Teaching Computational Science Using VPython and Virtual Reality

Stephen Roberts[1], Henry Gardner[2], Shaun Press[3], and Linda Stals[1]

[1] Mathematical Sciences Institute,
Australian National University, Canberra ACT 0200, Australia,
[Stephen.Roberts,Linda.Stals]@anu.edu.au
[2] Department of Computer Science,
Australian National University, Canberra ACT 0200, Australia,
Henry.Gardner@anu.edu.au
[3] Research School of Information Sciences and Engineering,
Australian National University, Canberra ACT 0200, Australia,
Shaun.Press@anu.edu.au

Abstract. The Australian National University has two new complementary computational science programs, the Bachelor of Computational Science and the eScience graduate program. Students from the eScience program have developed sophisticated visualisation projects which have then been used to educate current and prospective undergraduate students in the Bachelor program. In this paper we will discuss the use of VPython combined with a 3D visualisation theatre, the Wedge, to produce hands-on computational science tutorials which we use to motivate computational science. We will briefly describe the use of VPython in our outreach tutorials, in particular a bouncing ball and gas simulation tutorial. The wedge virtual reality environment is also described as is the porting of VPython to the Wedge. Overall we provide a glimpse into our coordinated approach to using high level visualisation and virtual reality in the promotion of computational science.

1 Introduction

In this paper we describe how we use a combination of VPython and the "Wedge" (a virtual reality theatre) to introduce computational science concepts to K-12 students and teachers through to undergraduate and postgraduate students.

The Australian National University (ANU) is home to two novel and complementary education programs in computational science and related technologies, the undergraduate Bachelor in Computational Science (BComptlSci) degree program (and an associated major as part of the Bachelor of Science) [2] and the eScience graduate program [6]. Based on these educational programs we have built an outreach program for year 9-12 students and teachers. This program is exemplified by a hands-on VPython tutorial in which we introduce computational modelling by showing how to produce a simple gas simulation code. This tutorial culminates with students viewing the results of their coding in a

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3039, pp. 1218–1225, 2004.

3D virtual reality theatre, the Wedge, which shows how advanced visualisation can be used to complement scientific discovery (and be fun). The development of these outreach projects and other undergraduate projects have been a driver for some of the educational outcomes in the graduate program. For instance the porting of VPython to the Wedge was completed as a master level project with the eScience program.

In the subsequent sections we will give a more detailed description of these efforts to use sophisticated visualisation tools and environments to educate and enthuse people in the area of computational science.

2 The Undergraduate BComptlSci Program

The ANU Bachelor of Computational Science took in its first students in 2001. The degree is a joint initiative of the Departments of Mathematics and Computer Science, with support from the Department of Physics. The structure of the undergraduate program is representative of the various "churches" which make up computational science: Half of its program is delivered from a combination of courses in mathematics and computer science. The other half of the degree program is associated with a chosen application area such as physics, chemistry, environmental sciences, etc. In this way, students can accumulate enough credits to take an Honours year in a chosen discipline but also to have a healthy mix of programming, numerical mathematics, high-performance scientific computing and related technologies. Check the web site [1] for details of the core courses and structure of the degree program.

In 2003 we had our first three students graduate from the three-year undergraduate program. All three have decided to continue and undertake an extra Honours year in the program. We are averaging about ten students enrolling in the degree each year. The students in our program are developing very good problem-solving skills and they have been employed over the summer break to develop teaching modules and case studies. The number of students enrolled in our specific program is small, but the number of students in our core courses are healthy, supported by students undertaking a wide range of other majors, ranging from the classic areas of physics and chemistry, to earth and environmental sciences, to biology and bioinformatics.

We have found that the concept of "computational science" is quite hard to describe to even our own students. To help attack this problem we run regular seminars in which speakers from a wide range of areas speak on the use of computational science, both in research and in the wider community.

3 The Graduate eScience Program

Starting in 2001, the Computer Science department at ANU also introduced a new graduate program in "eScience" with the objective of delivering "conversion" course-work programs for students with a background in science and engineering [6]. On the one hand the program attempts to provide them with

a range of computing skills which will help them "convert" to being computational scientists in their chosen disciplines. On the other hand, the eScience program recognises that many graduates will aspire to having careers in the IT industry. Because of this, it attempts to provide them with a rigourous introduction to modern programming languages, software engineering, computer graphics, networking, human-computer interaction and high-performance computing optimisation. In this latter case, the computational science orientation of the program serves to act as a "pedagogical hook" to empathise with the backgrounds of the students and to, hopefully, afford better learning. The eScience program has a present enrolment of about 40 and altogether some 20 students will have graduated by the end of 2003. Graduates are finding employment in universities and local industry or are undertaking further studies.

4 Python and VPython in Computational Science Education

The understanding of computational science in the wider community is extremely poorly understood. To help remedy this, we have been involved in numerous outreach activities to raise the profile of computational science to year 9-12 students and teachers.

As part of our computational science program we have been building up a library of teaching modules and tutorials, [3]. These modules use a range of computing environments, but we have found that VPython provides a particularly good environment for people with little previous programming experience. In this section we will describe our experiences using VPython to develop a simulation of a gas. As part of this process we have also found that these teaching modules provide very nice project topics for our graduate eScience students. We describe such a project in the subsequent sections in which VPython has been ported to the Wedge, a virtual reality theatre. But for now let us provide a short introduction to VPython.

Visual Python (VPython) is a 3-D graphics system that is an extension of the Python language. Its main usage has been in the area of demonstration of physical systems in Physics, Chemistry, and Engineering. VPython [8] was written by Dave Scherer under the supervision of Bruce Sherwood and Ruth Chaby. The software is released under the GNU Public License and is available from the VPython website `http://www.vpython.org`.

The VPython module is written in C++ and built to fit seamlessly into the Python Environment. It is imported as a Python module and provides the user with the ability to model 3 dimensional scenes. The strength of VPython is that the programmer need not worry about the mechanics of how to build the display environment. Unlike other graphics environment (Java3D^{TM} for example), the programmer only has to specify the layout of the shapes used in any particular model, and VPython handles the rest. Fig. 1 show the results of our 50 line gas simulation VPython code.

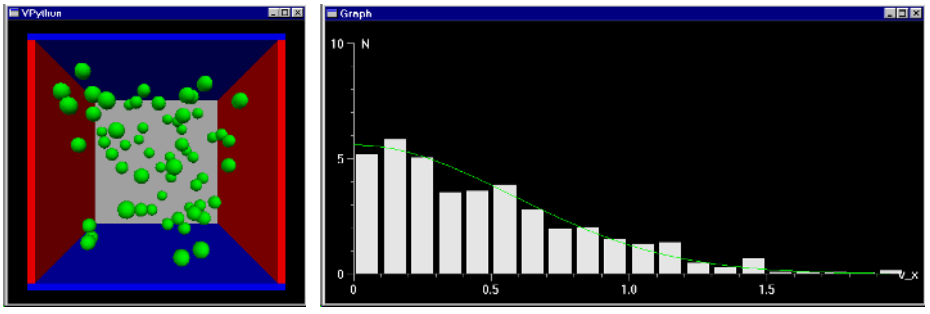

Fig. 1. Screen dump of the gas simulation tutorial showing particles inside a box, together with a graph showing the distribution of speeds of the simulated gas compared to the theoretical distribution

4.1 Bouncing Ball and Gas Simulation Teaching Modules

We have had great success in using the VPython bouncing ball example program as a basis for a tutorial for year 9-12 students and teachers. In the tutorial (see [4]), we lead the participants, who often have little or no programming experience, from creating a ball and wall in VPython, to creating a simulation of a ball bouncing in a box. This is a nice example as it introduces non-trivial programming concepts, such as objects, loops, conditionals, and numerical techniques such as Euler's method, all of which are motivated by the underlying physical problem.

As an extension we also have a gas-simulation tutorial [5], which starts from the bouncing ball code and ends with a simulation of a gas in which the distribution of the velocities of the gas can be observed and compared against the expected Maxwellian distribution. Fig. 1 shows the VPython visualisation of the gas. We find that the second tutorial allows participants to see how sophisticated models can be developed to a stage where useful intuition of the real system can be gained.

We have presented the bouncing ball tutorial to a number of different groups including year 9-10 students, year 11-12 students, Science/Mathematics teachers and Career Advisors. The overall response has been very positive; after attending a tutorial herself one of the teachers arranged for her students to be given the tutorial and we have been asked again, for the second year, to give the tutorial in the National Science Teachers Summer School. From our point of view it is very rewarding to listen to the participants discuss the tasks amongst themselves, ask us numerous questions and occasionally give shouts of joy as some problem is solved. For the most part the participants pick up on the ideas very quickly and keep their enthusiasm throughout the tutorial.

To date, all of the sessions have been held in computer labs at the university campus. In theory the tutorial can be given at the schools and other locations as all of the necessary software is available online. In fact, at the end of some of

the tutorials a CD was handed out containing a copy of VPython and the online tutorial.

We usually have about three staff members assisting with a group of 20-30 participants so any technical problems can be dealt with quickly and the participants can be guided through trouble spots. More importantly, as the tutorial progresses the participants often ask questions about how to extend the process to other contexts and it is at this point that we have the best opportunity to promote the role of computational science in the wider community.

The participants tend to pick up the notation used in VPython very quickly. They will often play around with the size, colour and speed of the balls, which is not specifically discussed in the tutorial. The main aspect of VPython that is not intuitive is the use of indents; if a participant is having problems compiling the code it is often because it was not indented correctly.

Observing the various groups we found that it is best if the participants have some elementary mathematics background. In particular, understanding vectors and the physics behind velocity is important. Many of the Year 9-10 students were not familiar with the 3D coordinate system and had some problems understanding the notation used in VPython. With some guidance they were able to complete the task, but more through a trial and error type of approach.

We have also found that it is important to stop the participants from jumping straight to the end of the instructions and searching for the answer. Many tend to take this approach rather than reading the directions given, so in the beginning it is necessary to "encourage" the participants to go through the tutorial step-by-step. Once they realise that they will get to build something in the tutorial they seem more willing to read the details to get a better understanding of the building blocks.

5 The Wedge

Virtual reality can be used as an important tool in scientific discovery, but can also be a great help in motivating the study of computational science. We use a 3D virtual reality theatre co-designed by one of the authors, Henry Gardner, as a core of the eScience program and as an environment for our outreach program. The Wedge [9] is a two-walled, walk-in, virtual reality theatre. It was originally built as a cheap alternative to the CAVE [7] for visualising scientific and engineering data. Participants look towards the vertex of the two screens and wear light, shutter-glasses which synchronise with the time-multiplexed images being calculated by the computer. The false-stereo effect makes images appear to float in space in front of the screens. Like the CAVE, the Wedge enables participants to see each other, the theatre and themselves as well as having a compelling experience of being "immersed" in the 3D graphics. Because of this, it is a good environment for collaborative virtual reality even though only one viewer has the ideal viewpoint. Built originally at ANU, Wedges have also been installed at two other Australian universities and in two science museums. A portable version of the Wedge has been used for K-12 outreach in country areas of New South Wales

and has been taken to conferences (one of which was on an island in the Great Barrier Reef!). A view of the largest Wedge constructed (the "WedgeOrama") is shown in Fig. 2.

Fig. 2. View of the largest Wedge installation: the "WedgeOrama"

It is not possible to take any graphics application and display it in a straightforward fashion on the Wedge. Instead, care needs to be taken to ensure that, the software interface to the graphics hardware (commonly using the OpenGL™ application programming interface) can be controlled to ensure that the scene is rendered in stereo and that each stereo buffer for each eye is rendered with the correct perspective transformation. The software also needs to communicate the eye coordinates to the graphics renderer – either by assuming a default viewer position or by interpreting the signals from an external, tracked, device (such as a hat) worn by the lead viewer. Combining all of this with a remote, tracked, mouse and a keyboard extender and an 8-speaker surround sound system makes programming the ANU Wedge a sophisticated exercise in interaction design.

Over the years a number of software interfaces have been written to the Wedge. Students undertaking projects in the eScience program usually make use of one which is based on Java™ and Java3D™. This interface, known as the Tracked Interactive Wedge Interface or TIWI [11], has been developed to a sophisticated level but programmers need to have taken courses in Java™ and in computer graphics before they are able to begin programming with it. It is a further, exciting, pedagogical hook for our eScience teaching program but it does not translate readily to mainstream computational scientists.

As an example of a sophisticated Wedge application, Fig. 3 shows an eScience student using the Wedge to design paths in 3D space which are later used to position sounds and so construct a fully 3D sound scene. This application was written by the student for their eScience project [10].

Fig. 3. An eScience student operating an interactive sound scene spatialisation program in the Wedge theatre

6 Porting VPython to the Wedge

As part of an eScience masters project, one of the authors, Shaun Press, ported VPython to the wedge. This provides an example of how the use of the Wedge and VPython provides a vehicle for our education program, in this case at the graduate level.

The porting of VPython turned out to be an interesting challenge. The basic rendering in VPython uses OpenGL, but as it turned out, VPython has only a limited dependence upon OpenGL. It utilises OpenGL functions for drawing and colouring shapes, but very little else. In a standard OpenGL implementation the majority of these tasks would be carried out via OpenGL calls. In VPython most of these functions are calculated via matrix multiplication operations to create the final product, which is the physical vertices for the shape.

To port VPython to the Wedge required a modification to the VPython rendering routines. Rather than rendering a single scene to the screen, the scene had to be transformed and rendered 4 times. Each scene was broken up into an eye/screen pair (ie Left Eye/Left Screen, Right Eye/Right Screen etc). The steps for rendering are now

- Switch stereo on
- Calculate eye position and eye offset for each eye/screen pair.
- Transform the screen for each eye/screen pair
- Render scene in OpenGL stereo buffers
- Swap OpenGL buffers to front

Code for handling Mouse and Keyboard input was also added. Support for the Domino interface was added, with it's major purpose to mimic the zoom and rotate functions of a desktop mouse.

In addition to porting VPython to the Wedge, we have also made a number of other extensions to the VPython. Additional geometries (Pyramid, Ellipsoid) have been added with other additions designed to extend the capabilities of VPython. At the time of writing a Particle Cloud object is being tested, with the intention that it be included in the next release of VPython. This object will help with the modelling of particle systems with VPython.

7 Conclusion

Our aim is to continue to leverage our investments in visualisation and VR to provide sophisticated projects for our eScience graduates which can be used in our outreach and undergraduate programs.

Acknowledgements. We would like to thank Sally Lloyd and Hugh Fisher for their help in developing the VPython tutorial and in porting VPython to the Wedge. The BComptlSci program was supported as part of the Australian Partnership for Advanced Computing Education program and the eScience program has been supported by the Australian Government's Science Lectureships Initiative.

References

1. ANU Computational Science Program. Bachelor of Computational Science Core Courses. `http://www.maths.anu.edu.au/bcomptlsci/core`, 2003.
2. ANU Computational Science Program. Bachelor of Computational Science Home Page. `http://www.maths.anu.edu.au/bcomptlsci`, 2003.
3. ANU Computational Science Program. Teaching Modules and Tutorials. `http://www.maths.anu.edu.au/comptlsci/tutorials.html`, 2003.
4. ANU Computational Science Program. VPython Bouncing Ball Tutorial. `http://www.maths.anu.edu.au/comptlsci/Tutorial-Gas/tute-bounce.html`, 2003.
5. ANU Computational Science Program. VPython Gas Simulation Tutorial. `http://www.maths.anu.edu.au/comptlsci/Tutorial-Gas/tute-gas.html`, 2003.
6. ANU eScience Program. Home Page. `http://eScience.anu.edu.au`, 2003.
7. Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. De-Fanti. Surround-screen projection-based virtual reality: The design and implementation of the cave. In *SIGGRAPH 93*, pages 135–142, New York, August 1993. Association of Computing Machinery.
8. David Scherer and Bruce Sherwood and Ruth Chaby. VPython Home Page. `http://www.vpython.org`, 2003.
9. Henry Gardner and Rod Boswell. The wedge virtual reality theatre. In *Proceedings of the Apple University Consortium Conference*, pages 23–26, New York, August 2001. Association of Computing Machinery. `http://auc.uow.edu.au/conf/conf01/downloads/AUC2001_Proceedings.pdf`, pp 2.1-2.6.
10. Rod Harris. escience project report, 2001.
11. David Walsh. The tracked interactive wedge interface. `http://ephebe/tiwi/index.html`, 2003.