Implementing Stereo Vision of GPU-Accelerated Scientific Simulations using Commodity Hardware

T.S. Lyes and K.A. Hawick
Computer Science, Institute for Information and Mathematical Sciences,
Massey University, North Shore 102-904, Auckland, New Zealand
email: { t.s.lyes, k.a.hawick }@massey.ac.nz
Tel: +64 9 414 0800 Fax: +64 9 441 8181

ABSTRACT
Stereo vision technology is becoming more and more commonplace in the movie and gaming industries. It has applications in many other fields as well, one of these is viewing scientific data. We develop a stereo vision system using commodity priced hardware and portable graphics software. Hardware and software details are described, as well as some resulting visualisations and performance issues. C++ and OpenGL are employed to create the stereo visualisation, using Nvidia 3D glasses and a professional GPU graphics card and driver. Key code fragments are presented, and we discuss some of the difficulties in setting up the stereo vision for scientific use. We also present some ideas for future development of scientific visualisation of voxel data in stereo.

KEY WORDS
stereo vision; OpenGL; Nvidia; Quadro; quad-buffering; simulation; interaction.

Figure 1: The authors using the stereo visualisation system with a test data set.

1 Introduction

Volume rendering[1, 2, 3, 4, 5] of scientific data sets is a challenging problem, particularly for interactive simulation. It is computationally expensive but very valuable to be able to interact directly with a simulation program to steer it through a non-trivial parameter space but also to detect and analyse subtle spatial features that emerge - particularly in complex systems simulations.

A number of sophisticated visualisation algorithms and techniques[6, 7] involving cut-away surface identification[8] and shading[9, 10] have been developed bit it is also possible to support various user-interactive ways of navigating through a solid model. Expensive state-of-the art interactive environments have now been feasible for immersive visualisation for some years[11] , but only very recently has it been economically feasible to build a stereo visualisation system from commodity priced desktop components and portable graphics software.

A simple program – known as "cubes" [12] – was developed in OpenGL[13] and C++ and allows a user to load in two- or three dimensional hyperbrick files and to render them in a number of simple ways, with options to transform, shift, rotate the data set, cut-away sections of it to see inside, selectively render only some of the voxel values present, and to produce various output graphics formats including raw portable pixel maps as well as input files suitable for use in more sophisticated rendering tools such as vtk[14] or povray[15].

This present paper describes how we added stereoscopic 3d support to the 'Cubes' program to further enhance visualisation of the hyperbrick data sets. Stereoscopic vision is already becoming prominent in the video gaming and movie industries, however it has also been applied in areas such as vehicle detection [16], medical practices [17] [18], and manufacturing [19], and it would be interesting if the technique could be used to better visualise and understand scientific data sets. For example, a scientist could be "inside" the data set adjusting the parameters, gaining insights immediately on what is happening as the data changes.

While only fairly recently (the last 5 years or so) gaining mainstream success in the arts[20], and in the video gaming and movie industries, the stereoscopic vision problem has been around for a far greater amount of time. In 1838, Charles Wheatstone [21] realized that each eye viewed an object from slightly different positions; the images projected onto each eye differ in the horizontal position, thus giving the illusion of depth (this is known as horizontal, retinal or binocular disparity). Da Vinci[22] also recognized this phenomenon and stated this as a reason why painters would never be able to realistically portray depth on a single canvas. More recently, stereoscopic vision was used in conjunction with photography to produce stereograms (stereo images seen through a stereoscope), and later "autostereograms" such as the popular Magic Eye pictures.

The original 'Cubes' was written in the C/C++ programming language using the OpenGL and GLUT libraries, and it therefore made sense to code the stereoscopic 3d programming in the same language using the same graphics API. OpenGL has built-in stereo functionality which can be used well if the required hardware can be found. This paper will firstly describe the sort of voxel data sets that we are attempting to visualise (Section 2) and in Section 3 we describe the hardware and software necessary to view the program in stereo, as well as giving some key code fragments. In Section 4 we present some results in the form of screenshots and in Section 5 discuss the perceived performance. Finally we offer some conclusions and areas for further research in Section 6. work in the

2 Volume Datasets

For simplicity we employ an easily coded and portable file format for passing voxel data between simulation programs and our stereo visualisation system. The hyperbrick file format[12] – with file ending ".hbrk" – was inspired by the incredibly useful portable pixmap format family (often known as "NetPBM") designed by Poskanzer and developed by Henderson[23]. The ppm and pbm formats have been available to programmers for over two decades and their value is largely due to their simplicity. One can off the top of one's head code up C/C++/Java to generate, or read and write these formats. The "H1" hyperbrick format is a generalisation of the pgm 2D greymap image file format, for the case of 3D (and in principle even higher dimensional) data.

Figure 2 shows the .hbrk file format, consisting of a two character textual header "H1" followed by a newline and an optional series of comment lines starting with a hash character. The subsequent integer – in this case a

```
H1
# a comment or header line
1
3 64 64 32
hyper-raster-of-raw-unsigned-chars
```

Figure 2: The .hbrk hyperbrick file format for a 3-d data set of unsigned chars with spatial extent $x = 64 \times y = 64 \times z = 32$, where the x-coordinate changes fastest, and z slowest.

size of "1" denotes the number of bytes in each payload entity. If one wanted voxels to be allowed to take on 2^24 different levels - like portable pixmap pixels, then one could use a size of "3" to denote three bytes per voxel. The next line gives the dimensionality d of the hyperbrick – usually $d = 3$ for examples discussed in this document, followed by exactly d integer edge lengths in order of increasing significance – so in the example shown $L_x = 64, L_y = 64, L_z = 32$. This line is terminated by a newline and the remainder of the ".hbrk" file is a set of binary characters in the "hyper-raster" order implied by the dimensionality and lengths. So in the example thei_x index would move fastest, the i_y next fastest and so forth.

3 Stereo Rendering

In order to run a graphics simulation in stereoscopic 3d in OpenGL the computer needs to be set up using the right hardware - specifically a graphics card able to support OpenGL Stereo. Standard consumer graphics cards (such as Nvidia GeForce) do not support this. Nvidia Quadro cards are professional versions of the Nvidia GeForce cards and contain additional support for various professional software packages such as Maya and 3ds Max. It is puzzling why open source software such as OpenGL Stereo are included in this package and do not just come standard in all graphics cards. Hopefully this will change in the future. For our program, Quadro cards need to also come with quad-buffering support. Quad-buffering allows the OpenGL program to render and swap buffers for each eye (back and front buffers for right and left sides makes four buffers in total). This allows for a smooth picture in each eye and each left or right set of buffers can only be seen through the corresponding eye when seen through the glasses. Unfortunately only a select few Quadro cards support quad-buffering (specifically the Quadro FX range, such as the Quadro FX 5800, FX 3700 and the older FX 380).

The glasses used are the standard Nvidia 3d Vision shutter glasses such as those featured in Figure 3 (left).

Figure 3: NVidia stereo 3D glasses and Infra Red transmitter which synchronizes the glasses' shutter rate with the monitor's frame rate

The glasses lenses are able to flicker on and off very quickly (undetectable by the human eye) and an infrared transmitter Figure 3(Right) is used to synchronize the glasses with the frame rate of the running application. Also needed to view the stereo program smoothly is a monitor with high refresh rate capabilities, as the glasses shuttering each image separately for each eye will effectively halve the frame rate of the program that would be otherwise be running in non-stereo mode. A monitor with a refresh rate of at least 120 hertz (typical monitors display at only 60 hertz) is good enough for a nice, smooth display (we used the Samsung 2233 RZ model). If the monitor does not have the required refresh rate the images will seem choppy and flicker constantly.

Nvidia 3d Vision is currently only compatible with Windows and (more recently) Linux operating systems. While ideally we would have liked to make the program on a Linux machine, the hardware required to run it on a Linux machine was very expensive so we chose to run it on Windows - specifically Windows 7 32 bit, however it can work on all versions of Windows 7, Vista and XP. We used an Nvidia Quadro FX 380 graphics card with driver version 266.45. OpenGL version 3.7 or higher is needed, as OpenGL Stereo requires OpenGL Game Mode to work, which was introduced in the 3.7 update.

Figure 4: The Nvidia Quadro FX 380 graphics card

We employed the Nvidia Quadro FX 380 graphics card (as shown in figure 4). This device is capable of managing two screens although for the purposes of the work reported here we only employed a single 120Hz screen. Nvidia manufacture a range of more powerful Quadro cards, but this was the most affordable one that could drive the stereo glasses system.

The C/C++ and OpenGL code used to render the hyperbricks in stereo is based on the code explained in [24]. Although one might think that rendering an object in stereo might be much more complicated than rendering simply in 2D, it is not so difficult. Instead of rendering the hyperbrick image on a single buffer, the program renders two images on two buffers, each image seen from a slightly different viewpoint. Algorithm 1 shows a generalized method of rendering objects in stereo.

Algorithm 1 the general method of rendering a hyperbrick in stereo

```
if rendering in stereo then
  update stereo camera, normals, viewpoints
  select right buffer
  render hyperbrick
  select left buffer
  render hyperbrick
end if
for all buffer in buffers do
  swap buffer
end for
```

Each buffer (left and right) is only seen by the corresponding lens in the stereo glasses. If one were to shut one eye, a full 2D image of the hypercube would still be seen through the other. Because of the slightly different viewpoints from each eye, an illusion of depth is achieved and thus the rendering in stereo is complete. These different viewpoints are the key to rendering a good stereo image - if they are not correct, the image may appear to have 'shadows' from the other buffer, in general just be an unconvincing 3D image, or even just appear to be rendered in 2D. Note that both left and right buffers are swapped after the hyperbrick has been rendered - this is not to say that the left buffer is swapped with the right buffer and vice versa, rather, the left buffer is swapped with another buffer on the left and the same with the right. Usually this swapping of buffers refers to each set of left and right buffers as back and front, making four buffers altogether: back left, back right, front left and front right, and hence the term 'quad-buffering'. As stated previously, this technique allows the program to render smoothly, in the same way as the double-buffering technique allows non-stereo graphics programs to render smoothly.

So how do we calculate the viewpoints in order to get a convincing 3d stereo projection? As described in [24],

there are two main ways of doing this. The first is what is known as the "toe-in" method, where both camera views are pointed at a single focal point, the second is what is known as the "off axis" method, where the views of each camera are parallel to each other. The "off-axis" method is considered the superior of the two methods as the discomfort levels are lower than the toe-in method due to the fact it does not introduce a vertical parallax [25].

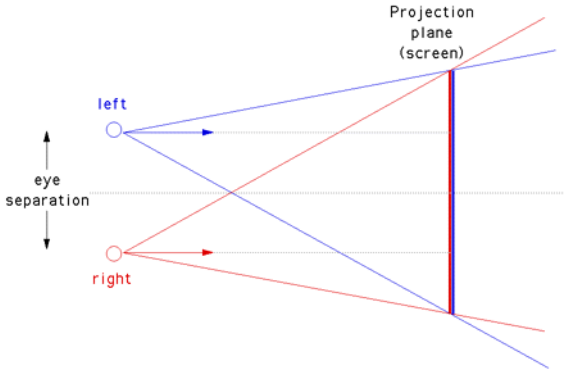

Figure 5: The off-axis stereo projection geometry.

Figure 5 demonstrates the geometry for the "off-axis" method to project the image in stereo 3D, showing the key distances of eye-separation and approximate eye-screen distance.

The code listed in Figure 6 shows how one would typically select the left buffer to be ready for rendering. The function glFrustrum() is used to calculate the correct viewing projection for the left "eye" and the glLookat() function is used to position the "eye" in the correct place and pointing in the correct direction. All variables not declared in the function (they are members of the camera class) are directly dependent on the camera's position, focal length, aperture and eye separation variables, all of which can be altered slightly by the user to get the optimum 3D image. The code for selecting the right buffer is very similar, with only minor changes to the glLookat() function making sure the right "eye" is "mirrored" to the left counterpart.

```
void Camera::LeftBuffer(){
  glDrawBuffer(GL_BACK_LEFT);
  glClear(GL_COLOR_BUFFER_BIT |
          GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  double ratio = scrWidth / scrHeight;
  double wd2 = near * tan(RAD*aperture/2);
  double ndfl = near / focalLength;
  double left = -ratio * wd2 + 0.5
    * eyeSeparation * ndfl;
  double right = ratio * wd2 + 0.5
    * eyeSeparation * ndfl;
  double top = wd2;
  double bottom = -wd2;
  glFrustrum(left, right, bottom, top,
    near, far);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glLookat(-Pos.x-norm.x, -Pos.y-norm.y,
        -Pos.z-norm.z,
        -norm.x, -norm.y, -norm.z,
        Up.x, Up.y, Up.z);
  glRotatef(-Rotation.x, 1.0, 0.0, 0.0);
  glRotatef(Rotation.y, 0.0, 1.0, 0.0);
}
```

Figure 6: OpenGL Code outline for preparing Left "eye" buffer for rendering.

4 Stereo Visualisation Results

The resulting stereo visualisation of the Cubes data sets was successful. It is difficult to show the results properly on paper for obvious reasons; it is not possible to show proper stereo 3d displays on paper. The Cubes program can load in any compatible hyperbrick data set in standard non-stereo mode, and the user is able to toggle stereo mode on and off as needed. The code to display the Cubes data in stereo is completely separate to the Cubes program code itself, and no functionality was sacrificed in order to display it in stereo. Any hyperbrick data set can be rendered in stereo, provided the same data set is rendered on each buffer, as shown in Algorithm 1 .

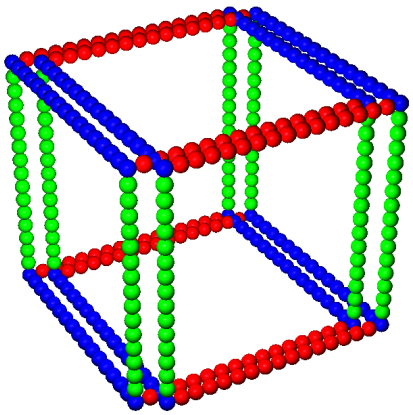

Figure 7: The 'scaffold' predefined test data set as it would appear on screen to the naked eye.

For the stereo display to be most effective it is recommended that the Cubes rendering properties be

changed to a sphere rendering model, with grids and frames surrounding the voxels removed and only non-zero data visible. It is also recommended to have some sort of lighting model switched on for added effect. While these properties produce the best results when viewing the data in stereo, these properties can be changed and the stereo display will still work. These properties are the ones chosen for the following screenshots showing some examples of the program.

Figure 7 shows the pre-defined test data set "Scaffold" as how it might appear on screen to the naked eye; this is not a true representation of how it would look, as the image would appear slightly 'faded' as each buffer flickered on and off. Because at any one time only one buffer is being displayed, it was necessary to render the images of both eyes on only one buffer when taking screenshots, otherwise only the currently displayed buffer would be shown and the image would appear as it would in a non-stereo display mode. The scaffold is simply a hyperbrick of voxels, rendered as Phong-lit[6] spheres, and with all but the edge voxels switched off, and with the x-y-z axes coloured red-green-blue to aid orientation. This data set illustrates well how the stereo display works - the image for each eye is not simply displaced horizontally by a certain amount, but rather the object is viewed at slightly different angles for each eye. This is shown in the scaffold data set by the cube's edges being almost but not quite parallel to each other. The camera's aperture angle can be increased to show this more clearly (this would give the effect of the camera moving closer to the object, or in stereo mode the object would appear further outwards of the screen and closer to the person viewing it).

A typical scientific use of stereoscopic vision is to aid identification of phenomena taking place in a simulation. The example we resent here is discussed further in[26] and involves growing a model of electro-deposition on a physical surface. Figure 9 shows the red particles grown on a green flat surface. The tendrils are hard to analyse and to see what is taking place in the simulated model. Cluster labeling techniques are used to identify separate clusters of connected tendrils, and the "biggest" is visualised separately in Figure 8. The stereo technology allows a much greater sense of the structure and morphology of the simulation that is possible with flat or shaded monocular rendering.

Figure 8 shows the different images for each eye - the left buffer is coloured blue while the right is coloured red. It features the hyperbrick "biggest", referring to the biggest tree generated by an invasion percolation. This data set is particularly effective in being displayed in stereo, as the voxels can be shifted in the x, y and z directions at the user's discretion. This allows parts of the hyperbrick to appear in the extreme front, back, top, bottom or sides which looks great in stereo mode.

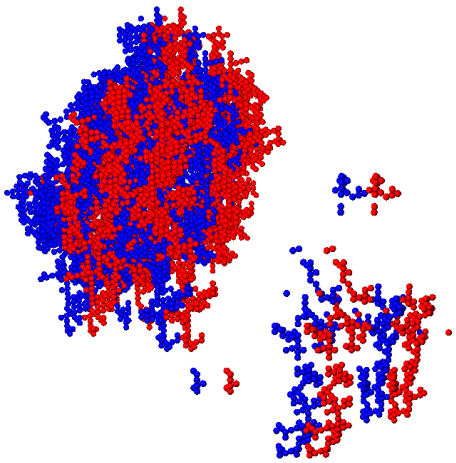

Figure 8: Red and blue colours for each eye for the predefined dataset 'biggest' - which is a cluster pulled out of a larger simulation dataset.

In Figure 8 the small group of voxels to the left and bottom of the image appear extremely close to the viewer, while the rest of the data set is much further away.

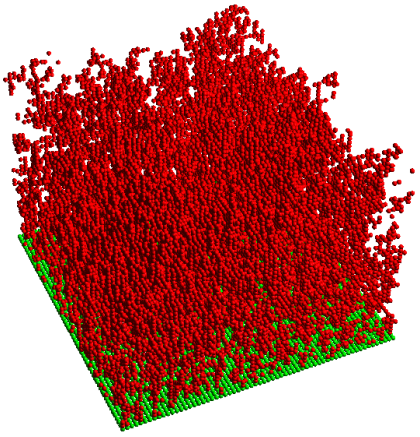

Figure 9: The 'original' hyperbrick dataset from a surface deposition simulation - showing how difficult it is to identify the structure of the individual clusters.

Figure 9 features the "original" hyperbrick dataset, referring to the entire data set generated by the epitaxial growth model (the "biggest" data set in Figure 8 is part of this data set). This data set was a lot more dense and contained a lot more voxels than most other hyperbricks, however the stereo display still worked fine. Because of the heavy clustering it was not practical to display a screenshot of both buffers, as it would be difficult to distinguish which voxels belonged to the left and right buffers. It is important to mention that this data set really challenged the graphics cards capabil-

ities; displaying so many voxels as spheres as well as adding lighting effects and stereo buffering meant the program ran very slowly on the Quadro FX 380.

5 Discussion

In general the program performed well, however as the number of voxels in the hyperbrick is increased (for example a 64 x 64 hyperbrick instead of the standard 16 x 16) the performance levels decline. As mentioned previously, this is most noticeable when rendering the individual voxels as spheres, and even more so when lighting is turned on. This is mainly due to the fact that the graphics card which we used (Quadro FX 380) was a much older model and thus could not keep up with the large workload; however, rendering such large amounts of voxels as spheres and under lights can be taxing on even good graphics cards, and rendering the images twice to obtain the stereo image effect only adds to the workload. It would be good to see if we can find a way to reduce that while still maintaining a clear and convincing looking stereo image.

Another thing to note is that different people seemed to prefer different settings for the stereo camera in order to get the most comfortable image. For this reason the focal length, aperture and eye separation settings for the camera were enabled to be changed on the fly by the user using keyboard support while they were viewing the stereo image.

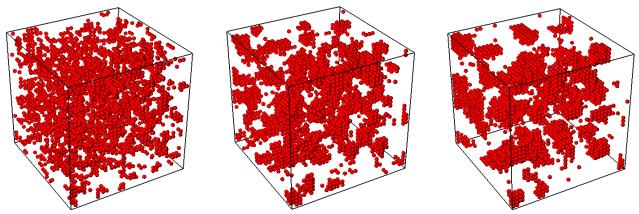

Figure 10: A sequence of voxel sets from simulation of a $32 \times 32 \times 32$ cell Kawasaki spin-exchange model. The particles clump together with time.

It is useful to view stereo-rendered animation sequences. Our system supports this by allowing a set of voxel sets to be cycled through dynamically. Typically a few thousand hyperbrick voxel set can be loaded in memory and used in this way. Figure 10 hows some sample renderings of a sequence of 1024 time-steps from a simulation of the Kawasaki Ising model[27] on a 32^3 lattice. In this model the red and white "spins" are initialised randomly and with time, the system anneals to forms clumping structures, whose precise shape depends upon the model parameters. Stereo visualisation aids identification of the clumping regimes and insights into the structures formed.

6 Conclusions

We have shown how a capability for stereo visualisation of voxel data can be readily implemented with commodity hardware and portable graphics software. Our cubes system offers this capability and users can choose whether to enable the stereo feature or not at run time. The stereo camera's focal lengths needed to be changed depending on the person viewing the program, otherwise the visualisation would leave a 'shadow' effect in each eye. Keyboard support for this was added. Although the stereo vision works well, it is only compatible with specific graphics cards (we used the Nvidia Quadro FX 380). The graphics card needs both OpenGL stereo support and quad-buffering support. OpenGL stereo support comes with all Quadro cards, which are the 'professional' equivalent to the Nvidia GeForce consumer cards. Quad-buffering support allows OpenGL to write to four buffers instead of just two. Only a selection of Quadro cards have this support. Hopefully in the future all or most graphics cards will come built in with both of these functionalities as stereo 3d becomes more prominent in the industry.

For future work it would be desirable to run the simulation on a Linux box. We believe the current generation Nvidia Quadro cards now support Linux. Other interesting interactive commodity-priced hardware such as Microsoft's Kinect device offer some promise for enhanced user-interactivity with a running simulation. Haptic devices such as the Kinect would allow users to manipulate or navigate the voxel data set using hand movements or other methods. It should be possible to take into account the user's head movements, and as the user moves around the image the projection would rotate accordingly, giving the effect of a 'virtual reality' environment. Several studies have been done [28][29][30] using this method. It has been found that coupling this method with stereo vision techniques gives better results than stereo vision on its own, so some experiments on the Cubes program using this method would be very interesting to do.

In summary, it is now economically and technically feasible to expect to have stereo vision and other user interactive devices commonly available on desktops to enhance the interactive experience and computational steering of a simulation experiment.

References

[1] Drebin, R.A., Carpenter, L., Hanrahan, P.: Volume rendering. In: ACM SIGGRAPH Computer Graphics. Volume 22. (1988) 65–74 ISSN:0097-8930.

[2] Wang, W., Sun, H., Wu, E.: Projective volume rendering by excluding occluded voxels. Int. J. Image Graphics **5** (2005) 413–432

[3] Stolte, C., Tang, D., Hanrahan, P.: Multiscale visualization using data cubes. IEEE Transactions on Visualization and Computer Graphics **9** (2003) 176–187

[4] Sherbondy, A., Houston, M., Napel, S.: Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In: Proc. IEEE Visualization 2003. (2003) 171–176

[5] Levoy, M.: Display of surfaces from volume data. IEEE Computer Graphics and Applications **8** (1988) 29–37

[6] Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F.: Computer graphics: principles and practice (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1990)

[7] Hearn, D., Baker, M.P.: Computer Graphics with OpenGL. Third edition edn. Number ISBN 0-13-015390-7. Pearson Prentice Hall (2004)

[8] Hawick, K., Playne, D.: Spectral and real-space solid representations and visualisation of real and complex field equations. Technical Report CSTN-101, Computer Science, Massey University (2009)

[9] Fletcher, P.A., Robertson, P.K.: Interactive shading for surface and volume visualization on graphics workstations. In: VIS '93: Proceedings of the 4th conference on Visualization '93, Washington, DC, USA, IEEE Computer Society (1993) 291–298

[10] Schott, M., Pegoraro, V., Hansen, C., Boulanger, K., Bouatouch, K.: A directional occlusion shading model for interactive direct volume rendering. Computer Graphics Forum **28** (2009) 855–862

[11] Cruz-Neira, C., Sandin, D.J., DeFanti, T.A.: Surround-screen projection-based virtual reality: the design and implementation of the cave. In: Proceedings of the 20th annual conference on Computer graphics and interactive techniques. SIGGRAPH '93, New York, NY, USA, ACM (1993) 135–142

[12] Hawick, K.: 3d visualisation of simulation model voxel hyperbricks and the cubes program. Technical Report CSTN-082, Computer Science, Massey University (2010)

[13] Woo, M., Neider, J., Davis, T., Shreiner, D.: OpenGL Programming Guide: The Official Guide to Learning OpenGL. 3rd edition edn. Addison-Wesley (1999) ISBN:0201604582.

[14] VTK Team: Visualization Toolkit (VTK). See Website: `www.vtk.org` (2010)

[15] Persistence of Vision Pty. Ltd.: Persistence of Vision (TM) Raytracer. Williamstown, Victoria, Australia. (2004)

[16] Bertozzi, M., Broggi, A., Fascioli, A., Nichele, S.: Stereo vision-based vehicle detection. In: Proceedings of the 2000 IEEE Intelligent Vehicles Symposium. (2000)

[17] Maupu, D., Horn, M.H.V., Weeks, S., Bullitt, E.: 3d stereo interactive medical visualization. IEEE Computer Graphics and Applications **25** (2005) 67–71

[18] Webster, R., Haluck, R., Ravenscroft, R., Mohler, B., Crouthamel, E., Frack, T., Terlecki, S., Sheaffer, J.: Elastically deformable 3d organs for haptic surgical simulation. In: Medicine Meets Virtual Reality. IOS Press (2002)

[19] Aguilar, J., Torres, F., Lope, M.: Stereo vision for 3d measurement: accuracy analysis, calibration and industrial applications. Measurement **18** (1996) 193–200

[20] Ahearn, L.: 3D Game Art f/x & Design. Coriolis, Scottsdale, Arizona (2001) ISBN: 1-58880-100-4.

[21] Wheatstone, C.: Contributions to the physiology of vision - part the first. on some remarkable, and hithero unobserved, phenomena of binocular vision. Philosophical Transactions **128** (1838) 371–394 `www.stereoscopy.com/library/wheatstone-paper1838.html`.

[22] Beck, J.: Leonardo's Rule of Painting - an unconventional approach to modern art. 2nd edn. Phaidon Press (1979)

[23] Henderson, B.: Netpbm history. Web Site (2007) Last Visted October 2010.

[24] Bourke, P.: 3d stereo rendering using opengl (and glut). Website PDF (2002)

[25] Bourke, P.: Calculating stereo pairs. `http://paulbourke.net/miscellaneous/stereographics/stereorender` (1999)

[26] Hawick, K.: Morphology and epitaxial growth in a directed diffusion model. In: Proc. International Conference on Modelling, Identification, and Control (MIC), Innsbruck, Austria, IASTED (2011) 1–8

[27] Baillie, C.F., Gupta, R., Hawick, K.A., Pawley, G.S.: Monte Carlo renormalization-group study of the three-dimensional Ising model. Phys. Rev. B **45** (1992) 10438

[28] Ware, C., Franck, G.: Evaluating stereo and motion cues for visualizing information nets in three dimensions. ACM Transactions on Graphics **15** (1996) 121–140

[29] Ware, C., Franck, G.: Viewing a graph in a virtual reality display is three times as good as a 2d diagram. Technical report, Faculty of Computer Science, University of New Brunswick (1994)

[30] Deering, M.: High resolution virtual reality. Computer Graphics **26** (1992) 195–202