MAVERIK: A Virtua l Reality System for Research and Tea c hing

Toby Howard, Roger Hubbold and **Alan Murta**
Advanced Interfaces Group, Department of Computer Science
University of Manchester, United Kingdom
toby@cs.man.ac.uk

Abstract

This paper describes some experiences with the use of the MAVERIK system for supporting undergraduate and postgraduate teaching and research. MAVERIK is a high-level system for creating and managing interactive virtual environments. It is available free under the GNU General Public Licence. MAVERIK is modular and extensible, and its use of Mesa, the free OpenGL-like graphics system, means that it can be run on low-cost PCs, making it especially suitable for use in computer graphics and visualization education. We discuss the novel architecture and main features of MAVERIK, and illustrate its use by presenting case studies of projects undertaken by our students.

Keywords: *Computer graphics education, MAVERIK, Mesa, Virtual Reality.*

1. An Introduction to MAVERIK

MAVERIK is a system for managing graphics and interaction in Virtual Reality applications [1, 2]. It is specifically designed to address the challenges of highly interactive virtual environments containing many objects with complex geometry. MAVERIK runs on PCs under the GNU/Linux operating system, using the free Mesa OpenGL-like graphics API [3], and can take advantage of 3D acceleration hardware if present. It also runs on Silicon Graphics workstations using Irix/OpenGL [4].

MAVERIK is free software, released under the GNU General Public Licence; the distribution includes all of the C source code, for both MAVERIK, and also a number of example applications (with data). This sets it apart from most commercial virtual reality systems, for which the source code is typically not available – or may be available, but at a cost prohibitive to educators. MAVERIK has a highly modular structure, and comprises a **micro-kernel** and a collection of **supporting modules**. The micro-kernel implements a set of core services, and a framework that applications can use to build complete virtual environments and virtual reality interfaces. The supporting modules contain default methods for optimised display management including spatial management, culling, interaction and navigation, and control of conventional and VR input and output devices. MAVERIK's structure allows these default methods to be customised to operate directly on application data, so that optimal representations and algorithms can be employed.

The MAVERIK micro-kernel uses a simple object-oriented **class structure**. Following normal OO philosophy, each class has an associated set of **methods**. For example, for geometric primitives, default methods are provided for displaying them, and for selecting them with a mouse. To keep the implementation simple, methods are implemented as callback functions. Changing a method is as simple as writing a new (or modified) callback function and registering it with the MAVERIK kernel. The kernel also provides a mechanism for defining new classes, making the system extensible. This approach was a deliberate design decision – we wanted the system to be easy to understand and install. In a student context, we have found that using C and callbacks has meant that those with basic programming skills can quickly become proficient at using the system.

A key difference between MAVERIK and many other VR systems is that MAVERIK does not use its own data structure for storing application data. Of course, the kernel does have data structures for managing classes and their methods, but as far as possible we have tried to avoid imposing data structures on applications. Instead, classes of object types can be created which suit the needs of particular applications. Methods for displaying and interacting with these objects are then defined and registered with the kernel. This means that only a **single representation** of the application data needs to be maintained. Any changes to this data are automatically reflected as soon as the next frame is displayed, because MAVERIK uses immediate mode rendering. The default supporting modules take much of the work out of writing the display and interaction methods.

MAVERIK is distributed with nineteen default object classes, and standard methods for navigation and for picking objects using a standard mouse and keyboard. In our research laboratory we also employ 3D mice based on Polhemus magnetic trackers, and code for handling these is included in the distribution. We also use head-mounted displays and large-screen projection systems and MAVERIK includes the code needed to compute stereo projections, and to synchronise frame buffer updates. Another part of the distribution is a set of tutorial examples, which are described in the documentation (in PostScript, HTML, and on-line man page formats), and some more advanced demonstrations which illustrate the use of MAVERIK for some of our own research applications.

2. Using MAVERIK for Teaching

There are several reasons we believe MAVERIK is useful for teaching computer graphics and visualization:

1. MAVERIK is free. Students can run it on their own PCs, allowing them to study in their own time as well as on campus. Similarly, other educators have free access to the system. MAVERIK runs on relatively low-cost PC hardware, relying only on other software which is also available free.

2. MAVERIK is a fully featured, professional-level system, not a small subset developed specifically for teaching. We use it in our own research to explore very large, complex virtual environments. For example, polygonal representations of some of our industrial CAD models would amount to more than 100 million polygons. Examples can be found on our Web pages [5].

3. MAVERIK implements a range of algorithms – such as culling, using a hierarchy of bounding volumes or occlusion techniques, object tessellation, computing projections, and stereoscopic viewing – which students can study to see how they work. The availability of source code means they can learn by example and by experimenting for themselves by changing default methods.

4. MAVERIK is distributed with default methods which handle the most common cases – such as primitives like boxes, cones, spheres, cylinders, tori, polygon meshes. These serve as examples of how to build other types of objects: NURBS perhaps, or even Jell-O! For example, as we describe in one of the case studies, it proved straightforward to add a new class of object to implement superquadric shapes.

3. Case Studies

We use MAVERIK extensively in our own teaching –for final year six-month undergraduate projects, and for postgraduate work – as well as our own research. In this section we present short accounts of a selection of recent and on-going student projects which illustrate the diverse range of applications of MAVERIK in an educational context.

3.1. Builder – Direct 3D Interactive Construction

Builder is a 3D object manipulation program (see Figure 1). It was designed to explore techniques for selecting and manipulating hierarchically structured objects in 3D space, using two-handed manipulation. The manipulation is carried out using a pair of 3D mice, and the resulting images are shown on a stereo projection screen, a head-mounted display, or a normal monitor. Builder uses voice recognition to issue commands, because both hands use the 3D mice.

Builder is a very dynamic program. Much as in a 2D drawing program, objects can be selected and stretched or squashed, using handles attached to key points on their bounding boxes. Objects can be created, deleted, copied, pulled apart, joined together, and positioned in space. Each operation may be reinforced using sound cues, via a MIDI interface. It is possible to enter a virtual environment and to completely rearrange it using these techniques.

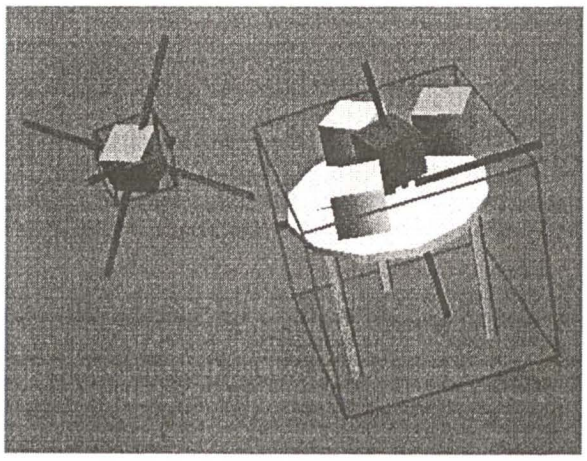

Figure 1. Builder: an interactive 3D modeller.

Buttons on the mice differentiate between moving around the environment and picking up objects. The two mice can be used interchangeably, so that it is possible to pick up an object in one hand and to carry the object around, using the other mouse to control navigation.

Builder relies on a key feature of MAVERIK – that it uses only the application's data and a set of callback functions to update the display. There is no separate scene graph or other data structure that needs to be synchronised with changes to the application's data. It also uses MAVERIK's culling methods to permit quite large environments to be displayed at fast frame rates, which is essential when using an HMD.

3.2. Virtual Lego

Another approach to interactive model building is illustrated in the Virtual Lego project (see Figure 2). Here we consider the joining and breaking apart of models consisting of a set of bricks. It differs from Builder in that constraints are applied to only allow the joining of shapes in a small number of known arrangements, and the system applies continuous object clash detection during the assembly process. Interaction is achieved using a combination of 2D mouse and keyboard.

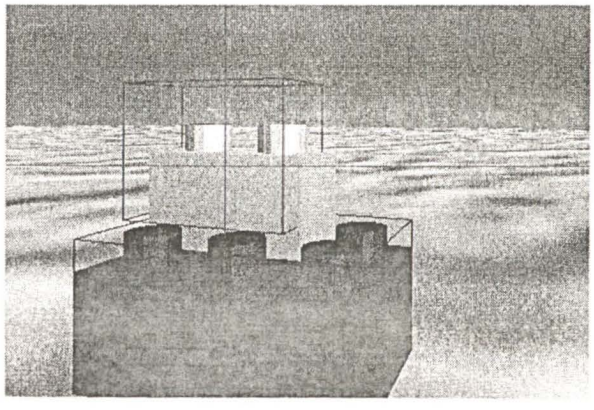

Figure 2. Constrained Lego brick manipulation.

3.3. Professor Dijkstra goes Walkabout

One of the demonstration programs distributed with MAVERIK is a virtual city ("cityscape"), with buildings, roads, trees, pavements, grassy areas, monuments, and animated avatars. This project explored extensions to the cityscape program to include a route planning mechanism. The initial method used Dijkstra's algorithm for computing the minimum cost path between any two places. To provide a more efficient implementation, a second version was implemented based on the Floyd-Walshall algorithm. Both of these algorithms are used to find the minimum cost for traversing a graph structure, and are standard algorithms taught to Computer Science students. The project used the Xforms interface builder [6] to provide a GUI. A speech recognition system can also be used to instruct the program. The project is quite interesting from two angles.

First, it is an interactive city guide. Using the interface, you can ask Professor Dijkstra – represented by an avatar – to guide you around the city (see Figure 3). For example, you may ask to be taken to the railway station, or the public library. The program builds a graph structure for all of the connected streets, and you are guided along the pavements, traversing roads at junctions or pedestrian crossings, using the shortest route. The result is displayed as an animated walk-through. At any time, you can stop or move away from your guide, and he will recognise this and wait for you.

Second, it is a nice illustration of the Dijkstra and Floyd-Walshall algorithms in practice. While the program is running, obstacles can be introduced, interactively, to block streets – to simulate road works, or perhaps an incident like a fire, for example – and the algorithms then search for the best way around the obstacles via alternative streets.

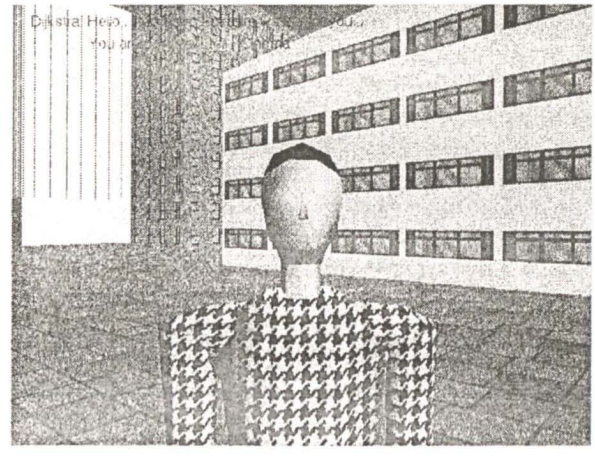

Figure 3. Professor Dijkstra goes walkabout.

This program exploits several features of MAVERIK. First, it uses occlusion culling algorithms to achieve good frame rates, even for large cities (e.g. 40 blocks square). Second, the navigation methods can be employed to wander around the city at will. Third, MAVERIK's default avatar class is used to display representations of

actions of people in the city. The avatars are animated, and walk around the city in real-time.

3.4. Head-Driven Navigation

Specifying scene navigation within interactive 3D environments is often based on 2D mouse movements and key-presses, or uses 3D mouse or dataglove gestures. This project considered the use of head motion (tracked using a 3D Polhemus sensor) to trigger user movement within a virtual environment. This may be useful when both hands are already involved in a manipulation task, for example. Navigation behaviours were devised which map head motion onto view rotations and translations. Alternative mappings were tried for both head-mounted displays and fixed large-screen projection screens. MAVERIK proved to be a good platform for investigations of this kind, since the user is free to customise navigation functions for specific applications. User tests were performed to compare the usability of the head-driven approach with more traditional hand-driven navigation schemes. These included a timed fly-through of a twisting 3D tube, and the navigation of a slalom course.

3.5. Out-of-Body Experiences

Navigation in virtual environments is a difficult task. One of the main hindrances is the limited field of view afforded by head-mounted displays and projection systems, as this is typically much narrower than we experience in the real world. It has been described as like looking at the world through a pair of toilet rolls! This makes life difficult, because it means that one tends to blunder into objects that are just out of view.

An important aspect of virtual reality is the notion of presence – that is, of being present in a virtual environment and of being surrounded by it. Thus, when navigating around, one is normally presented with a view corresponding directly to that of one's virtual body – an "in-body" or first person view. However, an alternative way to navigate is to follow a few paces behind one's virtual body – a kind of "out-of-body experience". In this case, one still controls one's body, but sees it as a separate representation, or avatar. This type of interface can be found in some PC games, such as Tomb Raider [7].

The purpose of this project was to investigate whether navigation is more easily controlled using an in-body or an out-of-body paradigm. One reason why the latter might be easier is that it affords a much wider field of view – you can see obstacles surrounding your virtual body that would not normally be within your field of view using the in-body approach. The project entailed devising some simple virtual environments containing obstacles and then monitoring the performance of volunteers in negotiating their way along prescribed routes. The code was instrumented to measure results, such as the number of collisions with objects, the length of the path followed, and the time taken to complete a route.

3.6. Visualization of Nanotechnology

This project investigated how computer graphics and Virtual Reality techniques might be used to visualize and control the construction and activity of nanomachines. The project involved the simulation of molecular interactions, and the modelling of collections of molecules organised into mechanical structures (see Figure 4).

This was an ambitious project, and MAVERIK proved especially useful – its provision of high-level virtual environment management enabled the student to concentrate on the application problem, rather than the infrastructure needed to manage the virtual environment.

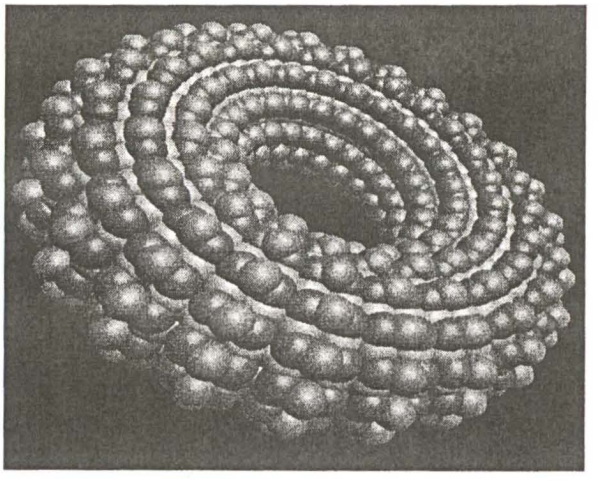

Figure 4. Visualizing nanotechnology.

3.7. A Clutterizer for Virtual Environments

A virtual environment is typically constructed by creating accurate geometrical architecture, and populating the scene with specific objects (and textures) at specific locations. Such environments are usually quite tidy. Real environments, however, are rarely tidy, having accumulated clutter over time.

The purpose of this project was to investigate methods for automatically "clutterizing"' a virtual environment, by semi-automatically adding "junk" geometry and texture.

MAVERIK provides default methods for testing for collision detection between objects in the virtual environment, and this feature facilitated the development of algorithms for rapidly populating the environment with large numbers of "clutter" objects, distributed statistically.

3.8. Interactive City Generation

This undergraduate project investigated methods for automatically generating realistic cities (see Figure 5). The project implemented a suite of three separate programs; the first generated a plausible layout of streets within the city; the second then populated the streets with buildings, monuments, parks and other urban features. The final program was an interactive viewer, which used MAVERIK for visualization, and navigation within the scene.

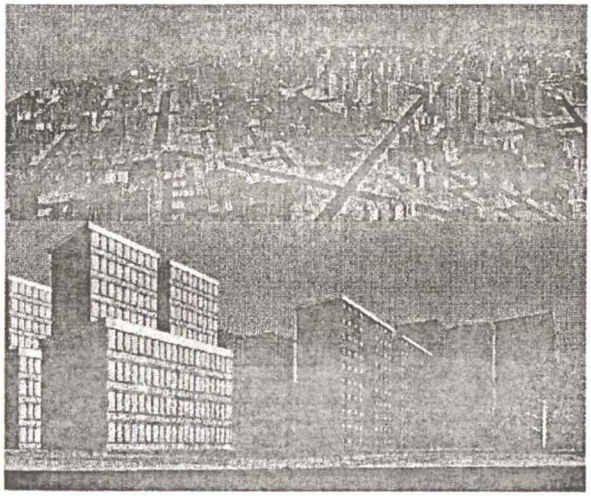

Figure 5. Interactive city generation.

3.9. Line of Sight and Trajectory Analysis for Scene of Crime Reconstruction

Working in collaboration with Greater Manchester Police, we have constructed a prototype virtual environment corresponding to a real scene of crime [8]. Subsequently, this undergraduate project undertook an investigation of two areas: first, methods for testing "line of sight" scenarios, to accurately determine which parts of a scene are visible from certain vantage points; second, for interactively tracing and modelling bullet trajectories within the scene.

MAVERIK provided the virtual environment infrastructure, and its default methods for computing vector-object intersections assisted the modelling of bullet trajectories, and multiple ricochets from rigid surfaces. The project implemented two kinds of trajectory simulation: a simple Newtonian model, and a "point mass" model which takes into account aerodynamic drag on a bullet. Figure 6 shows a simulated bullet trajectory and ricochet.

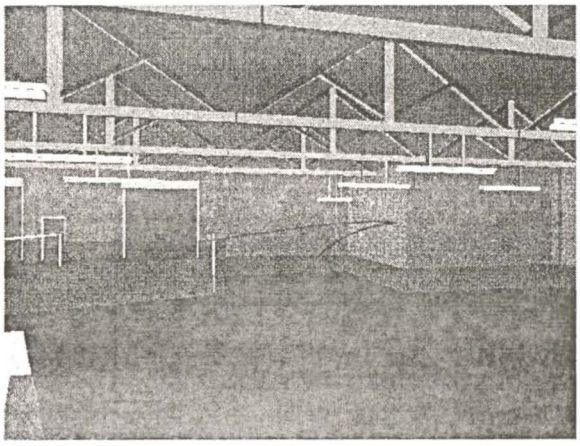

Figure 6. Visualization of bullet trajectories.

3.10. Simulating Undersea Viewing Conditions

We are interested in using VR as a means of rehearsing assembly and maintenance procedures on large industrial structures, such as offshore gas and oil platforms. This project investigated the graphical portrayal of the kind of environments encountered during such undersea operations (see Figure 7). In particular, the incorporation of visual phenomena such as animated caustics, lighting effects within participating media, limited visibility (fogging) and the presence of particulate matter in the environment were considered. This project used MAVERIK for navigation, interaction and scene management, with additional use of Mesa/OpenGL to achieve specific graphical effects. Real time undersea effects (over 20 frames per second) were achieved on a modest PC with a 3D graphics card.

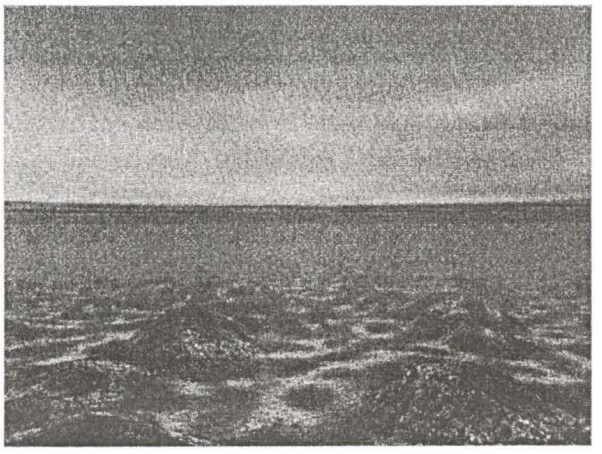

Figure 7. Shallow undersea view featuring animated lighting effects.

3.11. Interactive Modelling Using Superquadrics

Superquadrics are a family of solids derived from the parametric forms of the basic quadric surfaces – ellipsoids, tori and hyperboloids. Superquadrics are, however, much more flexible than standard quadrics and can represent a huge variety of shapes. This postgraduate project is investigating the use of interactively deformed superquadrics for object modelling in a virtual environment (see Figure 8).

This involved adding a new, quite complex, object class to MAVERIK. In practice this proved to be straightforward: the application defines a data structure for the object, and a minimal number of methods – which it registers with MAVERIK – to operate upon it.

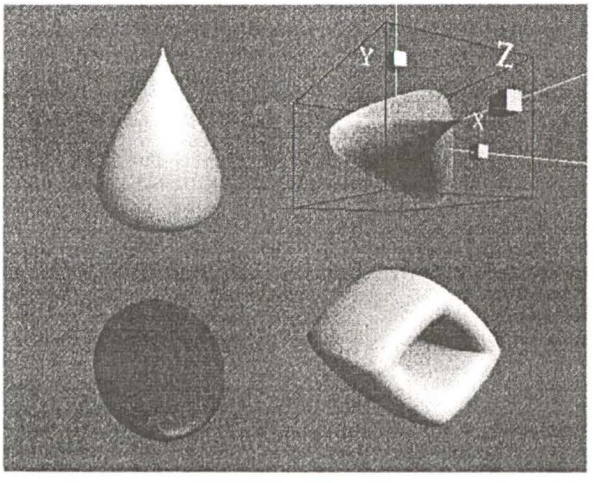

Figure 8. Interactive deformation of superquadrics.

4. Conclusions

We have been using MAVERIK as the basis for our teaching and project work since 1996, and have been extremely encouraged by its success. Because the source code is freely available, students have the opportunity to look inside and see exactly how a large professional-level software system is structured, and they can examine the algorithms in detail. And because MAVERIK uses Mesa as its rendering engine, computer graphics students can go further, and see exactly how the rendering pipeline is implemented. As for project work, because MAVERIK manages the complexity of the virtual environment, students are able to pursue more ambitious projects than have previously been realistic.

It is our hope that sharing our positive experiences of teaching using MAVERIK will be of benefit to other educators. We would be delighted if others would consider evaluating MAVERIK as a suitable tool for computer graphics and visualization education.

Acknowledgements

We would like to thank our colleagues in the Advanced Interfaces Group: Jon Cook, Simon Gibson, Martin Keates, Steve Pettifer, Adrian West – all of whom have contributed substantially to the MAVERIK system. And we'd especially like to thank our students for their enthusiasm, and for implementing a series of fun and interesting projects for us to report: Osian Ap Garth, Tim Davis, Sinem Guven, Chris Kirk, James Pearce, Jamie Pratt, Ahmed Rahali, Chris Redburn, James Sinnott, John Taylor and Irina Titovich.

MAVERIK was developed as part of the VRLSA project, funded by the Engineering and Physical Sciences Research Council (GR/K99701), with additional support from our industrial partners, CADCentre Ltd, Brown & Root Ltd, and Sharp Laboratories of Europe Ltd.

MAVERIK is available in tar and RPM formats from http://aig.cs.man.ac.uk/systems/Maverik/.

References

[1] Roger Hubbold, Dongbo Xiao and Simon Gibson. MAVERIK: The Manchester Virtual Environment Interface Kernel. In *Virtual Environments and Scientific Visualization 1996*, M. Gobel, J. David, P. Slavik, J.J. van

Wijk (Editors).

[2] Jon Cook, Roger Hubbold and Martin Keates. Virtual Reality for Large-Scale Industrial Applications. In *Future Generation Computer Systems*, **14**, pp. 157–166, 1998.

[3] http://www.mesa3d.org.

[4] http://www.opengl.org.

[5] http://aig.cs.man.ac.uk/systems/Maverik/.

[6] http://bragg.phys.uwm.edu/xforms/.

[7] http://www.tombraider.com.

[8] Alan Murta, Simon Gibson, Toby Howard, Roger Hubbold and Adrian West. Modelling and Rendering for Scene of Crime Reconstruction: A Case Study. In *Proceedings Eurographics UK*, pp. 169–173, 1998.