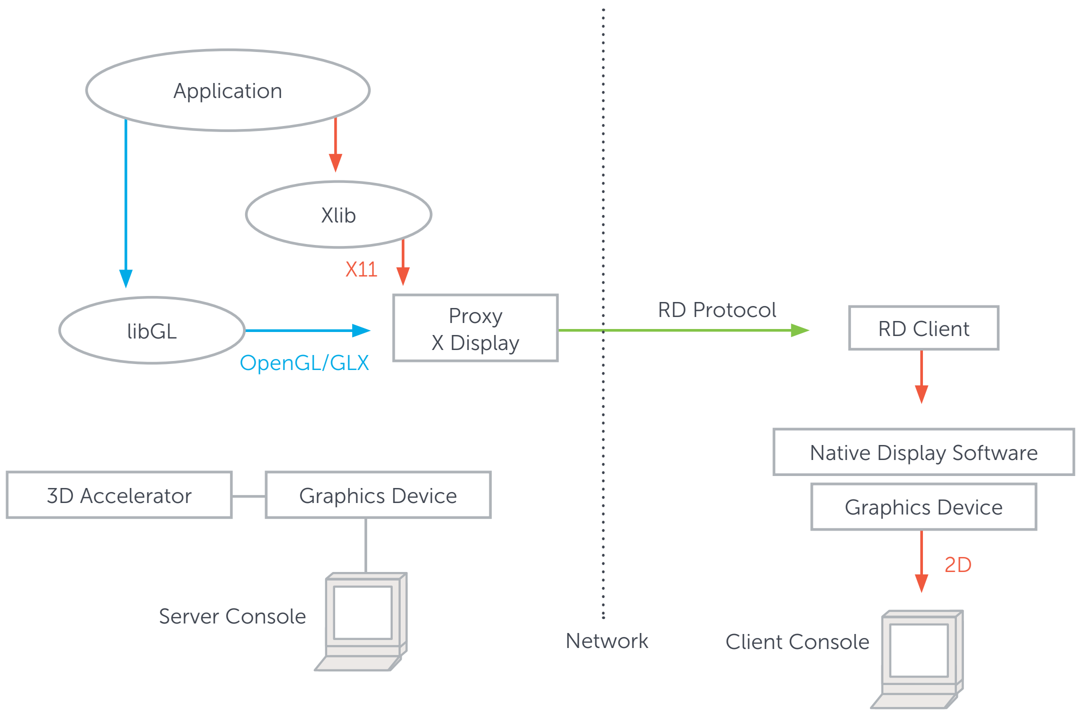

Figure 3. Remote Desktop: applications display to a proxy X server which may not be GLX-capable; the image of the virtual desktop is sent via a custom protocol to the client application on the remote machine.

A typical design for RD on a Linux/Unix server involves a daemon listening on a custom port, communicating with a remote client using an implementation-specific protocol (RFB in the case of VNC, and SSH in the case of NX). The daemon starts a "fake" (or proxy) X display on the server machine, onto which X11 applications running on the server can draw themselves. This display does not drive a physical graphics device, but instead the contents are encoded by the RD software and transmitted to the client. The client decodes the contents and draws the resulting image, which is of a virtual X desktop, onto the native display of the client machine. Keyboard and mouse events from the client are encoded and sent back to the proxy X server, where they are received in the proper way by the X11 applications connected to it. The RD network connection is usually tunnelled through SSH for reasons of security – with VNC variants this must usually be done explicitly, whereas NX handles this internally.

```
workstation$ ssh username@hpcserver.domain.name
Password:
hpcserver$ vncpasswd
Using password file /home/username/.vnc/passwd
Password:
```

Starting a remote VNC session on a Linux server is slightly more complicated than using remote X11/GLX because the VNC server process, which establishes a dummy X display and full desktop, must be started first:

This step is only needed the first time a VNC session is started – an encrypted password is placed in ~/.vnc/passwd. The next step starts a virtual X display with a desktop size of 1280x1024:

```
hpcserver$ vncserver  -geometry 1280x1024
New 'X' desktop is hpcserver:1
Starting applications specified in /home/username/.vnc/xstartup.turbovnc
Log file is /home/username/.vnc/hpcserver:1.log
hpcserver$
```

The :1 above indicates that the X display number allocated to the virtual display was 1 – if this number had already been taken, the first free number would have been used. Note that 0 is usually unavailable if there is a native X display on the server console. Vncserver is a script which launches the Xvnc binary; Xvnc provides the virtual X display and receives connections from VNC clients by default on port 5900+N, where N is the display number. The particular example here is of using TurboVNC, a variant of TightVNC with accelerated JPEG compression; the script ~/.vnc/xstartup.turbovnc is responsible for starting the desktop environment. We recommend modifying the top of this script to workaround a GNOME bug (if using that desktop):

```
#!/bin/sh

# Workaround for GNOME kbd settings bug:
export XKL_XMODMAP_DISABLE=1
...
```

TurboVNC also listens on port 5800+N for incoming HTTP, and serves a java-based client to web browsers visiting the address http://hpcserver:5801, in the above case. The server can be shut down with the command vncserver -kill :1 (the display number can be found with vncserver -list). Having started the session, one must connect to it from the remote workstation, however because the connection is not encrypted it will probably be desirable, and even necessary, to tunnel it over SSH. This can either be done manually (using the java client requires this) or in one step using the -via option to the vncviewer client:

```
workstation$ vncviewer -via username@hpcserver localhost:1
```

Authentication occurs in two stages – firstly, the user must authenticate to the SSH daemon which sets up the encrypted tunnel, and secondly the user is requested for the VNC password which was set via vncpasswd above (an entirely independent password from the user's account password on hpcserver). Equivalently, set up a tunnel explicitly with

```
workstation$ ssh -N -f -L 5901:localhost:5901 -L 5801:localhost:5801 username@hpcserver
```

and then connect to the session using either

```
workstation$ vncviewer localhost:1
```

for the native viewer, or

```
workstation$ firefox http://localhost:5801
```

for the java viewer.

The principal advantage of RD is that all resource-intensive operations are performed on the server itself (using the server's CPU, memory and storage) at "local" speed. For RD based on a pixel-based protocol like RFB (not to be confused with RDP for Windows servers) there is also the advantage that the demand on the network connection is determined by the size of the virtual desktop in pixels, the bits per pixel, and the efficiency with which the pixel information can be compressed, rather than by the implementation of the applications displayed on it. Thus in contrast to "traditional" methods of remote X11, RFB offers the hope of placing an upper limit on the network demand created by graphical applications, although it should be noted that transmitting entire desktops in pixel form imposes a greater network demand than remote X11 in 2D cases where only simple high level window drawing commands are involved; at least a 10Mbit/sec network connection between client and server is desirable.

Further advantages of RD are (i) the simplified software configuration required at the user workstation (installation of a RD client package plus possibly an SSH client); (ii) the persistence of the RD session through disconnections (the possibility of disconnecting and reconnecting from some other client or physical location while the applications on the virtual desktop continue uninterrupted); (iii) simple collaborative possibilities (multiple simultaneous connections from different users to the same virtual desktop); (iv) access to a complete Linux environment for users who don't wish to install this themselves. All of these are useful features in their own right, hence the popularity of RD solutions even where graphics intensive work is not envisaged.

Many examples of RD software are "screen scrapers", i.e. the virtual desktop is a faithful reproduction in real time of an actual desktop running on a physical console. Examples of this include WebEx, x11vnc, and VNC when running on Windows. This is not the normal mode of operation for NX or VNC when running on Linux, where the desktop is entirely virtual and independent of the physical console, which is more appropriate and useful in the context of a multi-user machine.

3.2 Limitations for 3D applications

The advantages offered by RD software invite the attempt to run 3D applications on the virtual desktop. Unfortunately, simply starting a 3D application within such a session is likely to fail immediately because the proxy X11 display created by the RD server does not possess the GLX extension required to support 3D drawing commands.

This is the case with the VNC proxy; the NX proxy on the other hand does support the GLX extension, but will attempt to indirectly render the image without hardware acceleration (software rendering). In general, 3D applications launched naively in RD are liable to instantly fail due to the absence of GLX, or show a completely black window because directly rendered images don't return from the graphics hardware, or work with extremely poor frame rate and interactivity as a result of software rendering by the proxy.

Assuming that a proxy server capable of hardware-accelerated 3D rendering were to become available, RD would then need only to encode and transmit the fixed number of rendered pixels making up the desktop per frame (however complex and three-dimensional the image rendered there, and however many vertices and large textures were required to perform the rendering). Frame rate and interactivity would then depend on the efficiency of this encoding and the available network bandwidth/latency. The compression/encoding methods used by RD usually need only to efficiently encode images with relatively few colours, low colour variation, and a lot of similarity between successive frames; 3D applications, on the other hand, can produce images involving many colours with rapid variation across and between frames, and a codec optimised for such images would be desirable.

4.0 VirtualGL

4.1 Overview

In the previous sections we examined various options for running graphical applications on a remote Linux/Unix server, with a view to identifying the optimal method for applications requiring hardware-accelerated 3D graphics. The conclusion was that traditional X11/GLX over SSH methods (when they work at all) may require a very high quality end-to-end network connection and additional hardware resources at the local server, whereas remote desktop software limits the required network bandwidth by offloading pixel rendering to the server but is not designed to support 3D applications with hardware acceleration.

VirtualGL (VGL) is an open source layer of software which addresses these problems by separating off the 3D rendering task and invoking the hardware-accelerated graphics system on the server to produce the required images. The resulting (2D) images are then fed to the target X display with the rest of the 2D drawing instructions. The key advantages of this are (i) any 3D image which can be rendered by the server's graphics hardware and software can in principle be rendered by VGL, and (ii) since only 2D commands are ultimately sent to the X display, the destination display need have no native 3D capability at all. The destination display here could be the X display on a remote user workstation, or a proxy display belonging to some RD software (in which case the end user receives the entire desktop, including both 2D and 3D elements, via the usual RD protocol). In either of these cases, if the X display is not local to the rendering hardware, a custom protocol ("VGL transport") is available with which to send the rendered 3D images to it in a compressed form, with or without additional encryption. With the expensive graphics hardware now firmly located with the rest of the expensive equipment at the server, it remains to re-examine the requirements on the network.

VirtualGL (VGL) is freely downloadable from www.virtualgl.org and is governed by the wxWindows Library License (see http://www.opensource.org/licenses/wxwindows.php).

4.2 Hardware and software setup

For a proper performance study of VirtualGL in various scenarios, please refer to http://www.virtualgl.org/pmwiki/uploads/About/vglperf21.pdf. Actual performance can vary for many reasons including server resource and network contention, and application idiosyncrasies, and there are a range of possibilities when employing VirtualGL in practice – e.g. whether VGL transport or X11 transport is used, whether the 2D X display is on the same machine as the 3D display, whether RGB or JPEG compression are used (and at what quality settings), whether VGL and/or X11 are encrypted, how the RD protocol is encrypted on its journey to and from the viewer. Here we limit ourselves to some illustrative but relevant cases in which the preferred RD technology (TurboVNC) is used with the RD server receiving the 3D images without any intermediate compression, direct from local memory via X11 transport (i.e. the 3D hardware, 3D display and RD proxy display are all on the same machine and VGL operates in "proxy" mode).

4.2.1 3D application server

The application server in the following practical examples is an internal node of the Darwin cluster at the University of Cambridge. Darwin at the time of writing consists of 576 dual-socket Woodcrest Dell PowerEdge 1950 nodes and 33 dual-socket Nehalem Dell Precision T5500 workstation nodes, connected via gigabit ethernet, and SDR and QDR Infiniband, with Lustre shared storage. The application server is a T5500 Dell Precision Workstation (2x quad core Intel Xeon 5560@2.80GHz, 24GB RAM) equipped with a nVidia Quadro FX 5800 (GT200GL) high-end graphics card and running 64-bit Scientific Linux 5.4. The native X display driven by the FX 5800 has full hardware-accelerated OpenGL/GLX 3D graphics through proprietary nVidia driver software version 256.35.

4.2.2 Client workstations

Three different types of client with different network connections to the server were chosen, the intention being to emulate three interesting categories of remote use:

A. Departmental. Intel Core2 6600 @ 2.40GHz, 2GB RAM, Fedora 12 64-bit Linux workstation , nVidia GeForce 9400 GT graphics card
- connected to the server via a 100Mbit/s departmental ethernet

B. Home. Intel Core2 E6850 @ 3.00GHz, 2GB RAM, Fedora 13 64-bit Linux workstation, nVidia GeForce 8600 GT graphics card
- connected to the server via a 10Mbit/s home broadband cable modem service

C. Laptop. Intel Core2 Duo T9300@2.50GHz, 2GB RAM, Windows Vista Ultimate SP2 + Cygwin 1.7.5-1 Dell XPS M1530 laptop
- connected to the server via local Eduroam 50 Mbps wireless.

4.2.3 Methodology

Two different ways of reaching the HPC server node's FX 5800 graphics card were tried in a simple practical test of the usability of a couple of example 3D applications:

1. GLX to a remote X11 display via SSH tunnelling (indirect rendering)

2. VirtualGL in X11 transport mode to a TurboVNC session running on the server and displaying to a workstation using a TurboVNC client via SSH tunnelling.

Each of the above client machines have a GLX-capable X11 display (through Cygwin, in the case of (C)). Although all of these client machines happen to have their own 3D-capable graphics cards, this aspect of them was not called upon as indirect rendering performed the OpenGL rendering in software, and all the other modes required only 2D rendering to occur on the client's native display.

4.2.4 Software

Although there are many (often proprietary) scientific graphical analysis packages on the market we chose to test VirtualGL with two readily available applications which would both produce complex and rapidly evolving images and at the same time require good interactivity to be usable without pain.

Celestia (http://www.shatters.net/celestia/) is a free, interactive 3D space simulator. Many Solar System and extra-solar bodies can be navigated to and flown around under the direction of the mouse and keyboard. Rotating and orbiting planets and moons are rendered with textures and special effects such as atmospheres, clouds, nightlights, sun reflection and lunar shadows. Version 1.5.1 with the default settings was used.

SmokeParticles is one of the demo programs included in the CUDA GPU Computing SDK code samples from nVidia (http://www.nvidia.com/object/cuda_home_new.html), which uses both the OpenGL and CUDA capabilities of the nVidia graphics card in the server. It simulates a cube of smoke sitting in a room which dissipates, and a moving smoke emitter which flies around. The viewpoint can be moved with the mouse while the simulation runs.

In addition to a 3D application and VirtualGL, suitable remote desktop software is required.

libjpeg-turbo is an improved (SIMD-accelerated) JPEG codec (see http://libjpeg-turbo.virtualgl.org/) for faster compression of the rapidly changing, complex images typically produced by 3D graphics applications. It is open source and freely downloadable.

TurboVNC is a TightVNC variant which is the current preferred accelerated VNC for use with VirtualGL. The improved codec used with TurboVNC was originally turbojpeg-ipp, which relies on some non-open source components from the Intel IPP library, but libjpeg-turbo can be used as a drop-in replacement, as we do in this whitepaper. Source and binary packages for common operating systems can be downloaded from http://sourceforge.net/projects/virtualgl/files/TurboVNC/. The version used was 0.6 with libjpeg-turbo 1.0.0, both of which were installed via rpm from binary packages. The native client was used in preference to the Java client.

4.2.5 Other remote desktop software known to work with VirtualGL

TigerVNC is another open source variant of TightVNC which makes use of libjpeg-turbo. It can be downloaded from http://sourceforge.net/projects/tigervnc/files/tigervnc/, but is already provided as the default VNC software in Fedora 12 and 13. TigerVNC is intended to supplant TurboVNC eventually, but at present using TurboVNC is still recommended.

NX is not completely open source but may be freely downloaded from www.nomachine.com. Server packages are available only for Linux and Solaris, however client packages can be downloaded for these plus Windows and MacOSX.

4.3 Installation of VirtualGL

The VirtualGL software (at the time of writing, the latest stable version is 2.1.4) can be downloaded from http://sourceforge.net/projects/virtualgl/files/. Binary versions are made available in various formats (e.g. .rpm and .deb linux package formats, .dmg MacOSX disk image, .exe Windows installation executable), in addition to the source (.tar.gz, .src.rpm). Installation of the binary packages is straightforward, here we will describe building the software from source (specifically from the .tar.gz compressed tar archive) on 64-bit, Intel64 Linux.

4.3.1 Prerequisites

Most prerequisites should already be satisfied (or can be easily satisfied through the usual distribution-dependent procedures) on a x86_64 linux system intended for software development: in particular, gcc and g++, GNU Make 3.7 or later, binutils 2.12.90 or later. Note that the X windows development group of packages should be installed, including both the native 64-bit and the corresponding 32-bit packages. In the case of Scientific Linux 5 (or by extension, RedHat Enterprise Linux 5 or CentOS 5), check that the following packages are installed:

glibc-devel.x86_64

glibc-devel.i386

make.x86_64

gcc.x86_64

gcc-c++.x86_64

binutils.x86_64

xorg-x11-proto-devel.x86_64

xorg-x11-proto-devel.i386

mesa-libGL-devel.x86_64

mesa-libGL-devel.i386

mesa-libGLU-devel.x86_64

mesa-libGLU-devel.i386

mesa-libGLw-devel.x86_64

mesa-libGLw-devel.i386

freeglut-devel.x86_64

freeglut-devel.i386

xorg-x11-tools.x86_64

In addition, the build system requires the openssl-static.x86_64 package (this requirement disappears in VirtualGL 2.1.90) and the libjpeg-turbo.x86_64 and libjpeg-turbo.i386 packages. The first is downloadable from the same site as VirtualGL, while libjpeg-turbo is a distinct open source project with files downloadable from http://sourceforge.net/projects/libjpeg-turbo/files/. libjpeg-turbo contains a SIMD-accelerated version of the JPEG codec which will be dynamically linked into VirtualGL and the VNC software in order to improve the efficiency of the compression applied to the 3D rendered images.

For this reason, libjpeg-turbo should also be installed on any server or client machine using either VirtualGL or an accelerated variant of VNC (TurboVNC or TigerVNC).

4.3.2 Building and installing

First, choose a suitable destination directory for the software – this is system dependent, e.g. on the HPCS cluster in Cambridge the preferred choice of target directory would be a location on the shared NFS storage such as /usr/local/Cluster-Apps/vgl/2.1.4. This directory should exist and be writable by the user (here, builduser) performing the build (for best practice, this should not be root).

```
root@hpcserver# mkdir /usr/local/Cluster-Apps/vgl/2.1.4
root@hpcserver# chown builduser /usr/local/Cluster-Apps/vgl/2.1.4
```

Next, as builduser, unpack the archive in a convenient build directory and compile and install the 32-bit components of VirtualGL:

```
builduser@hpcserver$ tar -zxvf VirtualGL-2.1.4.tar.gz
builduser@hpcserver$ cd vgl
builduser@hpcserver$ M32=yes make
builduser@hpcserver$ M32=yes make install prefix=/usr/local/Cluster-Apps/vgl/2.1.4
```

Now build and install the 64-bit components:

```
builduser@hpcserver$ make clean
builduser@hpcserver$ make
builduser@hpcserver$ make install prefix=/usr/local/Cluster-Apps/vgl/2.1.4
```

In order to use the software in this location, the end user will need to set the following environment variables (this can be done easily via environment modules):

```
export PATH=/usr/local/Cluster-Apps/vgl/2.1.4/bin:$PATH
export LD_LIBRARY_PATH=\
/usr/local/Cluster-Apps/vgl/2.1.4/lib:/usr/local/Cluster-Apps/vgl/2.1.4/lib64:\
$LD_LIBRARY_PATH
```

4.4 Configuring the 3D application server

Before VirtualGL can be used on the Linux application server, which we assume to possess a graphics card capable of hardware-accelerated 3D graphics, the X11 system on the server must be configured to receive the 3D rendering commands sent to it by VGL. This is necessary firstly, because access to the graphics hardware is mediated by the server's native X display, and secondly, because the 3D application typically starts out with authority only to contact the end-user's X display, which we are assuming is not the display with access to the 3D graphics hardware. Unfortunately, this presents a security concern, as currently the only way to allow access to the hardware is to allow the user full trusted access to the server's X display. Where multiple users must be granted this privilege, it opens the way for users to snoop and even change each other's 3D images. Furthermore, the activities of any user sat at the server's native 3D display in the normal way are then open to such monitoring and interference – note that whereas the 3D images generated by VGL are created in off-screen pixel buffers (so called pbuffers) which don't appear on the server's X console, the design of X11 nevertheless implies that VGL users have full permissions to interact with the on-screen and off-screen contents of the display. It is therefore strongly recommended that other users and root in particular should not login to the native X display if VGL is configured for use.

Thus bearing in mind that local X logins on the server should be discouraged, and that only trusted users should be allowed use of VGL, the recommended configuration steps to be performed on the server are as follows (we assume that the native X display is already set up to provide direct rendering of 3D graphics via its 3D device).

As root, move the server to run level 3 (no X desktop):

```
# init 3
```

This will normally shut down any active X display on the console.

On systems using a Linux distribution based on RedHat Enterprise 5 (e.g. Scientific Linux 5), check that the file /etc/gdm/custom.conf exists and contains lines beginning

```
command=/usr/bin/Xorg ...
```

In the absence of any such file, create one by referring to (or just copying) /usr/share/gdm/defaults.conf.

Next run the included configuration script:

```
# /prefix/bin/vglserver_config
```

where /prefix indicates the full path to the installation directory specified during the build process (for binary packages, this would be /opt/VirtualGL). This produces:

```
1) Configure server for use with VirtualGL in GLX mode
2) Unconfigure server for use with VirtualGL in GLX mode
X) Exit
```

Choose 1.

```
Restrict 3D X server access to vglusers group (recommended)?
[Y/n]
```

Because users with access to the 3D X display (the program is conventionally referred to as the X "server" in X11 parlance, and the applications drawing windows the "clients") will be able to monitor and even modify each other's commands, it is recommended to restrict these users to members of a particular unix group called vglusers. Choose Y – this will lead to the local creation of a vglusers group by the modification of /etc/group.

```
Restrict framebuffer device access to vglusers group (recommended)?
[Y/n]
```

This further limits direct access to the graphics hardware (required for 3D direct rendering) to members of the vglusers group. Choose Y – for Nvidia graphics cards this will change the permissions on the /dev/nvidia* devices so that only the vglusers group can read or write to them.

```
Disable XTEST extension (recommended)?
[Y/n]
```

This disables the XTEST extension within the X server program. The effect of this is to prevent users with access to the native display from inserting their own commands into a normal local login session; however it does not prevent such users from monitoring the session. Choosing Y is recommended to reduce the potential impact of malicious VGL users on console login users, however doing so will prevent the operation of "screen-scraping" remote desktop software attached to the native display such as x11vnc.

```
... Creating vglusers group ...
... Creating /etc/opt/VirtualGL/ ...
... Granting read permission to /etc/opt/VirtualGL/ for vglusers group ...
... Modifying /etc/security/console.perms to disable automatic permissions for
DRI devices
... Modifying /etc/security/console.perms.d/50-default.perms to disable automatic
permissions for DRI devices ...
... Creating /etc/modprobe.d/virtualgl to set requested permissions for /dev/nvidia* ...
... Attempting to remove nvidia module from memory so device permissions will be
reloaded ...
```

```
... Granting write permission to /dev/nvidia0 /dev/nvidia1 /dev/nvidia2 /dev/nvidiactl
for vglusers group ...
... Modifying /etc/X11/xorg.conf to enable DRI permissions for vglusers group ...
... /etc/X11/xdm/Xsetup_0 has been saved as /etc/X11/xdm/Xsetup_0.orig.vgl ...
... Adding vglgenkey to /etc/X11/xdm/Xsetup_0 script ...
... /etc/gdm/Init/Default has been saved as /etc/gdm/Init/Default.orig.vgl ...
... Adding vglgenkey to /etc/gdm/Init/Default script ...
... /etc/X11/xdm/Xservers has been saved as /etc/X11/xdm/Xservers.orig.vgl ...
... Disabling XTEST extension in /etc/X11/xdm/Xservers ...
... Disabling XTEST extension in /etc/gdm/custom.conf ...
... Setting default run level to 5 (enabling graphical login prompt) ...
... Commenting out DisallowTCP line (if it exists) in /etc/gdm/custom.conf ...
Done. You must restart the display manager for the changes to take effect.
```

Next edit /etc/gdm/Init/Default so that the vglgenkey that has been added to the top is replaced by

```
/prefix/bin/vglgenkey
```

Edit /etc/group so that the correct local users, including root and any potential login users of the native X display, are members of vglusers.

Finally, move the server to run level 5 (X desktop):

```
# init 5
```

Note that the configuration script has edited /etc/inittab so that the initdefault statement reads:

```
id:5:initdefault:
```

i.e. the server will start the native X display automatically on boot.

4.5 Running 3D applications

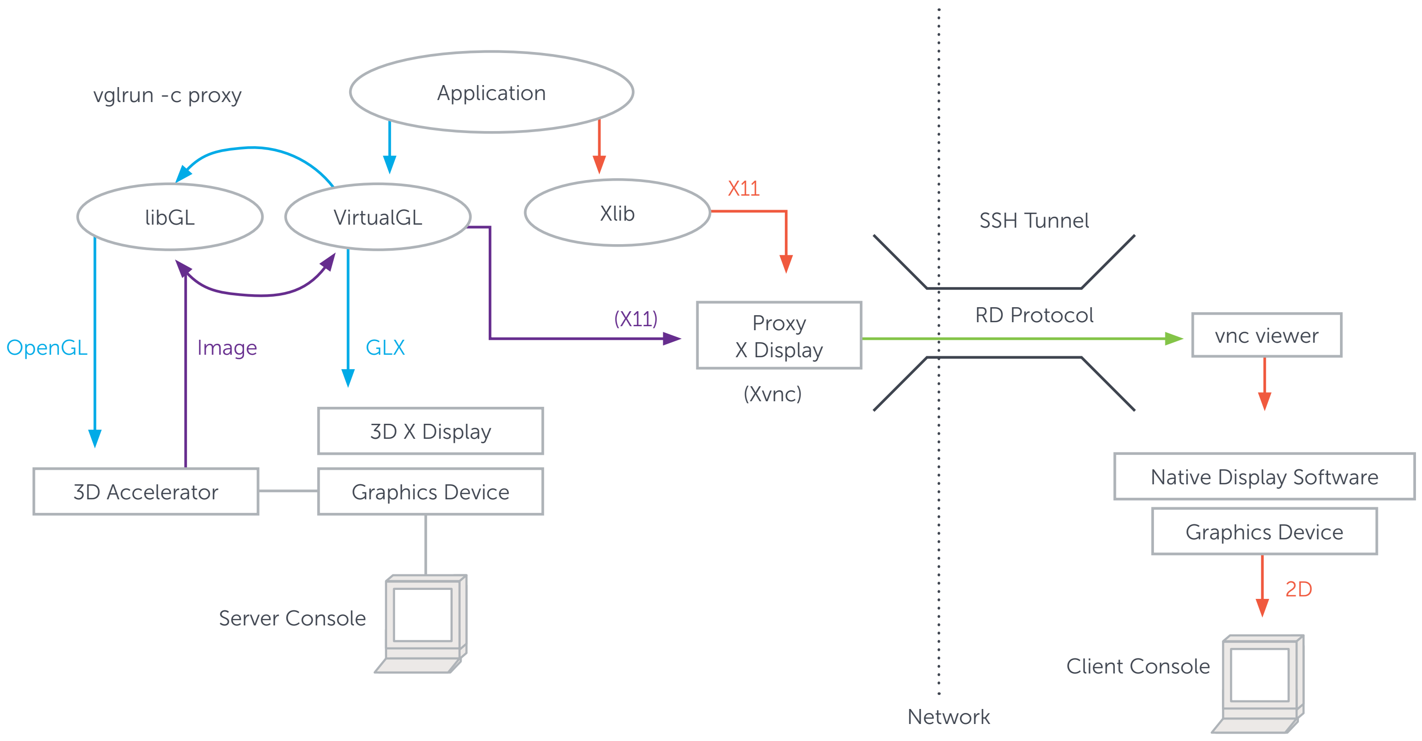

Figure 4. 3D application used remotely via VirtualGL and TurboVNC. The 3D drawing commands are forked off by interposing replacement shared objects in front of libGL.so, and rendered with hardware acceleration by the server graphics card. The rendered images are read back and pushed using X11 commands to the proxy X display. The entire desktop is compressed and sent using a combination of TightVNC and accelerated JPEG to the remote viewer, using an SSH tunnel for security.

4.5.1 Overview

In the following exercise we start and access a remote TurboVNC session using the native viewer on each of the three workstation types given above (Departmental, Home and Laptop).

Application frames per second (fps) reported values can be extremely misleading – these report the rate at which frames are delivered by the application to the proxy X display, rather than the rate at which frames representing the entire desktop are delivered to the workstation client over the RD protocol. One way around this is to use tcbench which is included in the VirtualGL distribution, and which should be run on the client machine and told to sample a block of the VNC window (see tcbench -? for options):

```
workstation $ tcbench –s100
Thin Client Benchmark
Click the mouse in the window that you wish to monitor ...
```

In addition to framerates (averages of three tcbench runs), we also give brief qualitative summaries of the usability for the various JPEG compression settings that can be adjusted on the fly within the TurboVNC viewer (press F8 in the viewer to access the settings menu on the Linux viewer; the Windows viewer has an icon to press top left of the viewer window). Unfortunately the tcbench utility cannot be used to measure the performance of the Windows VNC client so for the Windows laptop we confine ourselves to qualitative remarks.

First, however, we briefly characterise the performance for indirect rendering (GLX/X11 over an SSH tunnel to a local X display).

In what follows the departmental, home and laptop machines have hostnames **dclient**, **hclient** and **lclient** respectively, the graphics server is **gfxnode** and for simplicity the user accounts all have the same name: **abc123**. We will also assume that the HPC external interface is **loginnode** and that the clients can all see this across the network (if some need to approach through a gateway machine, this is a minor complication involving an additional tunnelling step).

4.5.2 GLX/X11 over the network (indirect rendering)

See Figure 1. For all three client types, connection was made to gfxnode via the following procedure assumed starting within a local X session (the Windows client using the SSH client and Xwin X server provided by Cygwin):

```
dclient $ ssh –N –f -L 22000:gfxnode:22 abc123@loginnode
dclient $ ssh -Y -p22000 abc123@localhost
Password:
gfxnode $ /path/to/smokeParticles
```

In all cases this reports:

```
[ CUDA Smoke Particles ]
freeglut (./smokeParticles): Unable to create direct context rendering for window 'CUDA Smoke Particles'
This may hurt performance.
Required OpenGL extensions missing.
```

smokeParticles is thus an example of a 3D code incompatible with indirect rendering. Celestia starts, but performs unusably in this mode, displaying at around 1 frame per second. On Windows/Cygwin furthermore it immediately crashes the Xwin server.

4.5.3 VirtualGL + TurboVNC

See Figure 4.
On gfxnode, start a TurboVNC session:

```
gfxnode $ /opt/TurboVNC/bin/vncserver
New 'X' desktop is gfxnode:1
Starting applications specified in /home/abc123/.vnc/xstartup.turbovnc
Log file is /home/abc123/.vnc/gfxnode:1.log
```

The default geometry of the virtual desktop is 1240x900, with colour depth 24. In the absence of compression, this suggests that each additional frame per second requires 3MB/s additional bandwidth, i.e. a 15 fps minimally acceptable frame rate would already require a gigabit network pipe (without compression).

Both Linux clients can then connect to this desktop as follows:

```
dclient $  /opt/TurboVNC/bin/vncviewer -via abc123@loginnode gfxnode:1
```

The user will be prompted firstly for the password or passphrase to loginnode, and then will be asked for the VNC password. Once the desktop window appears, press F8 to bring up the options popup panel.

On Windows we use the OpenSSH client provided by the Cygwin environment to set up the SSH tunnel explicitly – in a Cygwin command shell:

```
lclient $ ssh -L 5901:gfxnode:5901 abc123@loginnode
```

After authenticating to loginnode successfully, the Windows TurboVNC client can be started from the Start menu in the usual way, and a connection opened to localhost:1. Settings can be altered through the icons at the top of the viewer window.

The viewer window allows access to a virtual desktop session running on gfxserver (using a proxy X server which is not directly linked to the native X display on the server). In order to start a 3D application under VirtualGL:

```
gfxnode $ /path/to/vglrun –c proxy  /path/to/smokeParticles
```

The quality of VNC and (lossy) JPEG compression applied to the virtual desktop can be altered on the fly – more aggressive levels of compression reduce the network bandwidth required and improve frame rate and interactivity, at the cost of decreased image quality and increased server and client CPU load. Thus appearance can be sacrificed temporarily for improved responsiveness, and then restored when interactivity is not needed. There is also a "lossless refresh" function which refreshes the entire desktop with an exact image in order to remove any accumulating artefacts.

dclient (100Mbit/s Ethernet)

The experience here suggests that it may be possible to choose the `Perceptually Lossless' level of accelerated JPEG while maintaining usability, at least in the case of these two applications and in the absence of network contention.

smokeParticles

- Tight + Perceptually Lossless JPEG (95% quality, no chrominance subsampling)
  - 28.5 fps; good image, good interactivity.
- Tight + Medium Quality JPEG (80% quality + 2x chrominance subsampling)
  - 36 fps; good (not noticeably worse than previous case) image, good interactivity.
- Tight + Low Quality JPEG (30% quality, 4x chrominance subsampling)
  - 38 fps; noticeably blurry image and good interactivity.

celestia

- Tight + Perceptually Lossless JPEG (95% quality, no chrominance subsampling)
  - 21 fps; good image, but jerky and poor interactivity.
- Tight + Medium Quality JPEG (80% quality + 2x chrominance subsampling)
  - 28 fps; good image, good interactivity.
- Tight + Low Quality JPEG (30% quality, 4x chrominance subsampling)
  - 27.8 fps; noticeably blurry image and blurred text but good interactivity.

hclient (10Mbit/s Broadband)

The similar stories here suggest that the Medium quality accelerated JPEG is the best choice for 10Mbit/s home broadband, at least for these two applications.

smokeParticles

- Tight + Perceptually Lossless JPEG (95% quality, no chrominance subsampling)
  - 7.5 fps; good image, but jerky and poor interactivity.
- Tight + Medium Quality JPEG (80% quality + 2x chrominance subsampling)
  - 16.5 fps; good image, good interactivity.
- Tight + Low Quality JPEG (30% quality, 4x chrominance subsampling)
  - 25 fps; blurry image but good interactivity.

celestia

Tight + Perceptually Lossless JPEG (95% quality, no chrominance subsampling)
    7.5 fps; good image, but jerky and poor interactivity.
Tight + Medium Quality JPEG (80% quality + 2x chrominance subsampling)
    15 fps; good image, good interactivity.
Tight + Low Quality JPEG (30% quality, 4x chrominance subsampling)
    24 fps; blurry image and poor text but good interactivity.

lclient (50Mbit/s Wireless)

Quantitative frame rates were not obtained in this case due to the reliance of the tcbench utility on the X windows system (the Windows VNC client being a native Windows GUI application). Some qualitative remarks on usability follow - in general interactivity felt more variable than in the other two cases.

smokeParticles

Tight + Perceptually Lossless JPEG (95% quality, no chrominance subsampling)
    good image, but jerky and reasonable interactivity.
Tight + Medium Quality JPEG (80% quality + 2x chrominance subsampling)
    good image, slightly jerky but generally good interactivity.
Tight + Low Quality JPEG (30% quality, 4x chrominance subsampling)
    blurry image, sometimes jerky but good interactivity.

celestia

Tight + Perceptually Lossless JPEG (95% quality, no chrominance subsampling)
    good image, but very jerky and poor interactivity.
Tight + Medium Quality JPEG (80% quality + 2x chrominance subsampling)
    good image, jerky motion but acceptable interactivity.
Tight + Low Quality JPEG (30% quality, 4x chrominance subsampling)
    blurry image, poor text and irregular motion, but acceptable interactivity.

4.6 Issues

The vglrun launcher is a script which inserts VGL dynamic libraries in front of libGL when the 3D application is run, using the LD_PRELOAD mechanism. In this way calls to libGL can be intercepted, and the task of 3D rendering diverted to the 3D X server (where it is performed off-screen in so-called pbuffers). Some complex applications with a 3D component which are constructed from a hierarchy of scripts may already modify LD_PRELOAD or LD_LIBRARY_PATH, with the result that the graphical application may not see the VGL overrides; in these cases the application's scripts may need to be modified. Software for which this is an issue include COMSOL and STARCCM+.

The Windows implementation of the TurboVNC client does not have built-in SSH tunnelling corresponding to the -via command-line option of the Linux implementation; for Windows the SSH tunnel should be set up using an external program, e.g. putty or Cygwin OpenSSH.

5.0 Conclusion

The large data sets often resultant from HPC applications frequently require further processing by the owner, who is typically remote from the HPC local network. In many scientific disciplines such post-processing involves 3D graphical techniques, often requiring the services of a high-end graphics card for rendering complex 3D images in real time. The conventional approaches to this are either (1) perform all processing on the owner's local resources, requiring a heavy-duty, non-mobile workstation with adequate storage, processing power, memory and graphics hardware, plus local system support effort; or (2) perform the post-processing on the HPC server with remote graphics rendering taking place on the client workstation through SSH, X11 and GLX, which may work very slowly, or not at all. Similarly remote desktop techniques typically fail to support 3D applications, or if they do, do so only with non-accelerated software rendering, so also at best work slowly (probably unusably so).

The Dell | NVIDIA remote visualisation solution deployed here using open source software VirtualGL (VGL) allows a 3D application executing on a server with hardware-accelerated 3D rendering capabilities to offload the rendering task to the 3D hardware, producing a two-dimensional rendered image amenable to the usual SSH/X11 and Linux remote desktop methods. We have demonstrated in this whitepaper that when used together with a remote desktop package such as TurboVNC, VGL allows low-end clients, using multiple operating systems, to view and operate effectively 3D applications executing on a remote HPC server node equipped with a high-end graphics card. The improved codec provided by TurboVNC allows the network requirements between server and client to be mitigated dynamically so that a 10Mbit/sec bandwidth connection may be sufficient, offsetting image quality when appropriate against improved interactivity when this is required, and conversely. We have illustrated this using three systems (two Linux, and one Windows) with network connections simulating departmental, home broadband, and office wireless connectivity.

This allows HPC users "anytime – anywhere" visual access to large scale data sets thereby removing data locality issues and the requirement for high-end user workstations configured with complex visualisation software. Via the Dell | NVIDIA remote visualisation server users can now perform complex 3D visualisation of HPC data on the fly in the airport lounge or coffee shop on their laptop!