

Implementation of large-scale 3D heart data visualization

Jeremy Walton¹, NAG Ltd

1 Introduction

This note describes the way in which a large-scale visualization of 3D heart data was ported to the video wall in the Oxford e-Research Centre (OeRC). The original version of the visualization program was written by Dr Christopher Goodyer for the powerwall at the University of Leeds. It displays isosurfaces of MRI data stacks obtained from a high-resolution scan of a rabbit heart; this data was supplied by Dr Peter Kohl at the University of Oxford. Porting the program was requested by OeRC in order to facilitate wider dissemination of the visualization work.

Our article is arranged as follows. We first present (§2) an account of the motivation for this work, including some scientific background, followed by a brief description of the OeRC video wall's hardware (§3.1) and software (§3.2) configuration. There then follows (§4) a lengthy account of the porting process, including a description of the necessary software stack required for the building and running of the visualization application (which is called *oView*). Next, we describe how *oView* is run, including a discussion of its documentation (§5.1) and input files (§5.2), along with some specific details of how to run it on the OeRC video wall (§5.3). We conclude (§6), with an outline of future work which could be done with *oView*, and some observations on ways in which this work could be extended to the display of other types of data.

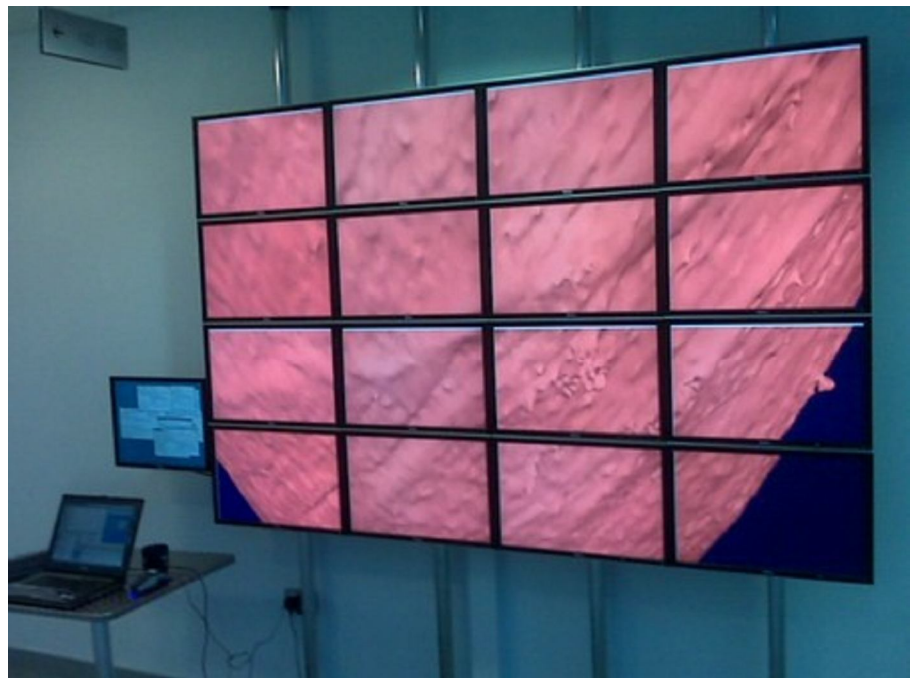


Figure 1: Using the OeRC video wall for large-scale display of heart data. This view shows the exterior of the heart.

2 Motivation and background

Scientists at Oxford University are building computer models of the heart in order to further basic science understanding of the heartbeat, and to help make the diagnosis and treatment of cardiac disease more efficient. In the future, the models will be used to simulate alternative surgical scenarios for the patient, such as the re-opening of blocked vessels, or the fitting of stimulation leads, so that surgeons will be able to

¹ Jeremy.Walton@nag.co.uk

make more informed decisions about their operating procedure. Experimental data forms part of the input to the scientists' models, and visualization of this data is required for better understanding of relevant structures.

The experimental data underlying this visualization comes from a BBSRC-funded collaboration between the teams of Dr Peter Kohl² (Department of Physiology, Anatomy and Genetics), Dr Jurgen Schneider³ (Department of Cardiovascular Medicine) and Professor David Gavaghan⁴ (Computing Laboratory). It comprises a stack of images generated from a sequence of high-resolution magnetic resonance imaging (MRI) scans of a heart, which are then segmented to discriminate between cardiac structures. The visualization software, developed by Dr Christopher Goodyer⁵ of the School of Computing⁶ at the University of Leeds (as part of the EPSRC-funded Integrative Biology project⁷) calculates isosurfaces through the segmented MRI data, generating 3D surfaces of tissue structure.

The ever-increasing resolution of experimental devices such as the MRI scanner is leading to larger datasets (for example, the heart data has a size of 1.5 GB) which cannot be viewed on conventional displays without loss of detail. Accordingly, the tissue surfaces were originally shown on the high-resolution display wall at Leeds (a visualization⁸ that won Dr Goodyer and his colleague Professor Ken Brodrie⁹ first prize in the vizNET Showcase 2008¹⁰). With the help of the Leeds team, we ported their application from their installation to the video wall at OeRC, which facilitates its use by the Oxford scientists, and promotes its wider dissemination.

Large-scale displays such as the OeRC video wall are built by tiling together several conventional LCD screens (see figures) that are powered by a compute cluster. Programs which use the wall have to handle the distribution of the scene to be rendered across the compute nodes so that each part of the display is coordinated with its neighbours. They also have to respond to commands from user-controlled input devices which allow the user to move the camera viewpoint and interact with the scene. A fixed camera position generates a static view, whilst programming a sequence of camera positions results in a flight through the



Figure 2: As Figure 1, but now showing the interior of the heart

² <http://www.physiol.ox.ac.uk/~pk/>

³ <http://www.cardiov.ox.ac.uk/staff/jurgenschneider>

⁴ <http://web.comlab.ox.ac.uk/people/David.Gavaghan>

⁵ <http://www.comp.leeds.ac.uk/ceg/>

⁶ <http://www.engineering.leeds.ac.uk/comp/>

⁷ <http://www.integrativebiology.ox.ac.uk/>

⁸ <http://www.comp.leeds.ac.uk/ceg/IB3dviewer.htm>

⁹ <http://www.comp.leeds.ac.uk/kwb/>

¹⁰ <http://www.viznet.ac.uk/node/130>

3D scene (see, for example, the video featured in the article¹¹ on the BBC website which describes this work).

The high resolution of the display means that it can be used to visualize very large datasets such as the 3D heart; typical scenes displayed here contain tens of millions of triangles. Moreover, the physical extent of the display (which, if the user stands close enough, can fill their field of view) leads to a compelling sense of the viewer being "inside" the scene, and also means that fine details can be explored without losing contextual information about location within the dataset. Other aids to understanding and orientation such as text labels and navigational reference points can be incorporated into the visualization.

3 The OeRC video wall

3.1 Hardware

The video wall at OeRC consists of a cluster of nine machines. The first of these is named `swan`, and acts as the headnode for the cluster, while the remaining eight are named `tile-0-0`, `tile-0-1`, `tile-1-0`, `tile-1-1`, `tile-2-0`, `tile-2-1`, `tile-3-0` and `tile-3-1`. The headnode controls a single physical screen, while each of the `tile` nodes controls two screens, arranged one above the other. These sixteen screens are arranged in a four by four grid (see figures), and the indices in the `tile` node names reflect the physical coordinates (column number, row number) of the screens in the grid.

Each screen's display has a width of 1920 pixels and a height of 1200 pixels, so each of the `tile` nodes controls a virtual display whose resolution is 1920 by 2400, and the video wall (excluding the headnode display) has a resolution of 7680 by 4800, or *circa* 37 megapixels.

3.2 Software

The cluster is currently running Linux, and incorporates the so-called Viz Roll¹² from the Rocks open-source Linux cluster distribution¹³. The Viz Roll includes various visualization middleware components such as DMX, Chromium and SAGE, although we do not make use of these in this work.

Instead, the *oView* application is built on top of the VR Juggler toolkit. This is an open-source framework for the development of virtual reality applications; typically, the application developer uses VR Juggler as a replacement for GLUT (the OpenGL Utility Toolkit, which provides a portability layer shielding developers from operating system and window system dependencies). Here, OpenGL is a widely-used standard API for 2D and 3D computer graphics. VR Juggler therefore provides a means for porting an OpenGL application from the desktop to a flexible virtual reality environment. Although it is necessary to rewrite the application to replace GLUT, one of VR Juggler's features is that it allows the same executable to be run on a wide variety of hardware (ranging from the desktop to complex VR systems) without modification: instead, the hardware details are read at run-time from a configuration file (see §5.2.3, below).

4 Building oView

This section describes how the *oView* application was built for the OeRC video wall. We log onto `swan`, the headnode of the cluster as `demo`, and do all development work in `/home/demo/jeremy` (the environment variable `MYHOME` is set to this location, and is used in commands below).

The required software stack for *oView* contains several packages which must be built and installed; an initial list came from the *oView* documentation (specifically, the file `$MYHOME/builds/oView/INSTALL`), although additional packages were also required. This table gives details of the locations from which the packages' source distribution was downloaded:

¹¹ <http://news.bbc.co.uk/1/hi/health/7774016.stm>, <http://news.bbc.co.uk/1/hi/health/7774413.stm>

¹² <http://www.rocksclusters.org/roll-documentation/viz/5.0/index.html>

¹³ http://www.rocksclusters.org/wordpress/?page_id=57

Distribution	Location
boost_1_35_0.tar.gz	http://www.boost.org/users/download/version_1_38_0
scons-0.98.5.tar.gz	http://www.scons.org/download.php
scons-addons.tar.gz	https://realityforge.vrsource.org/view/SconsAddons/WebHome
cppdom-0.7.10.tar.gz	http://sourceforge.net/project/showfiles.php?group_id=52718
gmtl-0.5.4.tar.gz	http://sourceforge.net/project/showfiles.php?group_id=43735
flagpoll-0.9.1.tar.gz	http://sourceforge.net/projects/freshmeat_flagpoll/
Doozer-2.1.6.tar.gz	http://sourceforge.net/project/showfiles.php?group_id=20758
vrjuggler-2.2.1-1-src.zip	http://sourceforge.net/project/showfiles.php?group_id=8041&package_id=8163&release_id=581899
changeset_r21069.zip	http://developer.vrjuggler.org/changeset/21069
ImageMagick-6.4.1-10.tar.gz	http://www.imagemagick.org/www/download.html
glf_1.4-1.tar.gz	http://www.imem.unavarra.es/3d_mec/download/sources/
oView.tar.gz	(obtained from Chris Goodyer)

Following download, the distributions were placed in the folder `$MYHOME/tarfiles`. Next, they were unpacked into the folder `$MYHOME/builds`, and built in there; the home directory for installs was `$MYHOME/juggler`. This is specified below in the `--prefix` option to the `configure` and `setup.py` commands; if this option isn't specified, the software is usually installed into `/usr/local`, which might raise issues associated with file permissions.

The builds were done in the following order.

4.1 Boost

```
cd $MYHOME/builds
tar xzvf ../tarfiles/boost_1_35_0.tar.gz
cd boost_1_35_0
./configure --prefix=$MYHOME/juggler
make -j 12
make install
```

4.2 Scons

```
cd $MYHOME/builds
tar xzvf ../tarfiles/scons-0.98.5.tgz
cd scons-0.98.5
python setup.py install --prefix=$MYHOME/juggler
```

4.3 cppdom

```
cd $MYHOME/builds
tar xzvf ../tarfiles/cppdom-0.7.10.tar.gz
cd cppdom-0.7.10
../../juggler/bin/scons install prefix=$MYHOME/juggler
```

The initial build of `cppdom` had a problem:

```
[demo@swan cppdom-0.7.10]$ ../../juggler/bin/scons install prefix=$MYHOME/juggler
scons: Reading SConscript files ...
NOTE: The build is currently in development. It needs the SVN trunk version of scons-
addons
Building CppDom Version: 0.7.10
Checking for arch [ia32] ...(cached) yes
Valid archs: ['x64', 'ia32']
AttributeError: 'module' object has no attribute 'is_valid_construction_var':
  File "/home/demo/jeremy/juggler/builds/cppdom-0.7.10/SConstruct", line 84:
    opts.AddOption(sca_opts.SeparatorOption("\nPackage Options"))
  File "deps/scons-addons/src/SconsAddons/Options/Options.py", line 586:
    if not SCons.Util.is_valid_construction_var(k):
```

This problem seems to be that `is_valid_construction_var` isn't defined as a method for the `Util` package. `is_valid_construction_var` is defined in `Environment.py`, in `../scons-0.98.5/build/lib/SCons`.

`Util.py` is in `./deps/scons-addons/src/SConsAddons`, and also in `../scons-0.98.5/build/lib/SCons`; these two files aren't the same, and it's not clear which one's being picked up.

As an alternative to using the `cppdom` distribution of `scons-addons`, we tried using the separate distribution. We pick this up by changing the line

```
sys.path.insert(0, os.path.join('deps', 'scons-addons', 'src'))
```

in `SConstruct` to

```
sys.path.insert(0, os.path.join('.', 'scons-addons', 'src'))
```

Now we get

```
[demo@swan cppdom-0.7.10]$ ../../juggler/bin/scons install
scons: Reading SConscript files ...
NOTE: The build is currently in development. It needs the SVN trunk version of scons-
addons
  File "../scons-addons/src/SConsAddons/Options/__init__.py", line 40
    from SConsAddons.Options.Options import (Option, OptionProxy, LocalUpdateOption,
PackageOption,
                                             ^
SyntaxError: invalid syntax
```

Comparing this `__init__.py` with the one in the `deps/scons-addons/src/SconsAddons`:

```
40,45c40,43
< import SConsAddons.Options.Options
< from SConsAddons.Options.Options import Option, LocalUpdateOption, PackageOption, \
<                                     StandardPackageOption, \
<                                     SimpleOption, BoolOption, SeparatorOption, \
<                                     ListOption, EnumOption, Options, OptionError
<
---
> from SConsAddons.Options.Options import (Option, OptionProxy, LocalUpdateOption, \
>                                     PackageOption, StandardPackageOption, \
>                                     SimpleOption, BoolOption, SeparatorOption, \
>                                     ListOption, EnumOption, Options, OptionError)
```

We swapped the `deps/scons-addons/src/SconsAddons` version in and built again, with no problems:

```
[demo@swan cppdom-0.7.10]$ ../../juggler/bin/scons install prefix=$MYHOME/juggler
scons: Reading SConscript files ...
NOTE: The build is currently in development. It needs the SVN trunk version of scons-
addons
Building CppDom Version: 0.7.10
Checking for arch [ia32] ...(cached) yes
Valid archs: ['x64', 'ia32']
Checking for cppunit...searching...
  could not find cppunit-config. Use CppUnitBaseDir to specify it: Ex:
CppUnitBaseDir=/usr/local
CppUnit not available. Skipping it...
cppunit base dir not found
CppUnit not available. Skipping it...
Updating boost
Loading initial settings for boost
  BoostBaseDir specified or cached. [/usr].
  BoostIncludeDir specified or cached. [/usr/include].
  Boost, autofinding toolset... toolset: [gcc]
```

```

boost include path: /usr/include
boost version: 1.32.0
using prefix: /home/demo/jeremy/juggler
types: [['debug', 'optimized'], True]
libtypes: [['shared', 'static'], False]
archs: [['x64', 'ia32'], True]

scons: warning: The env.Copy() method is deprecated; use the env.Clone() method instead.
File "../scons-addons/src/SConsAddons/Variants.py", line 163, in iterate
  Processing combo: type:debug, libtype:['shared', 'static'], arch:x64
  Processing combo: type:debug, libtype:['shared', 'static'], arch:ia32
  Processing combo: type:optimized, libtype:['shared', 'static'], arch:x64
  Processing combo: type:optimized, libtype:['shared', 'static'], arch:ia32
scons: done reading SConscript files.

```

4.4 gmtl

```

cd $MYHOME/builds
tar xzvf ../tarfiles/gmtl-0.5.4.tar.gz
cd gmtl-0.5.4
../juggler/bin/scons install prefix=$MYHOME/juggler

```

The initial build of gmtl had a problem:

```

[demo@swan gmtl-0.5.4]$ ../juggler/bin/scons install prefix=$MYHOME/juggler
scons: Reading SConscript files ...
AttributeError: 'module' object has no attribute 'options':
  File "/home/demo/jeremy/juggler/distrib/gmtl-0.5.4/SConstruct", line 30:
    except AttributeError: has_help_flag = SCons.Script.options.help_msg

```

The options attribute for Scons.Script couldn't be found. As a workaround, we commented out the check, and set the flag to false:

```

#try: has_help_flag = SCons.Script.Main.options.help_msg
#except AttributeError: has_help_flag = SCons.Script.options.help_msg
has_help_flag = False

```

gmtl now builds without error.

4.5 flagpoll

```

cd $MYHOME/builds
tar xzvf ../tarfiles/flagpoll-0.9.1.tar.gz
cd flagpoll-0.9.1
python setup.py install --prefix=$MYHOME/juggler/

```

Flagpoll is needed for VR Juggler (see §4.7, below); attempting to build VR Juggler without it causes autogen.sh to warn about not finding a macro AM_PATH_FLAGPOLL, and configure.pl to exit with these errors:

```

/home/demo/jeremy/builds/vrjuggler-2.2.1-1-src/modules/vapor/configure: line 16103:
syntax error near unexpected token `0.9.1,'
/home/demo/jeremy/builds/vrjuggler-2.2.1-1-src/modules/vapor/configure: line 16103:
`AM_PATH_FLAGPOLL(0.9.1, , '

```

Flagpoll files get installed into various places, including \$MYHOME/juggler/share/aclocal. The VR Juggler INSTALL.txt says that the aclocal tool (which gets called in autogen.sh) has to be told about this location, since it doesn't search this place for .m4 files by default. We do this by

```
% export ACLOCAL_FLAGS="-I /home/demo/jeremy/juggler/share/aclocal/"
```

INSTALL.txt also says to prepare flagpoll by telling it about the non-standard directories it needs to search for .fpc files:

```
% export FLAGPOLL_PATH=$MYHOME/juggler/share/flagpoll
```

The location of flagpoll is passed to VR Juggler via the `--with-flagpoll` option to `configure.pl` (see §4.7, below). In order for flagpoll to find the .fpc files it needs to determine the location of packages, the `--prefix` option has to be passed to `configure.pl` when building VR Juggler as well.

Finally, the flagpoll executable gets installed into `$MYHOME/juggler/bin`; this location must be added to the `PATH` for the build of *oView* to work.

4.6 Doozer

```
cd $MYHOME/builds
tar xzvf ../tarfiles/Doozer-2.1.6.tar.gz
cd Doozer-2.1.6
./configure --prefix=$MYHOME/juggler
make install
```

Doozer is needed for *oView* (see §4.10, below); attempting to build *oView* without it causes flagpoll to complain:

```
ERROR: Package Doozer not found.
```

4.7 VR Juggler

```
cd $MYHOME/builds
unzip ../tarfiles/vrjuggler-2.2.1-1-src.zip
cd vrjuggler-2.2.1-1-src
./autogen.sh
mkdir build.brock
cd build.brock
../configure.pl --with-boost-fs-lib=boost_filesystem-gcc34-mt-1_35 \
  --with-boost-includes=$MYHOME/juggler/include/boost-1_35 \
  --with-boost=$MYHOME/juggler/ \
  --with-gmtl=$MYHOME/juggler/ \
  --with-flagpoll=$MYHOME/juggler/bin/flagpoll \
  --prefix=$MYHOME/juggler
make build install >& install.log
```

Note that the first option to `configure.pl` reflects the contents of `$MYHOME/juggler/lib` – specifically, it picks up the file `$MYHOME/juggler/lib/libboost_filesystem-gcc34-mt-1_35.so`.

In addition, we had to add a VR Juggler patch (r21069) to help with picking up the Boost 1.35 filesystem.

This was downloaded from <http://developer.vrjuggler.org/changeset/21069>.

4.8 ImageMagick

```
cd $MYHOME/builds
tar xzvf ../tarfiles/ImageMagick-6.4.1-10.tar.gz
cd ImageMagick-6.4.1
./configure --prefix=$MYHOME/juggler --with-perl-options=PREFIX=$MYHOME/juggler
```

ImageMagick is needed for the build of *oView*; attempting to build *oView* without it causes the compilation to fail:

```
In file included from oViewDraw.cpp:48:
./oView.h:62:22: Magick++.h: No such file or directory
In file included from oViewDraw.cpp:48:
./oView.h:96: error: `Magick' is not a namespace-name
```

```
./oView.h:96: error: expected namespace-name before ';' token
make: *** [oViewDraw.o] Error 1
```

The second option to `configure.pl` is passed to the installer for the PerlMagick module; here, we tell it to install in `$MYHOME/juggler`, instead of the default location (usually a system directory such as `/usr/lib` which we don't have write permission for).

4.9 glf

```
cd $MYHOME/builds
tar xzvf ../tarfiles/glf_1.4-1.tar.gz
cd glf-1.4
make -f Makefile.linux
```

glf is needed for the build of *oView*, as it links with `glf.o`. This could be copied into the *oView* directory; alternatively, the *oView* Makefile could be edited to reflect its location (which is what we did – see §4.10, below).

4.10 oView

```
cd $MYHOME/builds
tar xzvf ../tarfiles/oView.tar.gz
cd oView
make install
```

The instructions in `$MYHOME/builds/oView/INSTALL` include setting values for `VJ_DEPS_DIR` and `DZR_BASE_DIR` in Makefile but in practice this doesn't seem to be necessary. The two changes which were made in Makefile were the deletion of

```
EXTRA_CXXFLAGS+= -DPOWERWALL
```

and the addition of

```
EXTRA_LIBS += ../glf-1.4/glf.o
```

In addition, these environment variables were set before building and running *oView*:

```
% export VJ_BASE_DIR=$MYHOME/builds/vrjuggler-2.2.1-1-src/build.brock/instlinks
% export VJ_DATA_DIR=$VJ_BASE_DIR/share/vrjuggler-2.2
% export VJ_CFG_PATH=$MYHOME/builds/oView/VRJCONFIG/
```

The `VJ_CFG_PATH` variable is the path of locations that VR Juggler applications (like *oView*) searches for configuration files (see §5.2.3, below). Finally, before running *oView*, the location `$MYHOME/juggler/lib` must be added to the `LD_LIBRARY_PATH` so that it can load the appropriate dynamic link libraries.

5 Running oView

This section describes how the *oView* application is run. We give a brief description of the available documentation for the program, followed by a discussion of its input, and the way in which the application is run on the OeRC cluster.

5.1 oView documentation

Documentation for the application is available at `$MYHOME/builds/oView/README`. In addition, typing:

```
% ./oView --help
```

at the command line produces a brief description of the available command line arguments:

```
---- oView Help ----
```

```
Usage: ./oView vrj_configfile[s] [flags]
```

Available options

```
-c xx | --case xx   Load dataset number xx from the index file oView.input
-h | --histology   Enabling of histology slices from (hardcoded) location
-a | --animate     Use the file logmats.ox to produce a looping animation
-l | --load        Use the file logmats.ox to load checkpoints
-L | --labels      Use the file heart.lab (or heartthumb.lab) to load labels
-nl | --no-lod     Disable the use of a lower Level of Detail when moving
-r | --record      Save checkpoints into the file logmats.ox
-t | --thumbnail   Start viewer in thumbnail mode; no translation available
```

Auxiliary input files mentioned in these options include `logmats.ox`, which contains a list of camera viewpoints, and `heart.lab`, which contains labels and 3D locations for them. The format of `logmats.ox` can be determined by inspecting the source file `$MYHOME/builds/oView/oViewDraw.cpp`, but it is most easily generated as output from *oView* itself (using the `--record` flag – see above).

5.2 Input to oView

This section discusses the origin and structure of the various input files that *oView* uses.

5.2.1 The oView.input file

The `oView.input` file contains details of which 3D dataset files are to be read. This is self-documented; for example, its current contents are:

```
DATA/heart1_          4 576   816   240   DATA/heart1_LOD_          1
DATA/MRI_split/heart1_ 4 576  1056  240   DATA/MRI_split/heart1_LOD_ 1

# 1 File
# 2 digits in file index
# 3 start
# 4 stop
# 5 step
# 6 LOD file to use
# 7 Number of surfaces
# 8
# 9
# 10
# 11
# 12
```

oView has the facility to read several dataset files and assemble the scene from them; the separate files (usually) correspond to different physical regions in 3D (for example, different pieces of the heart). The names of the files to be read are controlled by the `start`, `stop` and `step` variables; these are used in a loop to make indices that are incorporated into filenames (the loop index begins at `start`, and is increased by `step` after each iteration of the loop, which terminates when the index is greater than or equal to `stop`).

Which set of files gets read is controlled by the `--case` flag on the command line (see §5.1, above). For example, setting this flag to 1 causes *oView* to read the data files named `heart1_0576.pdw` and `heart1_LOD_0576.pdw` from `$MYHOME/builds/oView/DATA`, while setting it to 2 causes the files `heart1_0576.pdw`, `heart1_0816.pdw`, `heart1_1056.pdw`, `heart1_1296.pdw`, `heart1_1536.pdw`, `heart1_1776.pdw`, `heart1_LOD_0576.pdw`, `heart1_LOD_0816.pdw`, `heart1_LOD_1056.pdw`, `heart1_LOD_1296.pdw`, `heart1_LOD_1536.pdw`, and `heart1_LOD_1776.pdw` to be read from `$MYHOME/builds/oView/DATA/MRI_split`. The default value for this flag is 1.

5.2.2 3D dataset file

Chris Goodyer's application calculates isosurfaces through the 3D MRI data using routines from the VTK library, and outputs them as triangle strips to be read and displayed by *oView*. As its name implies, a triangle strip is a series of connected triangles that share vertices. More specifically, following the definition of the first triangle using three vertices, each subsequent triangle can be defined by only one additional vertex, sharing the last two vertices defined for the previous triangle. Triangle strips are an efficient way to describe surfaces in computer graphics, and their display is optimised on most graphics cards.

The basic format of the triangle strip file is that generated by VTK's `PolyDataWrite` routine, with a file extension of `.pdw`. The isosurfacing application has been developed using VTK 5.0.1, in which the version number of the `pdw` format is 3.0. In addition to this, subsequent revisions of the format have been numbered 4.0, 4.1 and 4.15. The most important things to note about 4.x datasets is that their endianness has been switched from that generated by VTK, and that the spatial organization of the triangle strips has been optimised. By moving strips that are physically close together to the same part of the file, the data volume has been divided up into smaller cubes, each of which can be either quickly rendered using a display list, or discarded if outside the viewing frustum. This optimization has been performed as a pre-visualization step using the *splitter* program (the source for which `-splitter.cpp` is part of Chris's *oView* distribution, and can be found in `$MYHOME/builds/oView`).

The *oView* package can use two versions of the same scene, generated at different levels of detail; by default, it will load the low level-of-detail version when the viewpoint is changing, and switch to the high level-of-detail version when the viewpoint is at rest. This option, which improves performance but uses more memory, can be disabled using the `--no-lod` flag on the command line (see §5.1, above). The two example 3D dataset files `heart1_0576.pdw` and `heart1_LOD_0576.pdw` in `$MYHOME/builds/oView/DATA` have been created using different degrees of decimation on the triangle strip.

5.2.3 VR Juggler configuration file

The other primary input to *oView* is the VR Juggler configuration file. This file contains details of the hardware configuration (i.e. cluster nodes, output devices such as screens and head-mounted displays, input devices such as keyboards, mice, wands and gloves, and the way in which they're all connected together) that the VR Juggler application is to use. As noted above (§3.2), a particular feature of VR Juggler is that it allows the same application to be run on a variety of hardware without modification or recompilation: instead, the only thing that needs to be changed is the configuration file.

The configuration file is written in XML, and can be edited by hand; alternatively the *VRJConfig* application provides a convenient graphical user interface to viewing and editing the file. This hasn't been built on `swan` because it uses Java, which isn't installed on that machine; however, it's available within, for example, the Windows distribution of VR Juggler¹⁴. Several example configuration files can be found in `$VJ_DATA_DIR/data/configFiles/`, and in `$MYHOME/builds/oView/VRJCONFIG/`. The former is the location of `standalone.jconf`, which can be used to run a VR Juggler application on a single node of the cluster, while the latter contains `tiles.jconf`, which is the configuration file for the OeRC cluster (see §5.3, below).

More details of the format of the configuration file can be found in the VR Juggler documentation¹⁵.

5.3 Running oView on the OeRC cluster

The basic command for running *oView* on a single node of the cluster is simply

```
% ./oView standalone.jconf
```

¹⁴ <http://www.vrjuggler.org/download.php#vrjuggler22>

¹⁵ http://www.infiscape.com/documentation/vrjuggler-config/2.2/configuring_vr_juggler/

which uses the configuration file for a single machine. To distribute the visualization across all the nodes of the cluster, *oView* must be run on all nodes simultaneously. In addition, the name of the configuration file must be replaced with `tiles.jconf`; this passes details of the cluster to VR Juggler, which handles the communication and synchronization between the nodes and displays. One way of doing this is to run the following shell script (which is contained in the file `$MYHOME/builds/oView/jrun.sh`):

```
#!/bin/sh

HOSTS="tile-0-0 tile-0-1 tile-1-0 tile-1-1 tile-2-0 tile-2-1 tile-3-0 tile-3-1"

for i in $HOSTS
do ssh $i "export __GL_SYNC_TO_VBLANK=1;export DISPLAY=:0.0;cd $PWD;export
VJ_CFG_PATH=$HOME/jeremy/builds/oView/VRJCONFIG;./oView tiles.jconf $" >& $i.log&
done

export __GL_SYNC_TO_VBLANK=1;export DISPLAY=:0.0;cd $PWD;export
VJ_CFG_PATH=$HOME/jeremy/builds/oView/VRJCONFIG;./oView tiles.jconf $* >& swan.log&
```

Here, the `HOSTS` variable contains the names of all the nodes in the cluster (apart from `swan`, the headnode: a separate command is used to run on this node to allow for special features such as interaction – see §6, below). This script is run at the command prompt on the headnode, and jobs are spawned onto the other nodes. Any command line options are passed to the *oView* command within the individual jobs – for example, this command runs *oView* in animation mode, loading the first set of input file descriptions from `oView.input`, across all nodes:

```
% ./jrun.sh --case 1 --animate
```

Currently, the easiest way to terminate *oView* is to kill the jobs on all the nodes; the shell script `$MYHOME/builds/oView/jstop.sh` has been provided to do this.

5.3.1 Modifying the displays

The VR Juggler configuration file allows us to specify that the full extent of each screen is used for the display window. The X window manager surrounds each window (including full-screen windows) with a border, which can be distracting; accordingly, we turn this behaviour off by not starting the window manager. Specifically, this was done by commenting out the line

```
exec /opt/viz/bin/fvwm
```

in the file `/opt/viz/etc/.Xclients` on each node (except for the headnode).

6 Current status and further work

Currently, *oView* is running on the OeRC video wall, producing static views of various 3D heart datasets. It also produces animated sequences as the camera is moved along a pre-defined path, constructed by smoothly interpolating between a series of checkpoints.

The next step would be to incorporate interaction into the scene, so that the orientation and position of the camera is controlled by the user (for example using a pointing device such as a 3D mouse). Since VR Juggler already supports interaction of this kind (which has already been used in the Leeds installation of *oView*), its incorporation would be a matter of modifying the configuration file to add the appropriate input device; some initial work has already been done in this direction.

The other type of further work would be the extension to other types of visualization. As noted above (§5.2.2), the *oView* application is restricted to the display of triangle strips (as produced, for example, by isosurfacing 3D datasets). Other triangle strip data could immediately be displayed by *oView* – for example, originating with the isosurfacing of datasets from – say – astrophysics, or chemistry, or

mathematics. Other 3D scenes (e.g. architectural models) could also be displayed, as long as they were composed from triangle strips. Finally, the experience gained with the porting of *oView* to the video wall could be applied to other OpenGL applications that use different visualization techniques such as volume rendering or terrain visualization.

Acknowledgements

We thank Chris Goodyer for helpful discussions, and for making his *oView* code available for porting.