

Using VTK and the OpenGL Graphics Libraries on HPCx

Jeremy Nowell
EPCC
The University of Edinburgh
Edinburgh EH9 3JZ
Scotland, UK

April 29, 2005

Abstract

Some of the graphics libraries and visualisation tools available on HPCx are described. Attention is given to the specifics of using the tools on HPCx rather than details of the tools themselves.

Contents

1	Introduction	2
2	Graphics Libraries	2
2.1	OpenGL	2
2.1.1	OpenGL Utility Libraries	2
2.1.2	OpenGL on HPCx	2
2.1.3	Example Programmes	3
2.2	Mesa	4
2.2.1	Use on HPCx	5
2.2.2	Examples	5
2.2.3	Offscreen Rendering	6
2.2.4	Mangled Mesa	6
3	The Visualization Toolkit	7
3.1	Object Models	7
3.1.1	Visualisation Model	7
3.1.2	Graphics Model	7
3.2	Usage on HPCx	8
3.3	Benchmarks	8
4	Summary	9
5	Further Work	9

1 Introduction

This report describes some of the graphics and visualisation tools available on HPCx. Documentation for all these tools is readily available elsewhere, so the report concentrates more on the details of using them on HPCx.

Section 2 describes the graphics libraries available on HPCx, whilst Section 3 covers the Visualization Toolkit (VTK).

2 Graphics Libraries

2.1 OpenGL

The most widely used graphics library is OpenGL [1]. OpenGL is a software interface to graphics hardware. It can be used for developing both two and three dimensional graphical applications across a wide range of computing platforms, and allows for the possibility that the computer on which the graphics is to be displayed is different to that on which the application runs. It has an API that provides access to a large number of graphics functions including rendering, texture mapping and other visualisation functions. The standardisation of OpenGL ensures that all applications will produce consistent results on any OpenGL compliant platform regardless of the operating or windowing system in use. To display an image on screen models are built from a small set of geometric primitives - points, lines and polygons, and then rendered into a form suitable for display by the graphics hardware. The OpenGL standard specifies the graphics processing pipeline for the geometric primitives and image data, however, the implementation of the pipeline is left up to the provider. A particular platform may therefore make use of dedicated graphics hardware for different parts of the rendering pipeline or alternatively may have an implementation of the whole pipeline purely in software. Standard OpenGL bindings exist for C, C++, Fortran, Ada and Java, although the C bindings are the most widely used.

2.1.1 OpenGL Utility Libraries

Several additional libraries have been developed to aid application development with OpenGL. The OpenGL Utility Toolkit (GLUT) [2] provides simple APIs for writing windowing applications with OpenGL in a portable fashion. A programme written using GLUT should be easily portable between Windows PCs, Linux/UNIX workstations and machines running Mac OS. The OpenGL Utility library (GLU) [3] provides several functions that use lower level OpenGL functions to perform common tasks such as setting up viewpoints, projections and rendering surfaces.

2.1.2 OpenGL on HPCx

IBM's implementation of OpenGL is available on HPCx in the standard system `/usr/lib` and `/usr/include/GL` directories. All user access to HPCx is via a remote ssh connection so the use of graphics applications requires an X Windows server to be present on the desktop, whether this be a UNIX/Linux workstation or by using an application such as Exceed [4] on a PC running Windows. The ssh application used to access HPCx should be configured to enable the forwarding of X11 connections. Furthermore, the use of OpenGL applications on X Windows requires the server to support the GLX [5] extensions. GLX is the interface layer between OpenGL and X windows, and also provides an API for X-based OpenGL programmes. The presence of the GLX extensions can be checked by the use of the `xdpinfo` command in

a terminal window. Exceed users need to install an additional package from Hummingbird, Exceed3D, which contains these extensions. The relevant part of the output of `xdpyinfo` is shown below for installations of Exceed with and without the GLX extensions.

```
jeremy@l1f51$ xdpyinfo
name of display:    11f51:21.0
version number:    11.0
vendor string:     Hummingbird Ltd.
<...>
number of extensions:    14
    BIG-REQUESTS
    DOUBLE-BUFFER
    Extended-Visual-Information
    GLX
    HCL-DOS-Access
    HCLMISC
    HCLxperf
    LBX
    SECURITY
    SHAPE
    TOG-CUP
    XC-APPGROUP
    XC-MISC
    XInputExtension
```

```
jeremy@l1f51$ xdpyinfo
name of display:    11f51:21.0
version number:    11.0
vendor string:     Hummingbird Ltd.
<...>
number of extensions:    13
    BIG-REQUESTS
    DOUBLE-BUFFER
    Extended-Visual-Information
    HCL-DOS-Access
    HCLMISC
    HCLxperf
    LBX
    SECURITY
    SHAPE
    TOG-CUP
    XC-APPGROUP
    XC-MISC
    XInputExtension
```

2.1.3 Example Programmes

All the examples used in this report may be found in the directory

```
/usr/local/packages/ mesa/examples
```

on HPCx. A simple “hello world” programme to test OpenGL is located in `hello/hello_opengl.exe` in this directory. Running this programme will show whether or not your X server is able to display OpenGL applications, and should show an image of a white square on a black background as shown in Figure 1. To exit the window press the `ESC` key. The source code (`hello.c`) shows how to create the white square using OpenGL and draw it in a window created using GLUT. A more complicated example may be found in `gears/gears_opengl.exe`. This should show an animated version of the image shown in Figure 2. Every five seconds the speed of the animation in frames per second will be printed. Further examples taken from the OpenGL Programming Guide [6] and available on the OpenGL website [1] may be found in the `examples/redbook` subdirectory.

Useful sources of information on OpenGL may be found on the OpenGL website [1], the OpenGL Programming Guide [6] and the OpenGL Reference Guide [7]. Earlier editions of these guides are available online as links from [8] and [9].

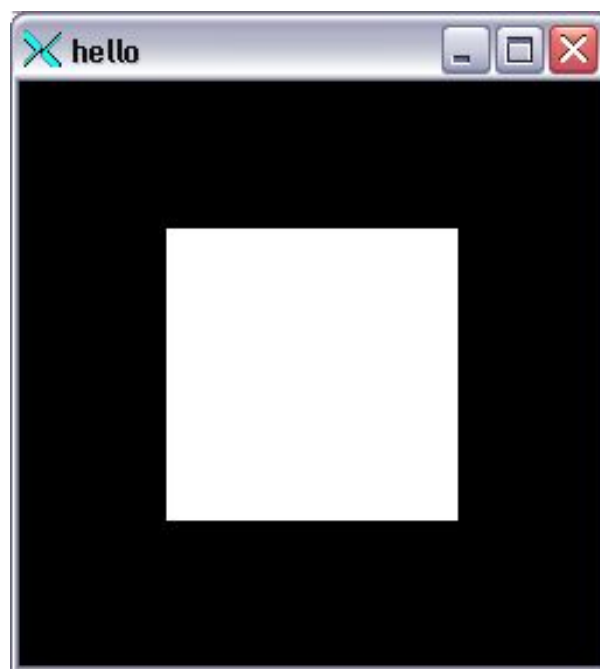


Figure 1: The output of a simple OpenGL “hello world” program.

2.2 Mesa

The Mesa 3D graphics library [10] is an open source implementation of the OpenGL specification, providing an API that is very similar to that of OpenGL. It is useful for situations in which no OpenGL implementation is available, whether that be on the server or the client. It is also useful for situations in which the X windows server does not support the GLX extensions, for example when using the non-3D version of Exceed on a PC. Mesa can also prove useful when off-screen rendering of scenes is required. The current version of Mesa, 6.2, provides an implementation of version 1.5 of the OpenGL specification, along with selected OpenGL extensions.

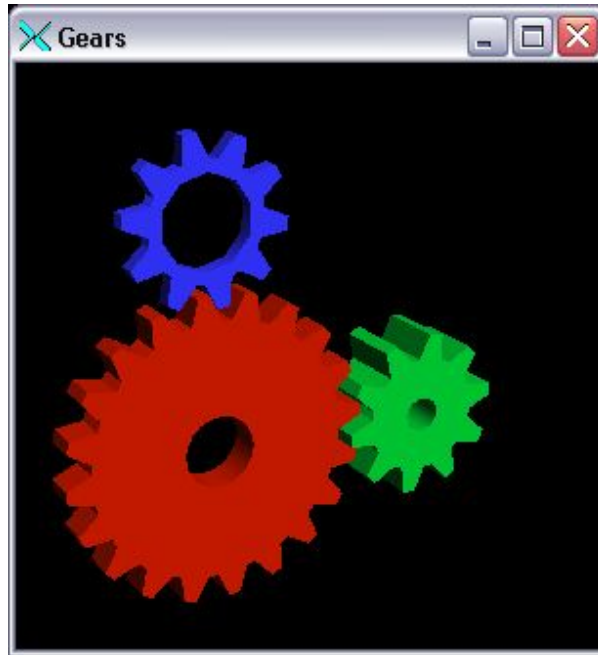


Figure 2: The output of the OpenGL animated gears program.

2.2.1 Use on HPCx

The latest version of Mesa (6.2) is installed on HPCx in the following directory:

```
/usr/local/packages/mesa/mesa6.2
```

Libraries are located in the `lib` subdirectory and header files in `include/GL`. The header files for Mesa have the same name and the same definitions as those for OpenGL so to compile an application with one or another of the libraries it should just be a case of changing the include path and linking against the correct set of libraries. The Mesa libraries have been named similarly to the corresponding OpenGL ones but with the addition of Mesa in the name to avoid confusion, for example `libMesaGL.a` instead of `libGL.a`.

2.2.2 Examples

The same examples as shown in Section 2.1.3 built using Mesa can be found in the same subdirectories of `/usr/local/packages/mesa/examples/`. Executables compiled using OpenGL are named `appname_opengl.exe` whilst those compiled using Mesa are named `appname_mesa.exe`. The images produced by the two different executables should be identical in both cases. In each subdirectory are two different makefiles `Makefile.OPENGL` and `Makefile.MESA`. These makefiles show the differences in building programmes with the different libraries, namely the changes to the include file search path and the libraries used during linking.

The performance of the two different implementations of the gears application (as shown in Figure 2) for a particular Windows PC running Exceed3D is shown in Table 1. As can be seen the OpenGL implementation is a lot more performant than Mesa. This is a result of the OpenGL implementation using GLX to access the OpenGL support in the graphics card and driver on the PC.

Library	Rendering Speed (Frames Per Second)
OpenGL	85
Mesa	1

Table 1: The Performance of OpenGL and Mesa for the gears application.

2.2.3 Offscreen Rendering

Mesa contains functions for offscreen rendering. These can prove useful for making, for example, individual animation frames at different stages of a simulation that can be later joined together to form a movie. A simple example of offscreen rendering can be found in the `offscreen` directory. When run (using the command `osdemo filename.tga`) a tga format image file will be produced as shown in Figure 3 instead of being displayed on screen.

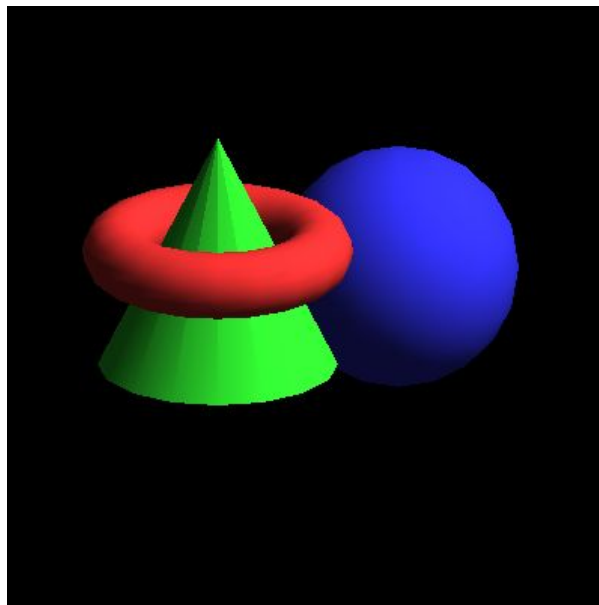


Figure 3: The output of the Mesa offscreen rendering program.

2.2.4 Mangled Mesa

Mangled Mesa provides a way of using both OpenGL and Mesa in the same application. This is useful if one wants to have the high performance of OpenGL for onscreen rendering but still have access to Mesa for offscreen rendering. The “mangled” header files and libraries contain the same symbols as the normal ones prefixed with an `m`. The libraries are named similarly, for example `libMesaMGL.a` and the header files are called, for example, `gl_mangle.h`. To make use of mangled Mesa in an application clearly requires some work to change function names at the appropriate places. In practice its only use is likely to be in building VTK.

3 The Visualization Toolkit

The Visualization Toolkit (VTK) [11] is an open source, freely available software system for 3D computer graphics, image processing, and visualisation produced by Kitware Inc [12]. It has a higher level of abstraction than OpenGL, making it easier to create visualisation and graphics applications. It is written according to an object-oriented philosophy and consists of a C++ class library. Several interpreted interface layers including Tcl/Tk, Java, and Python are available. These layers sit on top of the C++ libraries and simplify the creation of applications in the language of choice. The Tcl/Tk layer is the most developed of these and makes it easy to rapidly create simple visualisations. VTK supports a wide variety of visualisation algorithms including scalar, vector, tensor, texture, and volumetric methods; and advanced modelling techniques such as implicit modelling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation. Many different imaging algorithms have also been directly integrated in VTK to allow them to be mixed with the 3D graphics algorithms and data. Data can be read in several popular visualisation formats. VTK runs on most Unix/Linux platforms, Windows PCs and Mac OSX.

3.1 Object Models

The process that VTK uses to visualise data may be split into two parts, the graphics model and the visualisation model. The visualisation model takes raw information and turns it into a geometric representation. The graphics model takes this representation and turns it into a picture.

3.1.1 Visualisation Model

The VTK visualisation model is based on a data flow idea, as used by many other visualisation systems. Different modules are connected together into a network or pipeline. As the data flows through the network each of the modules performs a specific operation on it. This model allows new algorithms to be quickly applied to the data as required. Within the framework of the VTK object oriented design the visualisation model consists of two type of object, process objects and data objects. The modules are process objects whilst the data that the modules act on are data objects. The process objects can be split into three types, sources, filters and mappers. Source objects initiate the network and generate output datasets. Filters take one or more datasets as input and generate one or more outputs. Mappers take one or more filter outputs and terminate the network. Mappers act as the interface between the visualisation pipeline and the graphics model by transforming the data objects into graphical objects.

3.1.2 Graphics Model

The graphics model is an abstract model for 3D graphics based on the film industry, with influences from current graphical user interface (GUI) windowing systems. There are nine basic objects in the model:

1. Render Master - coordinates device-independent methods and creates rendering windows.
2. Render Window - manages a window on the display device. One or more renderers draw into a render window to generate a scene (i.e. the final image).
3. Renderer - coordinates the rendering of lights, cameras, and actors.

4. Light - illuminates the actors in a scene.
5. Camera - defines the view position, focal point, and other camera characteristics.
6. Actor - an object drawn by a renderer in the scene. Actors are defined in terms of mapper, property, and transform objects.
7. Property - represents the rendered attributes of an actor including object colour, lighting (e.g., specular, ambient, diffuse), texture map, drawing style (e.g., wireframe or shaded), and shading style.
8. Mapper - represents the geometric definition of an actor and maps the object through a lookup table. More than one actor may refer to the same mapper.
9. Transform - an object that consists of a 4x4 transformation matrix and methods to modify the matrix. It specifies the position and orientation of actors, cameras, and lights.

The object model allows these objects to be easily extended. For example actors may be combined in a group to allow property specifications or transformations to be easily applied to all of them at once.

3.2 Usage on HPCx

Version 4.4 of VTK is installed on HPCx in the following directory:

```
/usr/local/packages/vtk/vtk4.4
```

Two versions have been built, one using OpenGL for onscreen rendering and mangled Mesa for offscreen rendering, the other using Mesa for both (installed in `/usr/local/packages/vtk/vtk4.4-mesa`). The latter version can be used with any X server, however, it is *highly* recommended to use the OpenGL version whenever possible for performance reasons, as illustrated in Section 2.2.2. Along with the C++ libraries the Tcl/Tk bindings have also been built.

The examples provided with VTK can be found in `/usr/local/packages/vtk/examples/`. Before using these it is necessary to set the environment variable `VTK_DATA` to the value `/usr/local/packages/vtk/VTKData/`, add `/usr/local/packages/vtk/vtk4.4/bin` to the `$PATH` and `/usr/local/packages/vtk/vtk4.4/lib/vtk` to the `$LIBPATH`. It is recommended to study these examples to gain an understanding of the capabilities of VTK. A simple example to start with can be found in `sphere/sphere.tcl`. To run change to the directory and type `vtk sphere.tcl`. A window should pop up showing an image like that in Figure 4. Clicking in the window with the mouse has a different effect on the camera depending upon which button is used. The left button rotates the viewpoint around the sphere, the middle button translates the viewpoint in the direction of the mouse relative to the centre of the picture and the right button moves the viewpoint in or out of the scene depending upon whether it is clicked in either the top half or the bottom half of the window. Pressing `⏏` quits VTK.

3.3 Benchmarks

Example benchmarks obtained using Sébastien Barré's simple sphere benchmark [13] are shown in Table 2. This program may be found in `sphere/sphere-bench.tcl`, and produces an output as shown in Figure 5. Again these results illustrate the big advantages to be gained using the OpenGL implementation for on screen rendering rather than Mesa only. It is interesting to note also the range of results is bigger for the OpenGL case than for Mesa, illustrating

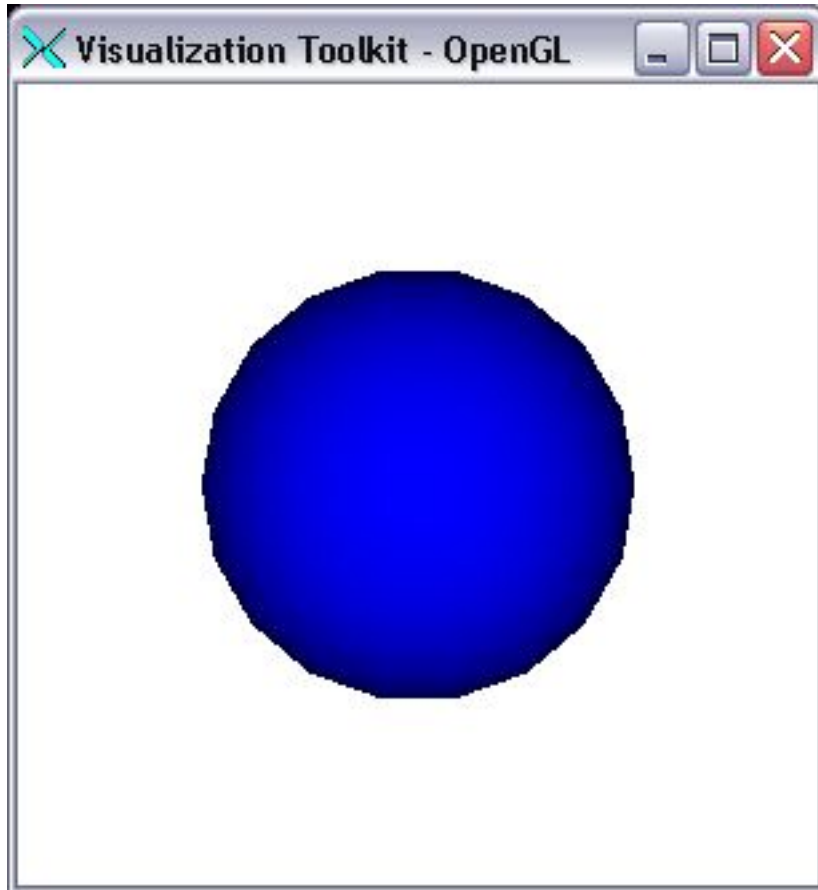


Figure 4: The output the simple VTK sphere example.

the fact that the OpenGL support on the PC is able to speed up certain cases more than others, whereas the Mesa performance is governed by the CPU speed.

4 Summary

The OpenGL and Mesa graphics libraries are installed on HPCx and may be used for producing graphical applications. OpenGL is recommended for any application that displays results on screen while Mesa provides an easy way of rendering images to files, for example for use in batch jobs, as well as acting as a back up for when OpenGL is unavailable. The Visualization Toolkit, VTK, has been installed and may be used for producing many different types of data visualisations.

5 Further Work

Parallel mode VTK using MPI is currently under investigation to try and take advantage of the capabilities of HPCx. The Paraview application [14] is being looked at for use on HPCx. This is a full application built on top of VTK for use, in particular, for visualising large and complicated data sets, with the possibility of use in a parallel mode.

Benchmark	OpenGL and Exceed3D	Mesa and Exceed 3D
	Performance in kpolys/s	
Normal	777.2	63.5
Stripper	796.6	63.2
Small	658.9	58.4
Transparency	981.4	64.7
Wireframe	852.2	44.4
Texture	118.4	60.5
Texture and Transparency	118.0	57.7

Table 2: The results of the VTK sphere benchmark program for a sphere of resolution 256×256 and window size 400×400 .

References

- [1] <http://www.opengl.org/>
- [2] <http://www.opengl.org/documentation/specs/glut/glut-3.spec.pdf>
- [3] http://www.opengl.org/documentation/specs/glu/glu1_3.pdf
- [4] <http://connectivity.hummingbird.com/products/nc/exceed/index.html>
- [5] <http://www.opengl.org/documentation/specs/glx/glx1.3.pdf>
- [6] OpenGL Architecture Review Board *et al.*, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.4, Fourth Edition (Paperback)*, Addison-Wesley Professional, 2003, ISBN 0321173481.
- [7] OpenGL Architecture Review Board, Dave Shreiner, *OpenGL Reference Manual : The Official Reference Document to OpenGL, Version 1.4 (4th Edition) (Paperback)*, Addison-Wesley Professional, 2004, ISBN 032117383X.
- [8] http://www.opengl.org/documentation/red_book_1.0/
- [9] http://www.opengl.org/documentation/blue_book_1.0/
- [10] <http://www.mesa3d.org/>
- [11] <http://www.vtk.org/>
- [12] <http://www.kitware.com/>
- [13] <http://www.barre.nom.fr/vtk/bench.html>
- [14] <http://www.paraview.org/>

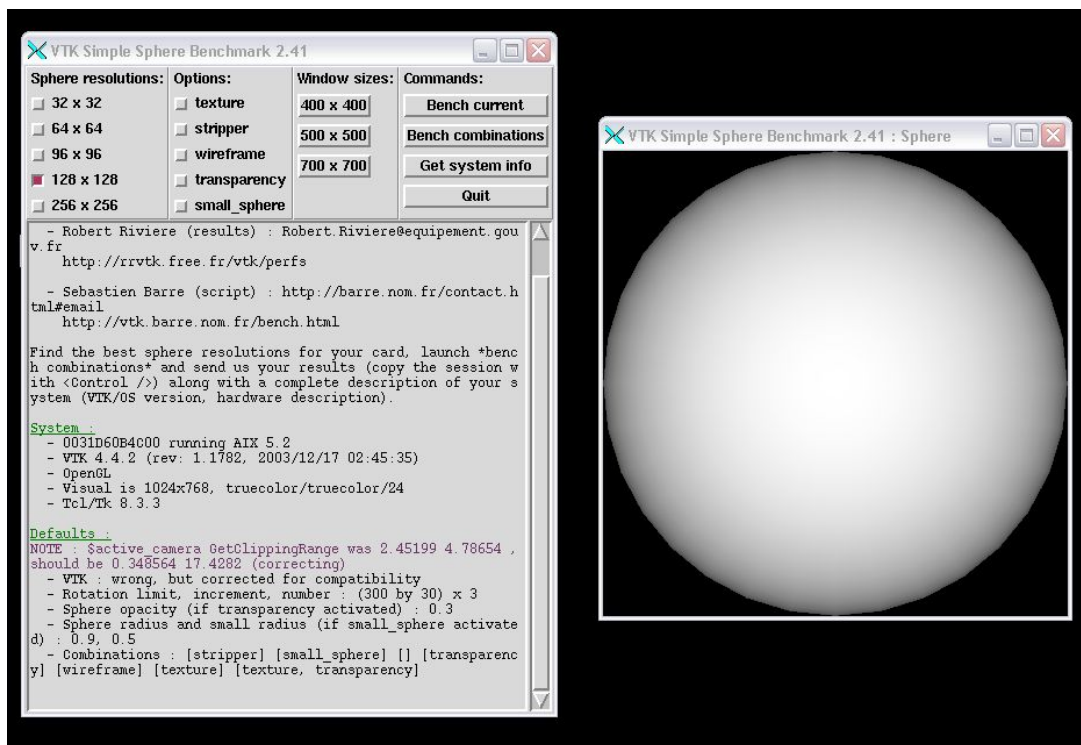


Figure 5: The VTK sphere benchmark program.