# A Virtual Jacob's Ladder

Mashhuda Glencross and Alan Murta

Advanced Interfaces Group
Manchester, UK

## Abstract

An algorithm for modelling and simulating a virtual Jacob's ladder is presented. Its motion is computed using our physically based simulator Iota. The model used to represent the Jacob's ladder is dynamically restructured during the simulation via scripted events, but can also be performed interactively by the user.

Modelling a Jacob's ladder is achieved by reducing the problem to two dimensions by observing that the connectivity of blocks results in the loss of a degree of freedom in the hinges. Simulating the motion of the model is affected by dynamically altering the connectivity of blocks in the model.

Finally, we show the Jacob's ladder in the context of a virtual environment in which a user can navigate and drive the simulation in real-time.

**Keywords:** constraint based, physically based, interaction forces, rigid bodies, multi bodies, particle systems, virtual reality

## 1 Introduction

Many researchers world-wide have investigated various techniques to improve the quality of the graphics in virtual reality (VR) systems [9, 8]. The majority of the VR demos we see today offer little more than the opportunity to walk through a largely static environment and interact with it at a very basic level by, for example, picking up objects. Users are provided with many labyrinths to explore but there is little to actually capture the imagination.

A number of dynamic VR systems do exist, but they tend to be for specific applications such as simulating the behaviour of crowds in an emergency[1]. More interestingly, a small number of people are now working on general purpose real-time physics based simulators [6, 11, 3] to compute the behaviour of virtual objects. The motivation of this type of research lies in the desire to create dynamic virtual environments (VEs).

The applications for software capable of plausibly simulating the motion of objects subject to real world physics at suitably interactive frame rates is widespread. Many VR systems designed to aid any sort of assembly process [5] can benefit from a good simulator. Moreover the entertainment appeal of such a system in the games industry is vast and this prospect has motivated the development of software such as MathEngine [11].

In this paper we describe an algorithm which uses our general purpose physically based simulator software (Iota) for modelling and interactive simulation of a children's toy. The simulator uses rigid bodies, particle systems forces and hinges to model objects [6], and we give an overview here.

[1]for example Colt VR's Vector software

## 2 Overview of the Simulator

The core simulator shown in Figure 1 consists of a scene modification block which can be used to preprocess the scene if required, and then the frame is advanced. This requires a

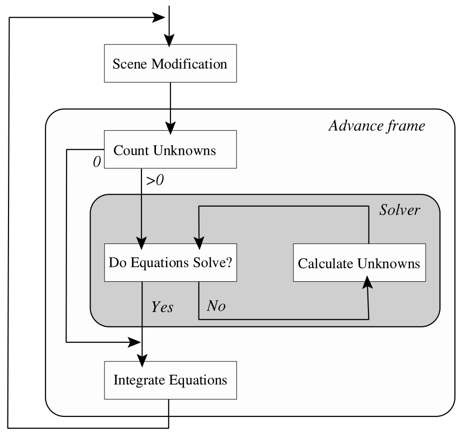

Figure 1: Overview of the simulator

number of different types of computation.

The physical model incorporates both interacting particle systems and articulated rigid bodies, so a scene could consist of any combination of them. Indeed the types of interactions possible can be very powerful and can be used to simulate quite subtle but sophisticated behaviours. More importantly interactions may occur between:

- Particles
- Rigid bodies
- Articulating rigid bodies
- Particles and rigid bodies
- Particles and parts of articulating rigid bodies

External forces may also be used to direct the motion of particles. Once all these forces are applied to the particles upon which they act, the constraint forces (if any) in hinges need to be computed.

For this part of the computation the simulator calls upon the services of our numerical solver module which is based on Newton Raphson [10] with line backtracking. We have modified the solver to make it more robust, which is essential in a VR application, as a model which spontaneously falls apart when least expected can be disconcerting. Once all the unknown forces in the hinges have been computed, they are also accumulated onto the particles upon which they act. Finally an Euler integration is used to integrate the equations of motion over a time interval.

The scene modification (preprocessing) stage gives us an opportunity to restructure the whole scene taking account of any changes to the data as a result of user intervention. Users are able to assemble and disassemble (dynamically restructure) models at run-time via the use of combine and separate routines. Broadly speaking the combine routine combines two particles passed to it into one particle and the separate routine separates them. The specific behaviour of these is complex and depends on the relationship between the two particles passed to them. For readers seeking further details please refer to *A Framework for Physically Based Modelling in Virtual Environments* [7]

The Simulator engine is written in C++ and interfaced to Perl which is used as the system scripting language, while GNU MAVERIK is used for the VR graphics.

## 3 Simulating a Jacob's Ladder

In this section we introduce a children's toy known as a "Jacob's Ladder". It is traditionally constructed using blocks of wood and ribbons as shown in Figure 2. Due to way in

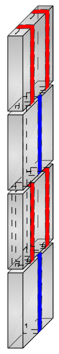

Figure 2: Jacob's ladder

which the ribbons connect to the blocks, each block can only tip in one direction which alternates along the length of the toy. If there are enough blocks in the ladder, multiple ripples can be sent down its length. Consider the situation in which two ripples are propagating along the toy; should the situation arise where the second ripple started too soon after the first, it may catch up and cancel them both out.

To be able to simulate the motion of a Jacob's ladder, it is necessary to understand the behaviour of a hinge (two blocks) as shown in Figure 3. Let us assume that the lower

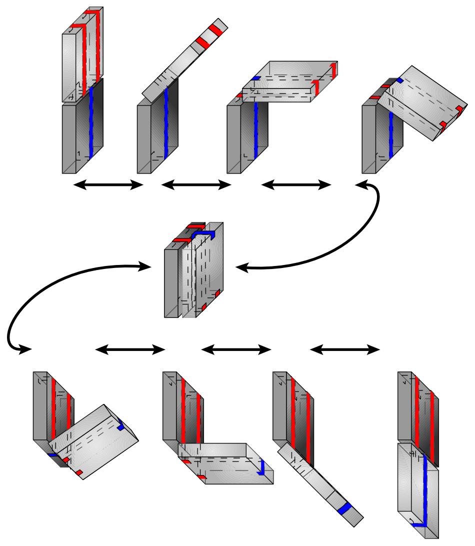

Figure 3: The behaviour of a single hinge in a Jacob's ladder

block is held still so that the behaviour of the upper block can be seen in relation to it. The upper (lighter grey) block starts at the top with two outer ribbons on it and the second block starts with just one central ribbon. When it is tipped over, it swaps to be at the bottom turning 180° at each end of the second block. Notice how the ribbons swap so that the block now at the top still has two ribbons.

Now consider what happens when we hold the first block and allow it to tip. Say it rotates clockwise through 180° bringing the top of it in contact with the bottom of the next block. If we continue to hold it then the second block tips through 180° anti-clockwise and so on, allowing a ripple to propagate down the length of the ladder. The important thing to realise is that each block's orientation is flipped and this can be seen by the alternation of the ribbons on the face of each block. The motion of the blocks appears simple at a first glance, but as we can see it is complex and can be hard to visualise. To be able to simulate this type of motion we need to specify a suitable model which will exhibit the appropriate behaviour.

## 4 Building the Model

To utilise the physical model for computing motion in the simulator, each block is modelled using simulator primitives. A number of possible configurations of these primitives could be used to construct the model. In the interests of performance it would be desirable to minimise the amount of computation required. A carefully chosen model can significantly improve the interactive response of the simulation.

Imagine viewing a Jacob's ladder side-on as shown in Figure 2. In this diagram the toy is oriented such that the narrow side face of each block appears to be at the front with the remainder receding into the paper. Our model abstracts away much of the detail while trying to secure as much of the behaviour as possible. This is achieved by retaining the essence of a block by representing it as a two dimensional body with four particles situated at the corners of the narrow face and one in the centre. The advantage of this approach being that each hinge consists of two particles thus providing a contribution of one equation in each dimension. This means that only two equations per hinge have to be solved simultaneously. We justify this by the observation that the hinges in the ladder cannot twist and so each hinge actually only has two degrees of freedom.

Unlike the model, blocks in a real Jacob's ladder do not actually rotate about a hinge but instead approximately about a point where the ribbons on each block cross. However at speed it appears simply as if the hinging occurs at the point where the blocks meet, corresponding to the hinges in our model. This further justifies the approach taken to model the Jacob's ladder. A numbering convention, shown in Figure 4 was adopted to represent the bodies and particles which make up the model. Each block is constructed from a body of

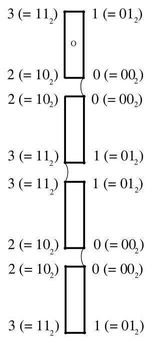

Figure 4: Numbering convention adopted in Jacob's ladder

five particles numbered from 0 to $N$. The four corner ones are numbered such that they reflect the system's symmetry and this in turn allows code to be written using relationships between adjacent blocks to reduce duplication.

A number of observations may be made with respect to this labelling of particles:

- Particles which hinge will always be numbered the same on both blocks
- Odd numbered blocks (1, 3, 5, ... ) will hinge on corners 0 and 1 with previous blocks
- Even numbered blocks (0, 2, 4, ... ) will hinge with the previous block on corners 2 and 3
- Exclusive-oring with $01_2$ will enable us to traverse a block along its length from any corner.
- Similarly, exclusive-oring with $10_2$ enables the width of a block to be traversed from any corner.
- It has been found useful to view each block as possessing two orientations, *lengthwise* and *widthwise*. In the initial state lengthwise orientations are alternately up and down, whereas widthwise orientations point right to left.

The implications of these observations will become clearer in the following section.

## 5 Simulating the Model

Actually simulating the motion of the entire ladder involves specifying the behaviour of a single hinge. Once the constraints on the motion of the model are understood, the physical model in the simulator can be used to compute the overall motion of the system. As we will see, many subtleties in the motion of the real toy appear in the virtual one. First let us consider the motion of one hinge again, but this time in a more rigorous fashion.

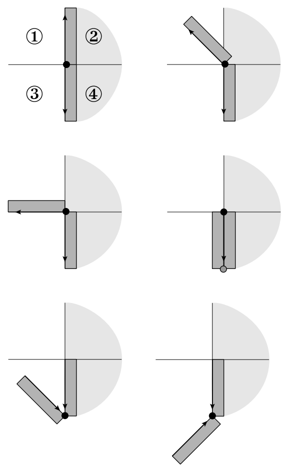

Figure 5: Allowable motion of a hinge in a Jacob's ladder

Figure 5 shows a pair of hinged blocks. Their range of allowable motion is confined to quadrants 1 and 3. Notice the position of the black dot representing the hinge as it is significant and governs the direction in which a block is allowed to tip. Moreover observe the lengthwise orientation vectors on the blocks as these too are also important.

Consider the first configuration of the pair of blocks, assuming that the second block remains stationary. If an attempt is made to push the first block into the shaded region both blocks will *lock* preventing any further intrusion into that region. The first block is allowed to move into quadrant 1 if it is subject to an unbalanced force which encourages it to do so. Following the sequence of allowable motion of the block it can be seen that after a 180° rotation the hinge is broken and reassembled at the bottom end of the second block and then the first block is allowed to continue through another 180° turn. This swapping of the order of the blocks is termed a *flip-flop*.

Now observe the relationship of the lengthwise orientation vectors, initially the first block's vector points upwards and the second block's points downwards. After a 180° turn the vectors both point down and after a flip-flop they point towards the hinge. These orientations can be examined to determine a couple of things. Firstly if the magnitude of the cross product of the two vectors is less than zero then an attempt has been made to enter quadrant 2 or 4. This means that some sort of action is required, the type of which can be determined by adding both vectors together and examining their magnitude. A small number resulting from the sum of the vectors implies that the pair of blocks have to be locked (see Algorithm 3) whereas a large number implies that the blocks have to be flip-flopped (see Algorithm 2). If the magnitude of the cross product of both vectors is greater than zero then no action is required.

The behaviour of one pair of hinging blocks has now been formalised so we can describe the algorithm for simulating the Jacob's ladder which is shown in Algorithm 1.

**Algorithm 1** Main Jacob's ladder algorithm

```
1:  for all block in ladder (excluding first one) do
2:    if hinge is locked then
3:      if torques acting on current and previous blocks
        cause them to unlock then
4:        Unlock hinge {Algorithm 4}
5:      end if
6:    else
7:      Calculate lengthwise orientation of current block
8:      Calculate lengthwise orientation of previous block
9:      Calculate cross-product of both orientations
10:     if cross-product $< 0$ then
11:       Sum both orientations
12:       if magnitude of sum is large then
13:         Perform flip-flop {Algorithm 2}
14:       else
15:         Lock blocks {Algorithm 3}
16:       end if
17:     else
18:       {Normal tipping – No action}
19:     end if
20:   end if
21: end for
```

Our flip-flop algorithm works by performing a separate on the two particles which make up the hinge. To identify the new particles to hinge on we exclusive-or with 1 ($01_2$) and combine these particles.

The flip-flop also causes the configuration of ribbons to change. While this is only cosmetic, it is necessary to enhance the realism of the simulation. There are a number of possible ways to treat the ribbons, but for performance reasons we decided to create all of them and hide those which are not visible.

**Algorithm 2** Code to flip-flop blocks in the Jacob's ladder

```
1: Separate currently hinged corners
2: Calculate particle numbers for new hinge {by exclusive-
   oring current particle numbers with $01_2$}
3: Combine particles for new hinge
4: Hide and show ribbons as appropriate
```

To lock two bodies, a further combine on the hinged corner is performed as this has the action of making the hinged bodies rigid. Anticipating that the particle will be required when the hinge is unlocked, it is subsequently recreated so that there are still ten particles in the two bodies. Finally, the hinge must be marked as locked so that it can be treated appropriately.

**Algorithm 3** Code to lock hinge in Jacob's ladder

```
1: Perform a further combine on the hinged corner {This
   has the effect of causing both blocks to become consoli-
   dated into the same body}
2: Clone hinged particle {to retain ten particles in new
   body, as before}
3: Mark hinge as locked
```

Unlocking a hinge is performed by detaching the central particle from one of the bodies using separate to create a new body. Each of the block's four remaining particles must also be separated and combined with the new body. In very rare circumstances the body which has become unlocked may actually be locked to a another block and some further manipulation is required to maintain the correct parent body. Finally the hinge is recreated by combining the corners in question and then marked as unlocked.

**Algorithm 4** Code to unlock hinge in the Jacob's ladder

```
1:  {Reconstruct two separate bodies as it was prior to the
    locking action}
2:  Detach central particle from notional second body
3:  Insert new body into data structure
4:  for all particles to be repointed to second body do
5:    Detach particle from parent body
6:    Repoint to new body
7:  end for
8:  If next body is also locked, repoint it to new parent
9:  Rebuild hinge using combine
10: Mark hinge as unlocked
```

## 6 Results

The first set of results presented in Figure 6 shows screen shots of the Jacob's ladder taken at intervals of eight frames. Although it is difficult to see the subtleties of the motion from such images, we would like to draw your attention to subimage A through G. Notice how the third block is pushed down and then pulled back up over the sequence, in particular a significant change in position has occurred between E and F. From subimages G to R we have gradually navigated down to follow the propagation of the ripple. Subimage N is of interest as the third block is momentarily unable to tip because the previous two blocks have locked and their united motion disrupts it. This behaviour can occasionally be observed in the real toy. Also, notice how the second and third blocks appear to have become locked in S and T and are then possibly unlocked by U. The types of subtleties introduced can only result from the use of a physically based model. There is no way of merely giving the impression of this sort of realism.

The next sequence of images show the Jacob's ladder simulated within a virtual world. The world contains a model of the Advanced Interfaces Group laboratory at the University of Manchester. Our dynamic Jacob's ladder can be seen to appear on the monitor of one of the computers in the model as it is navigated. The results have been generated on a Silicon Graphics Indy, however, our software runs adequately on a Pentium 75MHz with 32MB RAM running Linux with GNU MAVERIK and using Mesa 3-D.

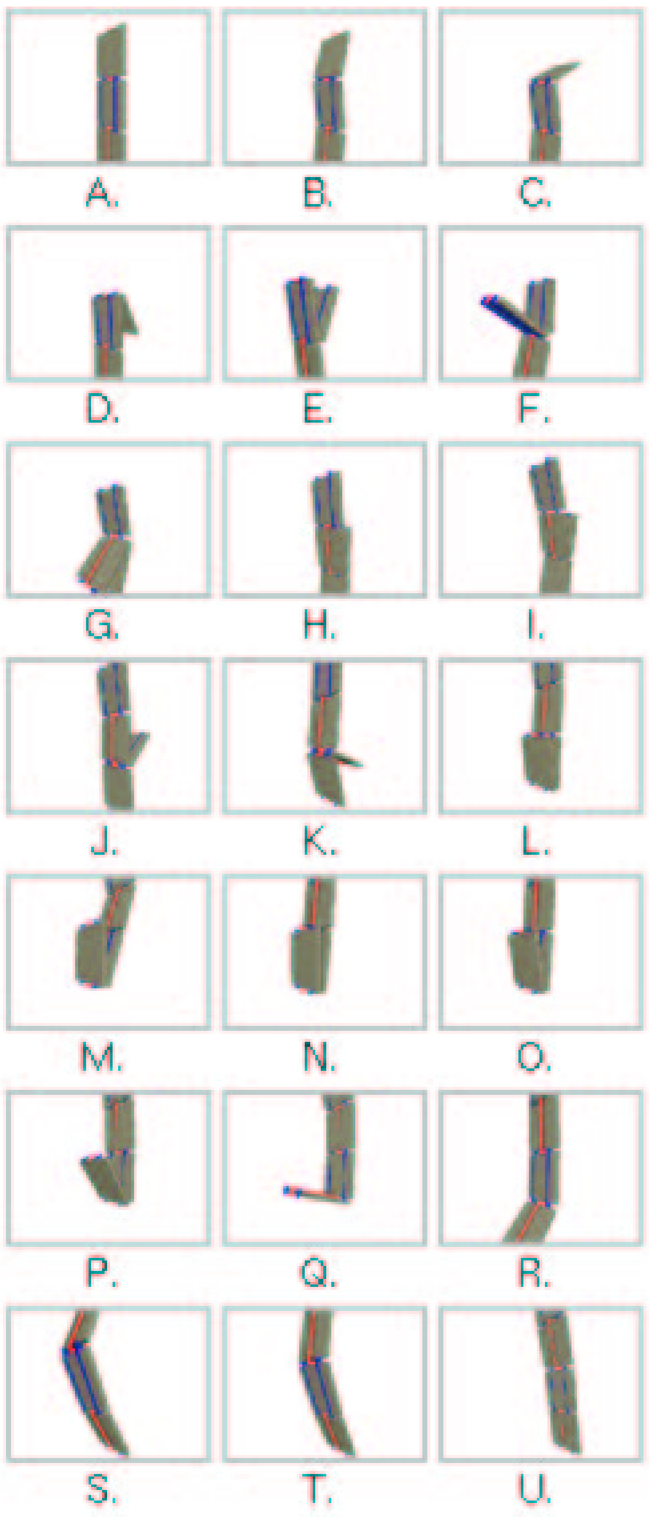

Figure 6: Jacob's ladder in action

## 7 Conclusions

Our simulator can be used to achieve a wide range of effects from flocking to articulated rigid body simulations but we chose to simulate a Jacob's ladder because it illustrates two very important properties of our simulator. Firstly, the solver is robust so there are no disconcerting anomalies introduced by it failing to converge.

Secondly, this particular example makes heavy use of our dynamic restructuring routines. Each flip-flop fundamentally changes the underlying model. We have only shown this restructuring occurring as scripted events but there is no reason why the user cannot affect such changes. Indeed any interactive breaking or reassembling is possible via these routines. Any degree of dynamic restructuring can be effected by various permutations of combines and separates.

Many constraint solvers have been implemented to generate animation sequences [2, 1], few have been tailored specifically for virtual reality applications [4, 11] and even fewer enable scene data to be dynamically restructured.

We have found that our Jacob's ladder model can be simulated at interactive frame rates. We do not implement any collision or contact constraints so blocks can be seen to occasionally pass through one another for a single frame.

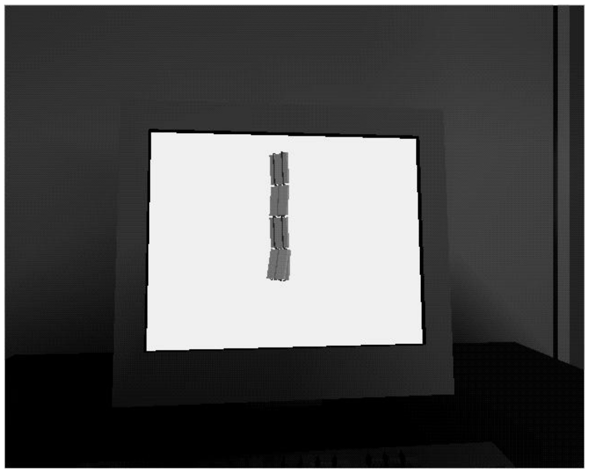
Figure 7: Jacob’s ladder in AIG lab model

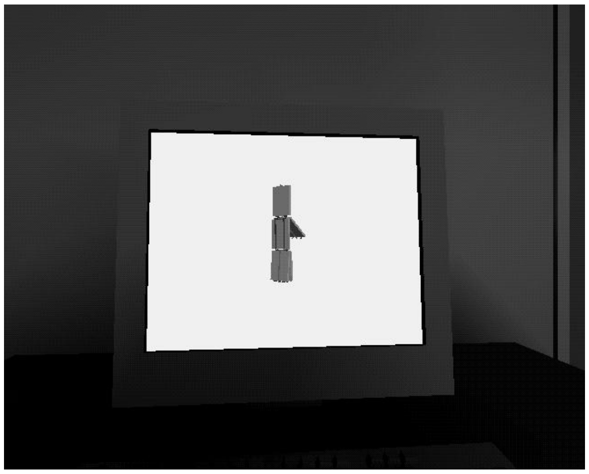
Figure 8: Jacob’s ladder in AIG lab model

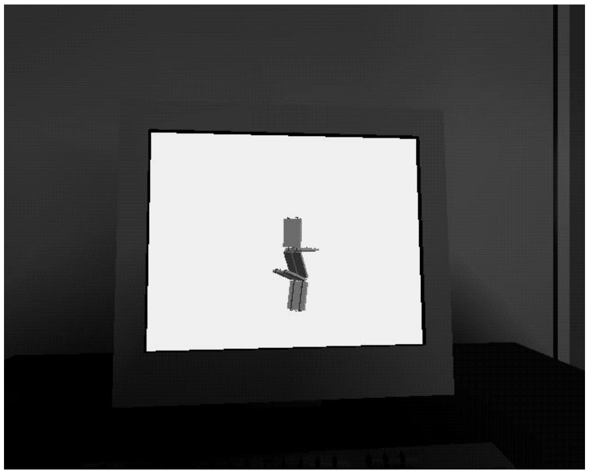
Figure 9: Jacob’s ladder in AIG lab model

Although the connectivity of blocks in a real Jacob’s ladder differs from that in our model and as such the model is not a faithful representation, our virtual ladder contains all the subtleties of motion exhibited by the real toy. Finally, the virtual Jacob’s ladder is as addictive as the real thing.

## References

[1] David Baraff. Dynamic simulation of non-penetrating rigid bodies. PhD thesis, Cornell University, 1992.

[2] Ronen Barzel and Alan Barr. A modelling system based on dynamic constraints. *ACM Computer Graphics*, 22(4):179–188, August 1988.

[3] Peter M. Chapman and Derek P.M. Wills. Towards a unified physical model for virtual environments. In *Proc. 4th UK VR-SIG Conference*, Brunel University, UK., November 1997.

[4] François Faure. Fast iterative refinement of articulated solid dynamics. *To appear in IEEE TVCG*, 1999.

[5] T. Fernando and P.M. Dew. Constraint-based interaction techniques for supporting a distributed collaborative engineering environment. In *Proceedings of the First Workshop on Simulation and Interaction in Virtual Environments*, pages 265–270, Iowa City, July 1995.

[6] Mashhuda Glencross. Multi-body simulation in virtual environments. In Richard Zobel and Dietmar Moeller, editors, *Simulation — Past, Present and Future. 12th European Simulation Multiconference*, pages 590–594, Manchester, England, June 1998.

[7] Mashhuda Glencross. *A Framework for Physically Based Modelling in Virtual Environments*. PhD thesis, Department of Computer Science, University of Manchester, 1999.

[8] R.J. Hubbold, D. Xiao, and S. Gibson. MAVERIK — the Manchester virtual environment interface kernel. In *Proceedings of the 3rd Eurographics Workshop on Virtual Environments*, Monte-Carlo, February 1996.

[9] A. Murta, S. Gibson, T.L.J. Howard, R.J. Hubbold, and A.J. West. Modelling and rendering for scene of crime reconstruction: A case study. In *Proceedings of Eurographics UK*, pages 169–173, Leeds, March 1998.

[10] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C The Art of Scientific Computing*. Cambridge University Press, Cambridge, second edition, 1992.

[11] Benjamin Wooley. Rules of the game. *Personal Computer World*, pages 272–273, May 1999. `http://www.mathengine.com`.

## Acknowledgements

We would like to thank Dr. Simon Gibson for the AIG lab model and code.

## Contacting the Authors

The authors can be contacted at:

Department of Computer Science
University of Manchester
Oxford Road
Manchester M13 9PL
United Kingdom

+44 161-275 6176

**Mashhuda Glencross:** glencross@cs.man.ac.uk

**Alan Murta:** amurta@cs.man.ac.uk