

# The design and implementation of a 50 million pixel Powerwall display

## Summary

Powerwalls are displays composed of multiple LCD panels aligned side-by-side to create a single display, but have far greater resolution than is achievable using any single display. This report describes a Powerwall that may be constructed using commodity hardware.

Compiled by:  
**John Hodrien, Jason Wood and Roy Ruddle**  
University of Leeds



Introduction .....	3
Visualization & Virtual Reality Research Group.....	3
System Design .....	4
Hardware .....	5
Master and Slave Nodes .....	5
Displays .....	5
Frame Construction.....	5
Software .....	7
Assumed Network Config .....	7
Operating System.....	8
Network Booting .....	8
DHCP .....	8
PXE .....	9
Clients .....	9
Server .....	9
NFS.....	11
Autologin .....	11
Shared Filesystems .....	12
Rendering APIs .....	12
Introduction.....	12
Chromium .....	13
Installation .....	13
Configuration.....	13
Experiences.....	13
VRJuggler .....	13
Installation .....	13
Configuration.....	14
Experiences.....	15
MPI_GL .....	15
Installation .....	15
Configuration.....	15
Experiences.....	15
Other Software Tools .....	16
Example Applications .....	16
Heart Scan Visualisation.....	16
Image Viewer.....	17
Iris Explorer .....	17
Costs and Installation Time .....	18
Reliability .....	18
Further Opportunities .....	18
Conclusions .....	19
References .....	19
Appendix A: Changes to the rc.readonly script.....	21

## Introduction

A Tiled Display (or Power) Wall is a large high resolution display. It is composed of multiple LCD panels aligned side-by-side to create a single display. The main advantage of such a wall is it has far greater resolution than is achievable using any single display. The large display area is also useful for collaborative work or multiple viewing participants. Tiled displays are in use at many sites internationally including the National Center for Supercomputing Applications [ 1], San Diego Supercomputer Center[ 2], and Pennsylvania State University[ 3].

The design of such a system is constrained by both the intended applications and the cost of the installation. In the past such systems would tend to have been powered using custom graphics hardware from companies such as SGI [ 5], yet it is now reasonable to build systems using commodity PC clusters [ 6] due to the increase in commodity graphics power, and the availability of cluster aware toolkits. If stereo is required then LCD panels are unsuitable (due to their slow refresh performance, and polarization characteristics) and rear projection CRT/DLP is typically used. This adds considerably to the cost due to the price difference between commodity LCD panels and typical active stereo projectors. Due to the network demands toolkits such as Chromium place on the system, gigabit Ethernet is an obvious choice.

Interacting with large tiled displays is entirely different from using a typical desktop machine, since it exposes problems that are not evident with a typical desk configuration. A standard WIMP design would tend to suffer when used on a wall as the user is likely to lose the mouse pointer (a standard size cursor is hard to see on a multi-million pixel display), can be disoriented by the presence of bezels, and also is likely to find interaction both fiddly and arduous as windows will appear great distances away from each other yet may still require accurate mouse interaction[ 7]. This can be handled through specific interfaces to the wall that use touch sensitive displays[ 8], or use of devices (e.g. wireless controllers) and interaction styles that are less demanding on the user.

Applications for these displays include flight simulation and visualisation of various high resolution image sets, especially astronomy data and satellite imagery [ 4]. NCSA have investigated the use of tiled display with meteorology, oceanography and hydrology [ 10]. At Leeds there is interest in using the wall for visualising various forms of medical data as well as complex transportation and infrastructure data.

At Leeds we have constructed two such displays: A 28 panel, 53 Mpixel wall approximately 3 metres across, and a smaller 12 panel, 23 Mpixel table approximately 1.7 metres across. The table is complemented by an Intersense IS-900 tracker, which enables millimetre accuracy 6-DOF (degree of freedom) tracking using interface devices commonly found in virtual reality applications. This report describes how such displays are constructed, and software that is useful to develop Powerwall applications and simplify their usage.

### *Visualization & Virtual Reality Research Group*

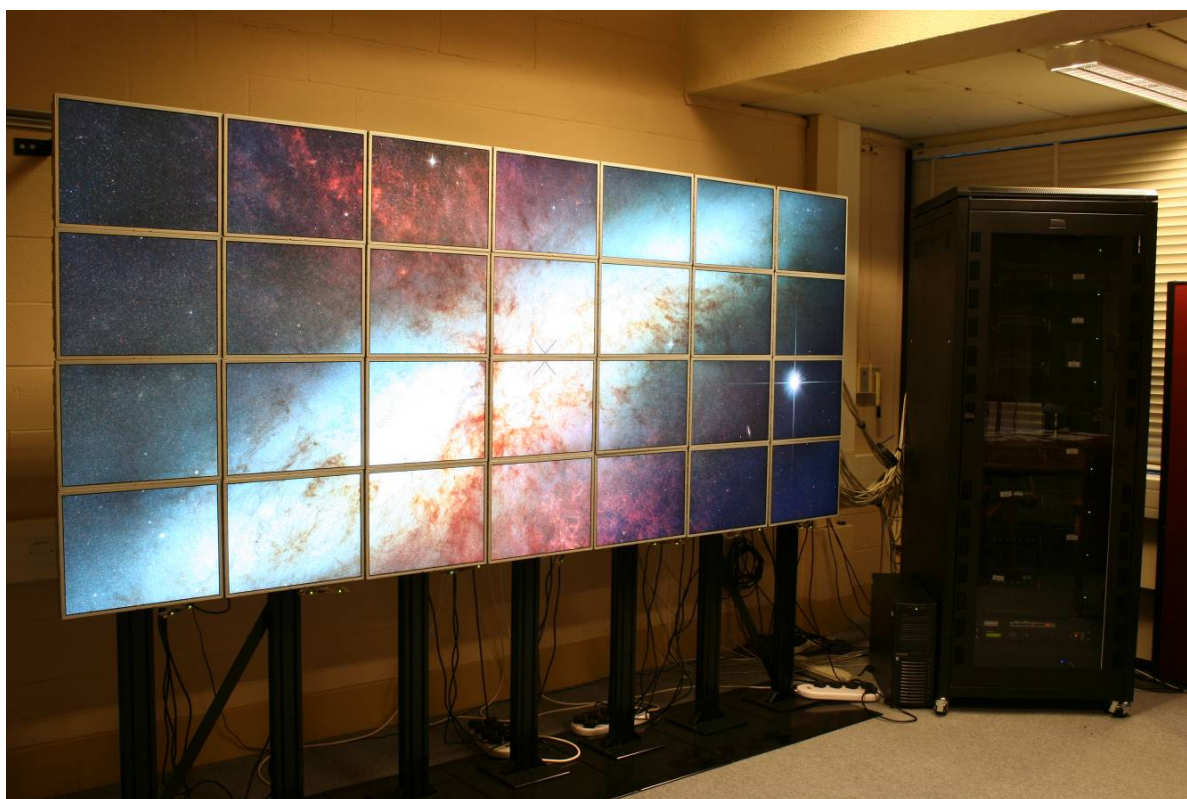
The displays were constructed by members of the Visualization & Virtual Reality (VVR) research group in the School of Computing at the University of Leeds. The **School of Computing** is an internationally-rated department (Grade 5 in the most recent UK Research Assessment Exercise). The **VVR group** comprises 13 academic staff, full-time researchers and PhD students whose work

covers both fundamental research in distributed systems, functional programming and human-computer interaction, and the application of visualization and virtual reality to domains that include medicine, bioinformatics, nano-engineering and transportation.

*John Hodrien* divides his time approximately equally between software development on e-Science projects (including DAME and HYDRA) and running the specialist facilities installed in the School's *Virtual Environment Laboratory for Interactive Simulation and Visualisation*. *Jason Wood* is a senior research fellow in the *Visualization and Virtual Reality* group. *Roy Ruddle* is a Senior Lecturer, and leads the human-computer interaction and virtual reality research.

## System Design

The setup designed at Leeds was constrained by the physical dimensions of the room. A 28 panel configuration (4 vertically x 7 horizontally) of 20-inch panels was chosen to maximise the usable space of the screen. This provides a screen area of 306 x 133 cm. To drive these, a total of 7 machines were required, using two dual-output cards per machine. All machines were networked using a gigabit Ethernet switch.



*Figure 1 Powerwall with the interactive image viewer*

## Hardware

### Master and Slave Nodes

<i>Component</i>	<i>Master Node</i>	<i>Slave Nodes</i>
Case	Supermicro 4U rack mount	Supermicro 4U rack mount
Motherboard	Tyan Thunder K8WE	MSI K8N Diamond
CPU	(2x) AMD Opteron 270	AMD Athlon X2 4400+
Graphics Card	(2x) XFX nVidia 7800 GTX	(2x) XFX nVidia 7800 GTX
PSU	Tagan TG480-U22	Tagan TG480-U22
RAM	(8x) 1Gb Corsair Registered DDR400	(4x) 1Gb Corsair Unregistered DDR400
Hard Disk	(2x) WDC WD800JD-08LS (80Gb)	None

Racking the machines was straightforward apart from the need to support the cables. Due to the length of the cables a considerable weight hangs off the graphics cards if simply plugged in with no additional support. Velcro cable ties were used to attach the cables to the rack, removal all load from the cards themselves. Previous work has shown that graphics card connectors are particularly vulnerable to weight induced damage.

### Displays

These are standard LCD panels (Dell 2001FP). 20 inch panels are typically the first screen size that runs at 1600x1200 but are still relatively cheap. The cases of these panels can be removed leaving a metal frame and a relatively narrow bezel. Standard VESA mounting holes are hidden under the plastic at the rear. 30 inch Dell panels (3007WFP) were considered, but ruled out due to cost.

### Frame Construction

The frame is constructed from lightweight aluminium with steel bases for stability. Diagonal cross braces provide stiffness for keeping the posts the correct distance apart. Each monitor is mounted on a sliding plate that is tightened using levers accessible from the rear allowing individual adjustment of their vertical placement. Building the frame in sections allowed easy transportation of the frame from the workshop in which it was custom made, into the room where there was only a standard size door for access.



*Figure 2 Front view of the powerwall*



*Figure 3 Side view of monitor mount*



*Figure 4 Rear view showing bracing*

The power supplies for the monitors need to be housed at the rear of the frame assembly. Partly this is due to a desire to keep the floor clear of unnecessary clutter, but also is the result of the power supplies coming with cables insufficiently long to reach the floor. Currently these are simply stood on flat sections of the frame, but these could be zip tied for further security.

### *Software*

The design of the hardware took into account a desire to minimise the noise created by the full rack of machines. This meant that the slave machines were not installed with hard drives. This presents a challenge for the system configuration beyond the norm, since the slaves need to boot from the head node. DHCP allows the machine to boot from the network and discover the head node. PXE provides a kernel and initial ram disk to the slave to boot from. NFS then provides all the files the slave machines need to boot fully. Direct interaction with the slave machines is also to be avoided, so setting the machines up to automatically start up into a state where you can use them is important.

### **Assumed Network Config**

<i>Machine name</i>	<i>IP-Address</i>
Head	10.0.0.1
slave1	10.0.0.2
slave2	10.0.0.3
slave3	10.0.0.4
slave4	10.0.0.5
slave5	10.0.0.6

<i>Machine name</i>	<i>IP-Address</i>
slave6	10.0.0.7

## Operating System

Fedora Core 5 (64bit) was used as the OS for all nodes. Having the same OS on the head node and slaves minimises any problems you will encounter with running software across the cluster. 64bit was chosen as the head node has 8Gb of RAM and this is not efficiently handled with 32bit versions. Fedora Core was chosen since it is the standard Linux used within the school and is well supported.

## Network Booting

### DHCP

A DHCP server was already present on the network, so the following content was added:

```
ddns-update-style none;
ignore client-updates;

subnet 10.0.0.0 netmask 255.255.255.0 {
    option routers          10.0.0.102;
    option subnet-mask     255.255.255.0;
    option domain-name-servers 10.0.0.101;

    default-lease-time 21600;
    max-lease-time 43200;

    group {
        next-server 10.0.0.1;
        filename "power-wall/pxelinux.0";

        host slave1 { hardware ethernet 00:11:09:d5:05:16;
                      fixed-address 10.0.0.2; }
        host slave2 { hardware ethernet 00:11:09:d3:3b:ea;
                      fixed-address 10.0.0.3; }
        host slave3 { hardware ethernet 00:11:09:d4:fc:03;
                      fixed-address 10.0.0.4; }
```

```
host slave4 { hardware ethernet 00:11:09:d3:3e:19;
               fixed-address 10.0.0.5; }
host slave5 { hardware ethernet 00:11:09:d3:3c:37;
               fixed-address 10.0.0.6; }
host slave6 { hardware ethernet 00:11:09:d3:41:cf;
               fixed-address 10.0.0.7; }

}
}
```

## PXE

To configure PXE you need a kernel and an initrd suitable for booting your machines, and you need to configure the clients to PXE boot.

## Clients

Making the clients PXE boot is simple. Enter the BIOS and ensure that the nVidia MAC ROM is enabled. Save the settings and restart. Reenter the BIOS and set the boot order so that nVidia MAC is the first option.

## Server

Configuring the server is harder work. Ensure you have the tftp-server package installed, and that it is enabled. Install the syslinux package for the PXE loader. In /tftpboot/power-wall we are going to collect together all the things that are required to network boot a machine. The PXE loader (pxelinux.0) should be copied there along with a suitable configuration file in pxelinux.cfg/default:

```
default DemoSystem

label DemoSystem
    kernel system/vmlinuz
    append ramdisk_size=200000 initrd=system/initrd.img
    NFSROOT=10.0.0.1:/images/FC5
```

The last remaining task is to put together the kernel and initrd suitable for booting. To keep things in order both of these can be placed in a 'system' directory within power-wall. The kernel should be the same as the head node is booting from and so can simply be copied from /boot/vmlinuz-2.6.17-1.2139\_FC5 (or equivalent). The initrd is much harder to make, requiring changes to allow network booting.

1. Take the initrd you already have in /boot and expand it with cpio:

```
mkdir stock
zcat /boot/initrd-2.6.15-1.1833_FC4smp.img | \
(cd stock;cpio -i)
```

2 Put the kernel modules you need for network and NFS into the stock/lib directory. For our systems this was forcedeth.ko, sunrpc.ko, lockd.ko, nfs\_acl.ko and nfs.ko. Then you have to modify the initrd to load these modules and bring the network up. The changes are shown below along with some more to be explained:

```
echo "Loading forcedeth.ko module"
insmod /lib/forcedeth.ko
echo "Loading nfs modules"
insmod /lib/sunrpc.ko
insmod /lib/lockd.ko
insmod /lib/nfs_acl.ko
insmod /lib/nfs.ko
/sbin/udevstart

ifconfig eth0 up

dhclient -cf /etc/dhclient.conf -pf /tmp/dhclient.pid -lf
/tmp/dhclient.leases eth0

echo Creating root device
mkrootdev /dev/root
echo Mounting root filesystem
/bin/mount -n -o nolock,ro $NFSROOT /sysroot
mount -o mode=0755 -t tmpfs /dev /sysroot/hack
cp /tmp/dhclient.leases /sysroot/hack/
```

So once the modules are loaded the network must be brought up, and a DHCP lease requested. Then you mount the nfs root onto /sysroot. The only remaining hack required is to pass your dhcp lease into the new sysroot so your newly booted system doesn't need to do a whole new dhcp request. If it did, you'd lose the network and it'd lose its root filesystem. Using the tmpfs filesystem gets around the problem.

3. All that remains is to bundle that back into an initrd and the PXE boot should succeed:

```
cd stock; find . | cpio -co | gzip -nc > ../initrd.img
```

## **NFS**

Initially a machine was installed with a basic Fedora Core 5 image, sufficient for running X windows, and with the nVidia device drivers installed. This image was then copied onto the head node, so it could be shared across the network to the slave machines. This image was placed in /images/FC5, and entries were made in /etc/exports to create the NFS export for the slaves:

```
/images/ 10.0.0.2(ro,no_root_squash) 10.0.0.3(ro,no_root_squash) \  
10.0.0.4(ro,no_root_squash) 10.0.0.5(ro,no_root_squash) \  
10.0.0.6(ro,no_root_squash) 10.0.0.7(ro,no_root_squash)
```

The system will now successfully PXE boot, pickup the kernel boot image and initial ram disk, successfully DHCP for an IP address, mount the NFS root, begin to boot, and then fail. At the moment the system will simply boot and kernel panic as it is unable to write to the boot image, and has nowhere else to write to. The basis of the fix for this came from the Stateless Linux project from Fedora, but when this was configured the system had been mostly abandoned\*. The basic idea is to create a tmpfs filesystem and then copy and bind-mount over the bits of the filesystem that you need to write to (such as a large proportion of /var). The rc.readonly script used is in Appendix A. In addition this project provides busybox which provides facilities for bringing the network up easily.

At this point all machine should boot up to the login prompt.

## **Autologin**

Getting X to automatically login is straightforward. Simply edit /etc/gdm/custom.conf for the machines so it will login as a given user (create one for this task):

```
[daemon]  
AutomaticLoginEnable=true  
AutomaticLogin=autoxlogin
```

You can then create an .xsession file for this user to perform any tasks you want for the screens:

```
xhost +  
xset s off  
xset -dpms  
sleep 10000
```

The final sleep simply makes the login script hold at that point (for 10000 seconds). Without it the X login would simply terminate and return you to the login prompt.

There are security considerations with this model (since any user logged into the head node is able to access the X servers running on the slave machines. Removing ssh access for the machines for normal users should be considered (thus limiting access to the console), along with firewalling the slave nodes as necessary.

## Shared Filesystems

In addition to the basic OS NFS share, you may want to consider shared space between the machines. Sharing `/usr/local` makes it easy to ensure the Chromium/vrjuggler installs are kept in sync between the nodes, and having some form of `/scratch` or shared `/home` makes life much easier to work with the machine.

## Rendering APIs

### *Introduction*

Three methods of running applications have been investigated: *Chromium*, *VRJuggler*, and a locally developed library *MPI\_GL*. Chromium was developed from WireGL, a research project at the Stanford University Computer Graphics Lab. It is licensed under a BSD-like license. VRJuggler was developed by Iowa State University and is licensed under the LGPL. Both tools support Windows (95/98/200/XP) and POSIX UNIX Oss (BSD/Linux/OSX/IRIX/Solaris). MPI\_GL is currently unreleased.

Chromium is a tool that wraps existing programs which has the advantage that it works with unmodified programs. It does this by intercepting the OpenGL calls and distributing them to the machines of the cluster to render. This minimises the effort required to make software take advantage of the power wall, but also imposes fairly significant restrictions.

VRJuggler is an application framework that replaces GLUT in the traditional OpenGL software stack. Rewriting the application is therefore necessary, although the work involved is fairly minimal. Window creation, viewport/camera management, and input handling are all different from GLUT. The benefit is that you can write an application that works on a desktop or a power wall with no modification to the code. A config file is supplied at runtime which describes the system in use, and input and shared variables are distributed amongst the cluster nodes.

MPI\_GL is a library that was developed to work around limitations in the performance of Chromium, by being even more restricted in its scope. MPI\_GL effectively takes a normal OpenGL program, and turns it into an MPI program, synchronising and modifying function calls to enable an unmodified program to work on the tiled display.

## *Chromium*

### **Installation**

Installing Chromium is straightforward. Untar the beta package available from <http://chromium.sourceforge.net/beta/>. If you want DMX support, edit options.mk to enable this. A simple ‘make’ builds Chromium, but installation is up to the user. A nearly complete cr.spec file is provided for those wanting to create RPMs.

### **Configuration**

Chromium comes with configtool.py, a tool for creating a configuration that suits your wall. This works fine if all your display machines create identical shaped tiles, but requires a small amount of work if this is not the case.

Chromium can be configured to automatically intercept OpenGL and render it on the wall:

1. config file has ("auto\_start", 1) and ("auto\_start\_apps", 1)
2. LD\_PRELOAD=/usr/local/cr-1.9/lib/libcrfaker.so is defined for your environment
3. ~/.crconfigs has a default line:  
\* ~/wall-config/chromium/auto/auto-nogaps-fullscreen.conf -m %m %p

This will call Chromium with the supplied config when running an application that matches the first parameter (\*). Specific configurations for applications that do not work optimally with the base configuration can be specified above this “catch all” entry.

### **Experiences**

Chromium is very demanding on the network, especially the head node. When viewing a scene that does not use display lists the network traffic can be considerable. Certainly using a 100Mbit network is not recommended, and even a 1Gbit network is minimally satisfactory. If possible configure your setup with a 10Gbit connection for the head node, and 1Gbit for each of the slaves. An alternative would be to use multiple Gigabit connections to the head node. When the network is heavily loaded there can also be synchronisation issues between the screens.

## *VRJuggler*

### **Installation**

Installing VRJuggler is made difficult by the fact that it doesn't insist on all dependencies being present to build, and as such it's quite easy to half build the tool and not notice. At least the packages below are required beyond the default Fedora install (in brackets the version number tested):

cppdom (0.6.4)

omniORB (4.1.0-beta2)

gmtl (0.4.11)

JDK (1.5.0\_07)

After untarring you have to call `configure.pl` with sufficient parameters to let it find the required libraries:

```
./configure.pl --with-boost-fs-lib=boost_filesystem --with-boost-  
includes=/usr/include/ --with-boost=/usr/ --with-gmtl=/usr/
```

```
make build install >& install.log
```

After that you should have a full install in `/usr/local/`. If anything fails to install correctly review the `install.log`.

## Configuration

The first step with VRJuggler is to create a display configuration for the power wall. VRJConfig is a tool provided for this. Creating the configuration is easy:

1. Add a ClusterNode. This represents a machine in your system. Configure the name of the machine, and the coordinates of its display surface in physical space. Repeat for the other machines.
2. Add a ClusterManager. Add into the `cluster_node` section all the machines you have just defined. Then add additional plugins you will want:
  - a) RIMPlugin - shares input devices between the nodes
  - b) StartBarrierPlugin – waits for all nodes to be ready before starting up
  - c) SwapLockTCPPlugin – blocks on swap: nodes will be at most one frame out of sync.
3. Add relevant devices to the configuration. This is well documented in the vrjuggler notes.

If your display coordinates are similar to those used with the provided examples you should be able to simply run one to test your config:

```
#!/bin/bash
```

```
HOSTS="slave1 slave2 slave3 slave4 slave5 slave6"
```

```
COMMAND="./demoApp total.jconf"
```

```
DIR=`pwd`
```

```
for i in $HOSTS
```

```
do
```

```
    ssh $i "export VJ_CFG_PATH=/usr/local/VR/config  
DISPLAY=:0.0 __GL_SYNC_TO_VBLANK=1;cd $DIR;$COMMAND"&
```

done

\$COMMAND

So to run an application you simply have to start an instance on each of the machines, and they setup the communication as specified in the configuration file. The DISPLAY needs to be forced to :0.0 to ensure the processes attempt to display to the correct screen.

## Experiences

So far the results with VRJuggler have been very promising. The effort required to convert a program is minimal and is well documented with numerous examples provided with VRJuggler as distributed. With a short program or a longer program that's well structured the changes will take around 30 minutes. The time to modify a program really does depend on how embedded GLUT calls are within your code.

Once your code is running it has the advantage of greatly reduced network traffic, compared with Chromium. In fact the minimum required traffic with VRJuggler is in the order of hundreds of bytes per frame which on a modern high-speed network is irrelevant. Currently this has only been tested with a single machine (and X server) per machine, although it could be interesting to note any performance variation by splitting it into two and having a separate process per thread. Certainly that would make the vertical sync on each monitor correct, which is currently not possible to guarantee (as that would require framesync between the two cards).

## MPI\_GL

### Installation

Since this is currently pre-release software there is no installation beyond a Makefile. A single source file is used to generate the library.

### Configuration

A program is run using LD\_PRELOAD to force loading of MPI\_GL, and other environment variables are used to assist the library. Mpirun is used to spawn jobs across the cluster. Example:

```
Mpirun -np 8 -x \  
ROWS=2,DISPLAY=:0.0,LD_PRELOAD=/usr/local/lib/MPI_GL.so \  
/usr/libexec/xscreensaver/skyrocket
```

## Experiences

The use of this library is extremely limited. Only single window OpenGL applications will work, and input is currently limited to GLUT applications. OpenGL programs that create their own viewing matrix without using standard calls (e.g. glOrtho) will not be correctly tiled. Due to the

way the library works, it is impossible to guarantee which programs will function correctly.

Performance of the library when it does work is very good, offering equivalent full wall performance to that of a single 3200x2400 window. Synchronisation appears to be excellent, certainly comparable to Chromium. Tested against the xscreensavers, rss-glx demos, and two other glut applications demonstrated good compatibility, with 80% of the external programs functioning, and both glut applications (with input) functioning well.

### ***Other Software Tools***

XDMX was looked at, but the performance was sufficiently poor that this has not been investigated further. XDMX allows you to create a logical X server across multiple X servers. This lets you treat the entire wall as a desktop and drag windows around freely. This also integrates with Chromium allowing you to gain the performance characteristics of Chromium while having superior window management. A future development in X is in the pipeline that may make this use of Chromium unnecessary. Testing on 4 screens suggested that XDMX had potential, but by the time it was running on all 28 screens the latency of interactions outweighed the benefits.

## **Example Applications**

### ***Heart Scan Visualisation***

Integrative Biology is an e-Science funded project investigating the causes of heart disease and how cancers grow, which together kill over 60% of the UK population. As part of this project Dr Peter Kohl of Oxford University's Department of Physiology has MRI scans (resolution 1024x1024x2048 pixels) of rabbit hearts, together with histology slices (1800 slices, each up to 8192x8192 pixels) of the same hearts.

To view this data on the power wall we have used VTK to construct isosurfaces in the MRI data, which were then smoothed and decimated. A custom written VRJuggler application was then used to load these surfaces. Histology slices were given transparent backgrounds and have been added to the MRI visualisation to provide the user with better context of what they are seeing. Through use of a wireless joypad navigation in 3-d space, plus control of clipping planes are easily achieved.



Figure 5 The powerwall displaying the MRI visualization

### ***Image Viewer***

An OpenGL image viewer was created that can load an image and render it on the wall using VRJuggler. Compressed textures were also experimented with as a way of cutting down memory usage of the graphics cards, although this either places a burden on the graphics cards at runtime (causing stuttering) or requires a preprocessing step. Currently this has been tested with images up to 170000x100000 pixels although there are no architectural limits other than those imposed by the use of the TIFF format.

Images are first converted into pyramid tiled TIFF files, and then are displayed on the power wall using VRJuggler and a pixel-perfect scaling system, i.e. one pixel on the power wall is equal to one pixel in the image. Images are loaded dynamically, streaming across the network from an NFS server. A wireless joystick provides controls for the user, allowing free movement in front of the screen. This provides the user with the ability to pan and zoom with analogue controls for each. See Figure 1 for a photo of this in action.

### ***Iris Explorer***

A basic test of running Iris Explorer with Chromium was performed. Using the autostart behaviour of Chromium proved adequate to intercept the OpenGL from the spawned processes and the rendering was correct. The performance was poor however, due to the lack of display lists and the resulting high network traffic.

## Costs and Installation Time

Item	Quantity	Unit Cost	Total Cost
Frame	1	£5000	£5000
Monitors	28	£330	£9240
Slaves	6	£2000	£12000
Head node	1	£2500	£2500
Joypad	1	£25	£25

Item	Time Cost
Removing monitors from their cases	5 hours
Mounting them and assembling the frame	8 hours
Building and troubleshooting PCs	11 hours
Racking and wiring machines	3 hours
Developing software stack	300 hours

## Reliability

Due to the large number of components it was always expected that failures would occur in the system. Over the first year this has been more than expected. Two power supplies failed soon after installation in an identical manner. This was assumed to be a manufacturing defect and no further PSU failures have been experienced. Two motherboards failed in the slaves, again in an identical manner, but after a while in service. All four of these failures occurred while the machines were powered down. Additionally a graphics card failed and began to render incorrect polygons. Overheating could be the cause of all failures, although due to the relatively short periods the machines are on for, and the reasonable ventilation, this is far from certain.

## Further Opportunities

In addition to the main power wall, an additional power-table has been constructed. While being physically smaller (4 x 3 monitor arrangement) it is in other respects very similar in design to the main wall. This has been placed within an Intersense IS-900 tracked region. The high resolution of this 6-DOF tracking system (0.75mm / 0.05°) combined with the high resolution of the display should prove to be compelling for interacting with high resolution data sets.



Figure 6 The powertable with IS900 trackers

## Conclusions

There are limitations to building a system like this on a budget. Since there is no frame locking there can be a small amount of synchronisation error between screens. At best the screens will be less than a full frame out of sync, which for many applications is acceptable. To use frame sync and achieve similar performance would mean using Quadro 4400 FX graphics cards with a G-Sync daughterboard. That probably equates to an extra £25k to the cost. Alternatively there are combined hardware and software solutions that can be made to sync the machines more accurately using a kernel module and a custom solution connecting the parallel ports together. This may be investigated in the future. The overall performance of the system is excellent however. The final image is bright and colourful, offering a much richer display than is typical of projected systems. With VRJuggler and MPI\_GL offering little overhead, the frame rates achieved are also good, although more work with scene graph based libraries may be necessary to gain the scalability possible with cluster rendering. Chromium while limited in performance offers a usable experience with a 1Gbit/s network. Previous studies on a 100Mbit network suggested that the performance of Chromium would be almost unusable offering performance up to 98% slower than with VRJuggler.

## References

1. NCSA's tiled display wall [<http://cave.ncsa.uiuc.edu/tiledisplaywall.pdf>]
2. SDSC's tiled display [<http://www.sdsc.edu/~moreland/projects/HDD/HDD.html>]
3. PSU's tiled display wall [<http://gears.aset.psu.edu/viz/facilities/displaywall/>]
4. "GeoWall: Stereoscopic Visualization for Geoscience Research and Education", Johnson, A.; Leigh, J.; Morin, P.; Van Keken, P. IEEE Computer Graphics and Applications, 2006, Vol. 26 Issue 6 10-14
5. "Introduction to building projection-based tiled display systems", Hereld, M.; Judson, I.R.; Stevens, R.L. IEEE Computer Graphics and Applications, 2000, Vol. 20 Issue 4 22-28
6. "An Overview of Cluster Solutions for Immersive Displays", Steed, A; Gelncross, M; Bierbaum, A. Presence, Vol. 12, No. 4, August 2003, 437-440

7. “The Large-Display User Experience”, Robertson, G; Czerwinski, M; Baudisch, P; Meyers, B; Robbins, D; Smith, G; Tan, D. IEEE Computer Graphics and Applications, 2005, Vol. 25 Issue 4 44-51
8. “View and Space Management on Large Displays”, Bezerianos, A; Balakrishnan, R. IEEE Computer Graphics and Applications, 2005, Vol. 25 Issue 4 34-43
9. “A Survey and Performance Analysis of Software Platforms for Interactive Cluster-Based Multi-Screen Rendering”, Stadt, O; Walker, J; Nuber, C; Hamann, B. Ninth Eurographics Workshop on Virtual Environments 2003
10. Applications of Tiled Displays [<http://vis.iu.edu/Publications/MeteorWall.pdf>]

## Appendix A: Changes to the rc.readonly script

```
mount_empty_dir() {
    if [ -e ${1} ]; then
        mkdir -p /var/writable${1}
        mount -n --bind /var/writable${1} ${1}
    fi
}

mount_files_onlydirs() {
    if [ -e ${1} ]; then
        mkdir -p /var/writable${1}
        find ${1} -type d -print0 | cpio -p -0vd /var/writable &>
/dev/null
        mount -n --bind /var/writable${1} ${1}
    fi
}

mount_files() {
    if [ -e ${1} ]; then
        find ${1} -print0 | cpio -p -0vd /var/writable &>
/dev/null
        mount -n --bind /var/writable${1} ${1}
    fi
}

mount -t tmpfs none /var/writable

# Set up temporary file directories
mount_empty_dir /tmp
chmod a+rwxt /var/writable/tmp
mount_empty_dir /var/tmp
chmod a+rwxt /var/writable/var/tmp

mount_files_onlydirs /var/gdm
mount_files_onlydirs /var/lock
mount_files_onlydirs /var/log
```

```
mount_files_onlydirs /var/run

mount_files /etc/fstab
# Having a writable mtab but unwritable /etc seems to break due to
locking
#mount_files /etc/mtab

# Can be changed by dhcp:
mount_files /etc/resolv.conf
mount_files /etc/ntp.conf
mount_empty_dir /var/lib/dhclient
if [ -f /initrd/tmp/dhclient.leases ]; then
    cp /initrd/tmp/dhclient.leases /var/lib/dhclient/dhclient-
eth0.leases
    echo found the leases
fi
if [ -f /hack/dhclient.leases ]; then
    cp /hack/dhclient.leases /var/lib/dhclient/dhclient-eth0.leases
    echo found the leases
fi

# Changed by ntp
mount_files /etc/adjtime

# Changed by kudzu
mount_files /etc/modprobe.conf
mount_files /etc/sysconfig/hwconf
mount_files /etc/X11/xorg.conf

# Clean out hardware state so we start fresh
> /etc/modprobe.conf
> /etc/sysconfig/hwconf

#Additions to the default confing
mount_files /etc/resolv.conf.predhclient
```

```
mount_files /var/lib/random-seed
mount_files_onlydirs /var/lib/nfs
mount_files /.autorelabel
mount_files /root
#End of changes

# FIXME: should we remount this?
#umount /initrd/var/lib/nfs/rpc_pipefs

umount /initrd/tmp
umount /initrd
umount /hack

# scan for local swap
/usr/sbin/scanswap
```