Managing the Complexity of Physically Based Modelling in Virtual Reality

Mashhuda Glencross and Alan Murta

Advanced Interfaces Group
Manchester, UK

Abstract

Physically based simulation has been used to a limited degree in virtual reality (VR) because of the performance demands imposed by the real-time requirement. Simulating complex physical models in VR is attractive as it adds a richness of detail to virtual environments which is valuable in a number of application areas.

In order to successfully use physical simulation in VR it is essential to reduce the computational cost. To this end we describe methods for minimising the complexity of models prior to and during simulation. We illustrate adaptive variation in complexity by dynamic addition and removal of primitives from a Newton's cradle model. A sequence of images is used to show this technique performs sufficiently well to enable user interaction with the cradle.

Keywords: physically based, rigid bodies, particle systems, dynamic restructuring, virtual reality.

1 Introduction

In this paper we discuss a technique for reducing the complexity of physically based simulations by dynamically adding and removing objects on the fly. We suggest that this method is ideal for use in virtual reality (VR) where real time performance and consistent behaviour is critical. To motivate the research topic addressed in this paper we begin by describing physically based modelling, why it is of value in VR and what problems make it difficult to use in real time interactive applications.

2 Physically Based Modelling in Virtual Reality

Physically based modelling involves using physics to model a particular phenomenon or object. Motion computation in animation is an area which has benefited significantly from physically based approaches. Most animation systems use Newton's laws as the basis for algorithms to simulate a wide number of things ranging from fire [10], waterfalls [10, 8], flocking animals [11], wind blown leaves [12], powders [7] and even clay [1].

Many of these systems would be desirable to simulate in virtual worlds particularly for gaming and entertainment applications. An increasing number of researchers and games companies are developing physically based simulators [2, 13]. In particular there is significant pressure in the games industry to add more realism and provide users with more exciting immersive experiences.

General purpose simulator engines all suffer from a number of problems related to the methods used to solve the equations of motion. Often, correct physically based simulation involves computations which may depend on a wide range of attributes and require sophisticated physical models. The performance cost of these models can be too high to enable their use in VR. Furthermore, the methods used to solve for the motion of complex systems are rarely totally robust and can spectacularly fail to supply an appropriate meaningful solution. In VR this is a major problem because users expect the model to behave in a particular way; if it reacts to some event in an unpredictable fashion then the model loses its plausibility. Consequently, the application risks its integrity and acceptability for wide-scale use as these types of problems are perceived as ‘bugs’ rather than an unfortunate result of the solution method.

It is currently not possible to build a totally robust general purpose simulator engine which solves the equations of motion for any type of physical system [9]. Thus if we wish to use physical simulation in VR the only realistic approach is to sufficiently simplify the problem such that a well implemented simulator would be able to consistently return acceptable results. Before we can outline possible methods for achieving this it is appropriate to briefly describe our simulator.

We have implemented a general purpose framework (collectively known as Iota) for physically based modelling in VR which consists of a purpose built simulator engine, a freely available VR kernel (GNU MAVERIK [6]), and a high level scene description environment supported through the Perl scripting language. The simulator combines a particle system with an articulated rigid body based approach. Models to simulate are constructed from particles (point masses), bodies (collections of rigidly connected particles) and systems (domain of a simulation). Particles and bodies may be interconnected through any combination of force functions and hinges. The equations of motion for models are dynamically constructed and solved using inverse and forward dynamics techniques [4].

3 Managing Complexity

Now that an impression of the primitives used to construct models to simulate has been given we can describe how we advocate managing the complexity of simulations. This is achieved by a combination of two tightly bound techniques; simplifying the model and reducing the problem domain by structuring data appropriately.

3.1 Simplifying Models

A simulation may be simplified in two ways, firstly through exploiting knowledge about the system being simulated. In order to provide freedom for customised simulations, the Iota framework does not enforce inflexible conventions or data representations. All simulator primitives can be extended at the Iota level and any type of force function can be implemented via a callback mechanism. Simulator services can be called upon if required and can be used as little or as heavily as desired.

The second method for simplifying a model is by dynamic activation or deactivation or even addition and removal of simulator primitives on the fly. This is achieved through functionality implemented in the simulator engine and enables us to adapt complexity as and when necessary [3]. A consequence of this functionality if that the inertia matrix for the system must be computed as required, although this does add a small performance cost.

3.2 Reducing Problem Domains

The problem domain of a simulation is reduced by careful internal structuring of data. We store a scene graph as a hierarchically structured tree with a system at the root, which specifies the domain of an instance of a simulation. Every primitive in an instance of a simulation is totally independent of any other instances of system that may exist. Beneath this is a layer consisting of any number of internal container objects called articulates. These are not available to the user but are used internally to store bodies related by hinges. This is very valuable because an articulate specifies the domain of an inverse dynamics solution [5]. Beneath the articulate layer is a body layer with individual particles as the leaves.

It is not possible to illustrate many of the benefits of this structuring of data in a paper but this approach provides a powerful means of managing the complexity of the scene graph [3]. Furthermore routines exist in the simulator to manipulate and analyse the scene graph. Now it is possible to illustrate some of these concepts through a simple 'ad-hoc' example within a physical context.

4 Case Study: Simulating a Newton's Cradle

A Newton's cradle is an interesting executive toy composed of a set of metal balls suspended on wires from a cradle like frame. Lifting a ball at one end of the cradle, and releasing it results in a collision between the moving and stationary set of balls. Interestingly upon impact the moving ball almost comes to an abrupt halt, and an impulse is transmitted through the stationary set causing the ball at the other end of the cradle to continue the motion. Grouping together two balls, lifting them and releasing in the manner described above, results in the impulse being transmitted through to two balls at the other end of the cradle. This behaviour can be shown for any number of balls. Furthermore if two sets of balls, one either side of the cradle, are released at the same time the groups appear to swap upon collision with the stationary set.

Now that we know how a Newton's cradle behaves, we can consider how to model it. Potentially each ball suspended from the cradle could be treated as a pendulum and the impact characteristics of the metal balls modelled. The problem with this approach is that in a VR application the Newton's cradle would simply provide subtle detail in a broader context. Therefore it is critical that the computational cost of such a simulation is minimised.

Simplifying the model relies on prior knowledge of how the system behaves so it has to be achieved by the developer. In our model we exploit our observations of the toys behaviour. We know from studying a Newton's cradle that any number of raised balls behaves like a single pendulum, they collide with a stationary set and the same number of balls continue the motion on the other side. These two sets appear to have swapped positions in the cradle. This is very important because it implies a model which will appear to be correct regardless of the number of balls raised or the combination in which they are raised. The important thing to realise is that although this type of model gives an illusion of correct behaviour, it is still sufficiently realistic to enable user participation. In the following section we outline how this model is constructed.

4.1 Building the Model

Since all balls in a group move in unison only one pendulum is required to simulate each group. Splitting the group into two changes the situation, now two pendulums are required to simulate the motion. So it is safe to say that a linear relationship exists between the number of groups and the number of pendulums required to simulate them. Figure 1 illustrates two possible groupings for the Newton's cradle together with the underlying pendulums simulated. The first underlying pendulum used to simulate the motion of a group consisting of two balls is centred about the group at an offset into the cradle of -150. A second group consisting of four balls is represented by the underlying pendulum offset at 50 units from the origin of the cradle.

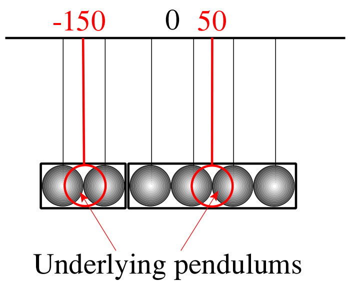

Figure 1: Newton's cradle groupings

Since there is only one pendulum in the underlying model of a group, only one simulator body has to be stored together with its constituent six particles: one at the top hinge, one at the centre of the ball and four around its circumference. An appropriate moment of inertia is maintained by placing four particles around the circumference of a ball.

A cradle with n pendulums at rest is represented by just one underlying pendulum; exactly how this state changes is described in the following section.

4.2 Simulating the Model

After a simulation begins, the state of a cradle can be altered by selecting and raising a number (x) of balls. At this point a new group is dynamically constructed consisting of x balls but with only one underlying pendulum used to model its behaviour. Until this new group collides with the stationary group its motion is automatically handled by the simulator. When a collision takes place, some action is required.

In the event of a collision an offset is applied to both groups involved and then they are swapped. A subsequent collision may occur immediately with the swapped group and its neighbour so it is necessary to retest before rendering. The effect of this manipulation is that groups are literally swapped and allowed to continue their motion.

Another issue which must also be addressed is how to handle a collision with a group that is being dragged by a user. In this case the two groups involved in the collision are dynamically consolidated into one. Without consolidation, in the worst case scenario each group could consist of just one ball adding a significant performance overhead, clearly the scenario we want to avoid. Thus dynamically consolidating groups has a performance benefit.

Figure 2 in the Appendix shows an example simulation in which a user selects, raises and releases various combinations of balls. By convention the selected ball is coloured to convey this information to the user. In subfigure B the two rightmost balls are raised and later released in subfigure G. Subsequently the three rightmost balls in subfigure K are also raised and then released in M. Note how both sets of balls are involved in a collision with the stationary group containing two balls in the centre. The number of balls in each moving group swap upon collisions. From subfigure T onwards the cradle is spun by the user. A user interacting with the Newton's cradle is shown in Figure 3 (Appendix) within the broader context of a VR model of the Advanced Interfaces Group (AIG) laboratory at Manchester University. Both simulations run at interactive frame rates on a modest PC with a Pentium 366 processor.

5 Conclusions

In this paper we have outlined the Iota system, briefly described the techniques implemented to manage complexity of simulations within this framework, and illustrated the concepts through the example of a Newton's cradle simulation.

To conclude we would like to draw the readers attention to a number of issues which bring together the various sections of this paper. In the Newton's cradle example a minimal system is simulated. This is used to illustrate complexity management by simplifying the model. Correct physics is applied to pendulum(s) so a proper physical simulation is carried out but minimised through the use of an 'ad-hoc' model which describes the behaviour of the cradle. Pendulums are dynamically added/removed from the simulation as varying complexity is required in the model.

Furthermore each underlying pendulum is held in one articulate for this particular example. The benefits of this may be unclear, but the most valuable consequence is that each articulate's motion is computed independently so there is less likelihood of the solver failing to return an acceptable solution.

The performance of the Newton's cradle example is dependent on the degree of user participation. As the user selects, lifts and drops balls new groups are being dynamically constructed, so the number of pendulums required to simulate the system may increase. This results in a higher level of complexity in the simulation. Subsequent dynamic consolidation of groups overcomes this problem.

Since the Iota framework provides a mechanism for dynamically managing the complexity of simulations the developer only has to specify appropriate logic for the application. The complexity management routines then dynamically alter the scene graph as required according to the specified criteria. Essentially, the model evolves, self adjusts when required and continues to evolve over the duration of a simulation. In the Newton's cradle example user interaction and consolidation of groups are the criteria for dynamic alteration of the scene graph.

The techniques decribed are applicable to any particle or rigid body model which is specified in terms of the primatives supported by Iota's simulator engine. Finally, further details of the implementation of our complexity management techniques are detailed in the first authors doctorate thesis [3].

References

[1] M. Desbrun and M.P. Gascuel. Highly deformable material for animation and collision processing. In *Proc. 5th Eurographics Workshop on Animation and Simulation*, Oslo, September 1994.

[2] François Faure. An energy-based method for contact force computation. In *Computer Graphics Forum (Proceedings of Eurographics 96)*, volume 15, pages 357–366, August 1996.

[3] Mashhuda Glencross. *A Framework for Physically Based Modelling in Virtual Environments*. PhD thesis, Submitted to University of Manchester, November 1999. Pending acceptance.

[4] Mashhuda Glencross and Alan Murta. Multibody 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.

[5] Mashhuda Glencross and Alan Murta. A virtual Jacob's ladder. In *Graphicon 99*, pages 88–94, Moscow, August 1999.

[6] Roger Hubbold, Martin Keates, Simon Gibson, Alan Murta, Steve Pettifer, and Adrian West. MAVERIK programmers guide. Technical Report MPG v4, University of Manchester, October 1998. Draft.

[7] G. Miller and A. Pearce. Globular dynamics: A connected particle system for animating viscous fluids. *Computers and Graphics*, 13(3):305–309, 1989.

[8] Alan Murta and James Miller. Modelling and rendering liquids in motion. In *Proceedings of WSCG*, pages 194–201, Plzen-Bory, Czech Republic, February 1999.

[9] 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.

[10] W.T. Reeves. Particle systems — a technique for modeling a class of fuzzy objects. *Computer Graphics*, 17(3):359–376, July 1983.

[11] C.W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, July 1987.

[12] Jakub Wejchert and David Haumann. Animation aerodynamics. *Computer Graphics*, 25(4):19–22, 1991.

[13] 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 Advanced Interfaces Group laboratory program.

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

A Appendix

A.1 Newton's Cradle Images

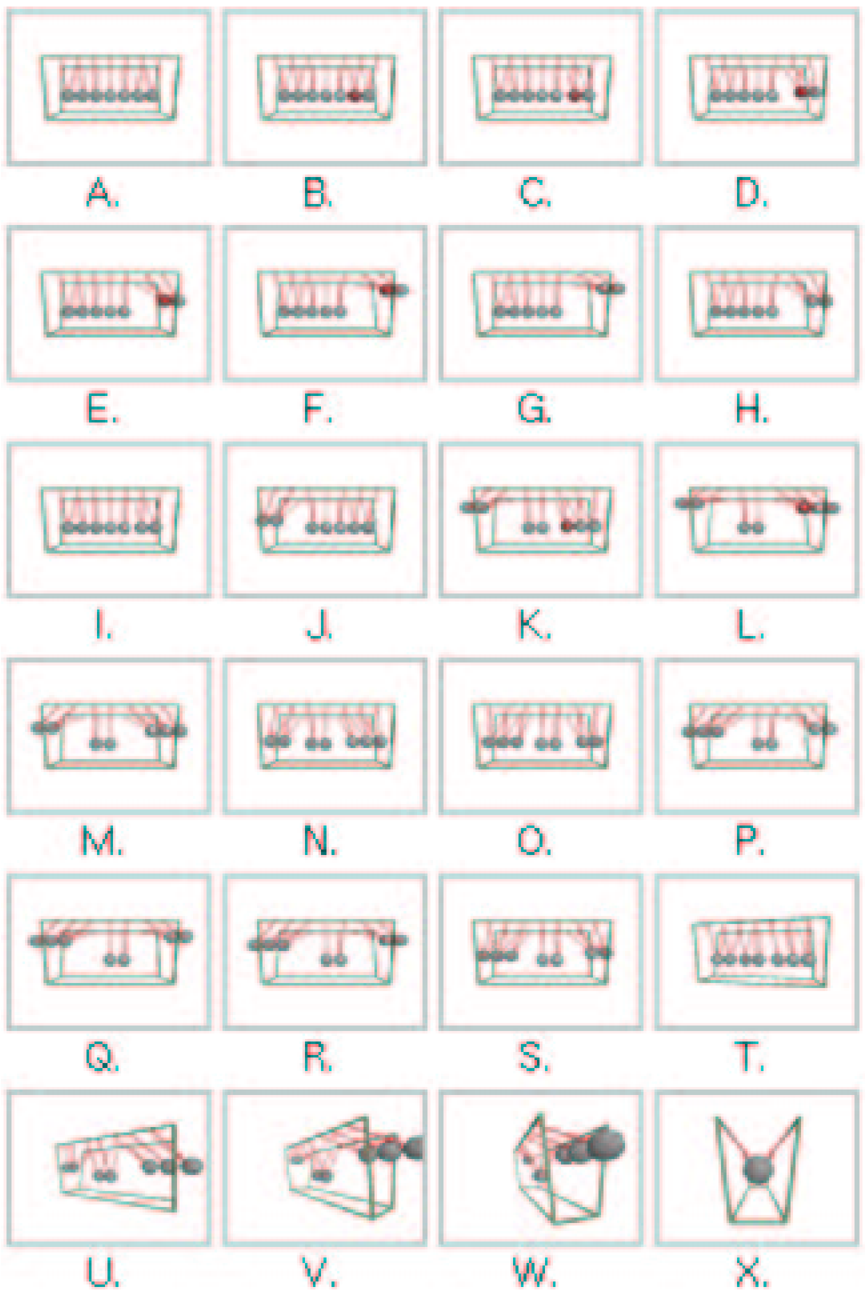

Figure 2: Interactive simulation of a Newton's cradle

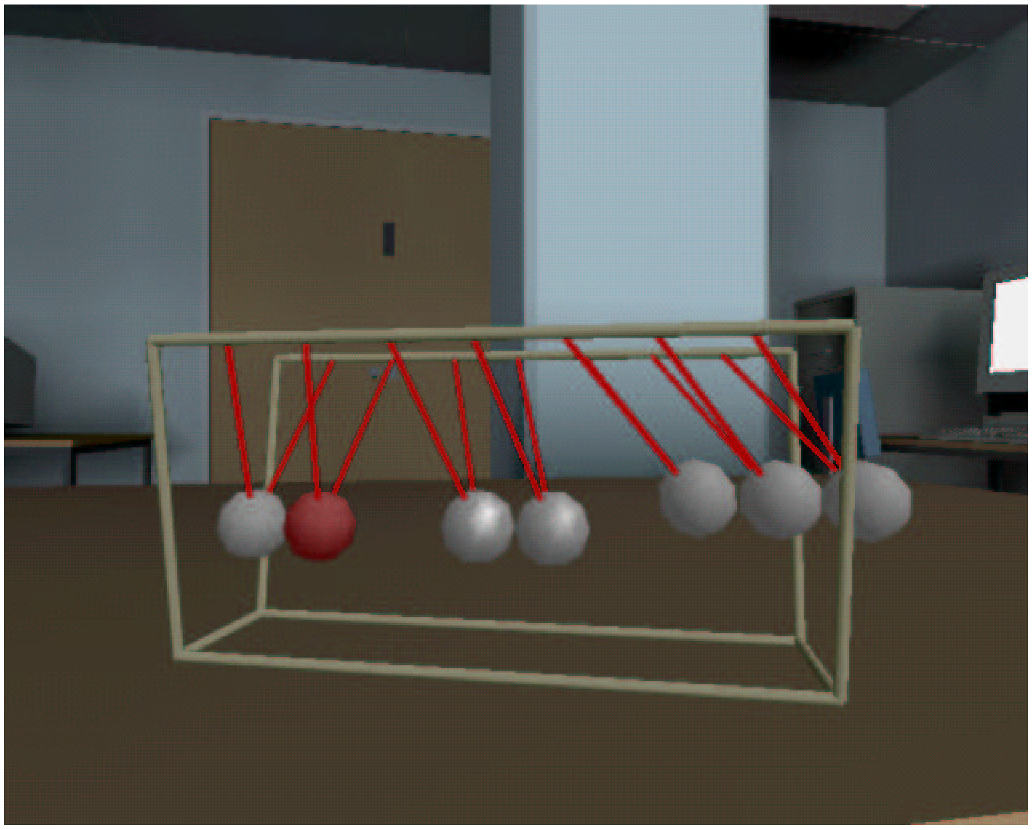

Figure 3: Interactive manipulation of a Newton’s cradle in the AIG laboratory