Discrete element modelling using a parallelised physics engine

DOI:
10.2312/LocalChapterEvents/TPCG/TPCG09/207-214

EG UK Theory and Practice of Computer Graphics (2009)
Wen Tang, John Collomosse (Editors)

Discrete Element Modelling Using a Parallelised Physics Engine

S.M. Longshaw*, M.J. Turner†, E. Finch⊥ & R. Gawthorpe⊥

*School of Computer Science, The University of Manchester, UK
†Research Computing Services, The University of Manchester, UK
⊥Basin Studies and Petroleum Geoscience, School of Earth, Atmospheric and Environmental Sciences, The University of Manchester, UK

Abstract
Discrete Element Modelling (DEM) is a technique used widely throughout science and engineering. It offers a convenient method with which to numerically simulate a system prone to developing discontinuities within its structure. Often the technique gets overlooked as designing and implementing a model on a scale large enough to be worthwhile can be both time consuming and require specialist programming skills. Currently there are a few notable efforts to produce homogenised software to allow researchers to quickly design and run DEMs with in excess of 1 million elements. However, these applications, while open source, are still complex in nature and require significant input from their original publishers in order for them to include new features as a researcher needs them. Recently software libraries notably from the computer gaming and graphics industries, known as physics engines, have emerged. These are designed specifically to calculate the physical movement and interaction of a system of independent rigid bodies. They provide conceptual equivalents of real world constructions with which an approximation of a realistic scenario can be quickly built. This paper presents a method to utilise the most notable of these engines, NVIDIAs PhysX, to produce a parallelised geological DEM capable of supporting in excess of a million elements.

Categories and Subject Descriptors (according to ACM CCS): I.6.8 [Types of Simulation]: Discrete Event I.6.8 [Types of Simulation]: Distributed I.6.8 [Types of Simulation]: Parallel

1. Introduction

The Discrete Element Modelling (DEM) technique is utilised widely throughout many subject areas in both industry and the theoretical sciences. It was initially proposed by Cundall [Cun71] and later presented practically by Cundall and Strack in their paper *"A discrete numerical model for granular assemblies"* [CS79]. The basic premise is that a collection of disparate bodies are allowed to physically interact over a set time period, while being contained in a specified area either by rigid static structures or opposing forces. The result is that the combined movement of each individual element produces an overall system, which can allow for accurate simulation of many material types.

DEMs are especially useful while modelling the effect of fracturing or any other form of discontinuity in a material. While the more common Finite Element Modelling (FEM) technique discretises an overall volume into a collection of points, each of which represents a portion of that volume, DEM is intrinsically discontinuous in its nature. It is more difficult to represent the absence of a material using FEM than with DEM, especially if that absence occurs during the models evolution. This discontinuous nature means DEM is the preferred solution for the analysis of rock and other fracturous materials. As with an FEM, a DEM with a greater number of individual elements will allow for higher model accuracy due to each element representing a smaller portion of the overall model, therefore increasing the models resolution. However, increasing the number of elements that exist in a DEM has a very obvious and detrimental effect to the amount of time required for processing.

One of the most computationally intensive portions of the majority of DEM codes is that of contact detection between each element. Typically, DEMs utilise spheres as elements, as spheres allow for simple contact detection while still offering reasonable representation of a portion of the model space. However, more complex models have begun to employ non spherical elements, increasing the computational complexity of collision detection. The techniques used to accelerate collision detection have improved rapidly. Initially, when DEM models tended to contain less than 10,000 elements and were in two dimensions, this operation was normally treated serially with each element being compared against every other, often using a simple Euclidean distance check. However as the models moved into three dimensions and the number of elements began to increase significantly, it was no longer feasible to perform the check in this way. Codes began to surface that utilised more advanced domain sub-division techniques, employing tree based algorithms from the world of computer graphics. At the same time, the increasing complexity of the models, with the inclusion of more real world physical properties has meant their computational demand has increased significantly.

This increase in computational complexity has led researchers to begin to distribute and parallelise their code over multiple cores. Typically this is done using common parallelisation techniques, utilising standardised libraries such as OpenMP [CMD*00] and MPI [WLS99]. This parallelisation, coupled with the ever increasing availability of large computing resources, has meant that the ability to calculate DEM models with in excess of 1,000,000 elements, in a reasonable timescale, has become a reality.

However, even with all of these advances, the use of DEM as a technique within some sciences that could be making use of its ability to model discontinuous systems, is relatively small. One key reason for this could be the mathematical inaccessibility of the DEM technique and the programming complexity of implementing fast collision detection. While there are a few commercial software packages that offer the ability to perform industrial DEM simulations [Sol09], they are not specifically aimed at the researcher. Recently there have also begun to emerge reasonable attempts at generic DEM codes that can be used to define and run large DEM models. The most notable of these are YADE [KD08] and ESyS-Particle (formally known as LSMEarth) [PM00]. ESyS-Particle offers good functionality and also the ability to parallelise the model over multiple CPUs using MPI.

Recently, software libraries designed specifically to compute the physical properties of objects as quickly as possible, have become available. These libraries are currently primarily aimed at the gaming industry, with closed source examples including NVIDIA's PhysX [NVI09] or Intel's Havok [Hav09] and some common open source examples including Bullet [Cou09] and ODE [Smi09]. As with graphical libraries such as OpenGL, these physics libraries are designed to allow the programmer to define complicated physical 'worlds' and then automatically perform all necessary calculations in as optimised a manner as possible, while still maintaining a reasonable level of accuracy. In order to ensure that calculations are performed as quickly as possible, the libraries employ complexity reduction techniques, combined with optimised tree based searching algorithms to calculate the positions and forces acting on each 'actor' contained within the 'world' at each time-step. The libraries allow for quick and relatively easy definition of complex physical environments, including non geometric triangulated objects. The libraries also allow for a large number of physical properties to be applied to each 'actor', resulting in the ability to produce complex DEM simulations in a relatively short period of time, without the need to re-implement physical calculation code.

Physics engines are especially suited to modelling scenarios which involve the usage of real-world physics. Unlike the majority of DEM code, which assumes each element is a sphere and bases the contact detection code around this fact, a physics engine provides a generic contact detection system. Due to this, the type of elements that can be used in a physics engine based DEM can be swapped quickly and easily from the standard sphere. Replacements can be as simple as a cube or as complex as a deformable meshed object. Perhaps even more compellingly, it is possible to easily mix interacting objects of varying types. Traditionally, DEM models tend to use a pre-defined set of elements, however when designing with a physics engine, the researcher is able to alter the physical properties of every single element in the system. From simple variations on size and mass through to dyanmically generating each element algorithmically.

However, physics engines, due to their target audience of game programmers, tend to employ strict limits on the number of physical 'actors' that can exist at any one time. Within NVIDIA's PhysX for example, it is currently only possible to introduce 64,000 individual 'actors'. Open source implementations can be altered to allow for larger simulations, but currently the result is that models become too slow to calculate in a reasonable timeframe and a more sensible option would be to use code specifically designed to calculate spherical DEMs.

This paper explores a set of methodologies that have been designed to allow a researcher to implement a large-scale physical modelling scenario using a modern physics engine. The methods presented allow distributed parallelism to be utilised to overcome the limitations of simulation scale that current real-time physics engines enforce. The methods themselves are applicable to most current libraries and have scope to be included as part of the feature set of future libraries.

2. Related Work

Within Earth sciences, it is typical to perform scaled down modelling experiments using physical media such as sand or clay. The 'sandbox' experimental apparatus is often used by researchers to study the effects of folding and faulting in rock. This has allowed for a reasonable approximation of the structure of rock to be created using particulate materials such as sand, and friction inducing components such as rubber sheeting, within an enclosed space. Through steady compression or expansion of the enclosing space, the particulate material moves in a similar manner to rock, only over a much smaller scale and time period.

While the results of this form of analogue experiment are undoubtedly invaluable, numerical modelling using DEM offers an obvious alternative with many advantages. A numerical model is not restricted by the concept of scale; similarly, it is possible to make each element in a DEM mimic the physical properties of the rock that is being simulated much more closely. When using a DEM the researcher is also able to begin to consider model parameters that simply would not be possible using real world experimentation, specifically in the case of crustal fault evolution experimentation, the use of a breakable spring/damper connection between every element in the system to make it behave more like a fracturous continuum.

There is precedent for using DEM for fault analysis, with examples of two dimensional code available [FHG03] and a solution presented by Mora and Place in which they connect the elements together using a spring-damper structure. In their paper entitled *"A lattice solid model for the non-linear dynamics of earthquakes"* [MP93], they present the use of a lattice of spring/dampers to produce an initially continuous DEM that is able to evolve to include fracturous regions. This work led to a software implementation known as LSMEarth [PM00], which provided the ability to produce three dimensional DEM models based around a set of generalised input parameters. LSMEarth [PM00] has eventually evolved to become ESyS-Particle, which could be considered to be a universal parallelised spherical DEM calculation code.

3. Physics Engines

Many different physics engines exist and most are designed in a similar manner. Most engines use the object-orientated programming paradigm, with C^{++} being the preferred choice. Typically they use tree based domain decomposition and catagorisation techniques to allow for fast yet accurate calculation of physical interaction and movement of a collection of bodies in a system. Generally they are used to calculate user defined fixed timesteps and allow for variable sub-division of each timestep in order to increase simulation accuracy at the expense of computatiion time.

As with certain tasks within computer graphics, the calculation of Newtonian physics is a well defined problem. Currently, it is normal for code to calculate physics to be re-implemented and thus re-written by each researcher every time a new model is designed. Within Computer Science, it is generally accepted that it is best practice to standardise the code for repeat calculations, thus ensuring that all code has an equal starting point. Within graphics, this process was begun though the development of API libraries, which perform the most common calculations undertaken in the generation of graphical geometry. Today we primarily use either OpenGL or Microsoft's DirectX libraries. This has the effect that improvement and refinement of the graphical calculation code is handled by those that develop the graphics APIs and all other software is built upon that code base. Physics engines offer this same possibility to researchers developing physical simulation code.

While the current generation of physics engines may not be suitable for all physical simulations, there are certainly cases where the calculable drop in integration accuracy can be acceptable, while the ease of implementation and diversity offered is sufficiently compelling to make a physics engine a realistic choice. One such area is that of a DEM on a geological scale. Models of crustal deformation take place over millions of simulated years; often involve limited overall movement and include very large conceptual scales. Typically the precise movement of each element in such a DEM is not as important as the general trend of movement in the system.

3.1. NVIDIA PhysX for Geological Modelling

Physics engines offer the ability to define a simulated world using a collection of different shapes and joint types. A shape can be defined as either a primitive geometric shape or as a customised triangulated mesh. Shapes in the world are commonly referred to as actors, however an actor can actually be a compound of multiple shapes. One of the acceleration techniques the engines employ is by determining whether a shape is ever going to move during a simulation or not. To this end, an actor can be considered to be fully dynamic, entirely static, or in a state in-between known as kinematic. Static actors never move, dynamic actors may move at every time-step and kinematic actors exist in the world as dynamic actors, only their movement is fixed unless a specific command is issued telling them to do so.

One of the most highly developed and well documented physics engines is NVIDIA's PhysX [NVI09]. It is free for commercial and non commercial use, but closed source with the option of paying to access the source. A recent study [BB07] has shown the accuracy of the PhysX integrator to be closest to that of standard Verlet or Euler integration schemes. While its accuracy for other physical properties such as friction, restitution and angular velocity are all closer to the idealised normal than most other engines. Only its joint solver lacks behind other engines and this is currently an area of development within NVIDIA. The overall calculation performance of PhysX appears to be superior to most engines in most situations.

However, there are limitations to the size of the simulation physics engines allow. These vary per engine; however PhysX employs the following limits:

- Maximum number of scenes per world: 64
- Maximum number of shapes per scene: 64,000
- Maximum number of contact pairings per scene: 256,000

If we assume a simple 3D DEM case, containing only equally sized spheres, then each sphere is theoretically able to come into contact with 12 others. This means that the 256,000 contact pairings limit per scene in fact reduces the maximum spheres that can exist safely per scene to 21,333.

In the next section methods to exploit the limitations imposed by PhysX as a basis for a parallelisation model are explored.

3.2. Parallelisation of Physics Engines

As has been previously discussed, to make a physics engine usable on the scale required for DEMs, while maintaining computational performance, it is essential to exploit parallelism. A single instance of a physics engine has a hard limit on the number of actors it supports, in some engines this can be as high as 1,000,000 in a single scene. PhysX provides an extra layer of sub-division in that it can support multiple scenes within a single instance of the engine; this therefore raises the possible number of actors to 4,096,000 as it is possible to create 64 scenes, each of which can hold 64,000 actors. However, the actors in one scene do not interact with actors in another, effectively meaning that a PhysX world containing 4,096,000 actors would in fact contain 64 individual DEM models, each with 64,000 elements with no awareness of the movement of elements in the other 63 models.

To overcome this limitation, and allow simulations bounded only by system resources, while still maintaining the use of a black box physics engine, a form of inter-model interaction has been defined to suit the nature of modern physics engines. The concept is similar to domain subdivision, in which the elements at the edge of a predefined sub-boundary within the model are recreated within the next boundary. Elemental overlap of the boundaries is then resolved by allowing an overlap of the domains equal to the size of the largest element in the model. Inside this overlapping region, the element may exist twice.

The model developed to allow for the parallelisation of multiple instances of a physics engine operates in a subtly different manner. Effectively each instance of the engine receives a portion of the overall number of elements; the maximum number that can be added to each instance is limited by the engine itself, henceforth an instance is referred to as a model. Dynamically adjusted bounds are then calculated for each individual set of elements within each model, at each timestep. Elements at the edge of these bounds are tested against the elements within any other model approaching the current models bounds. Where it is found an element is theoretically about to come into contact with an element within the approaching scene, a 'ghost' element is created at the exact location of the tested element, but within the approaching scene. 'Ghost' elements have the same physical presence as the element they represent and are subject to forces from ele-

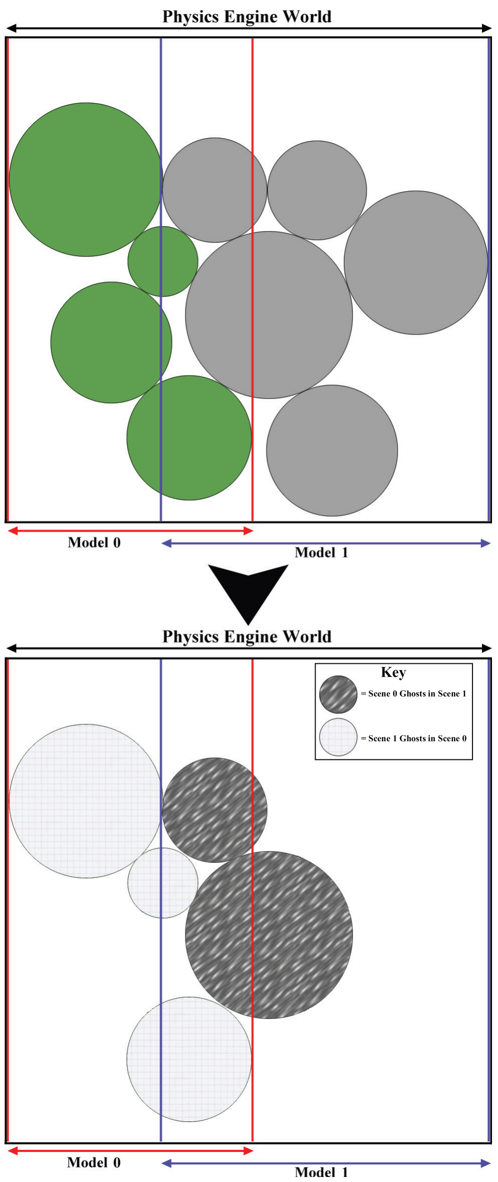

Figure 1: *Overview of the element ghosting model used to allow multiple DEMs to interact as one. The red boundary lines show the dynamically adjusted bounding area of model 0, while the blue lines show the bounding area of model 1. Model elements are shown in the top portion while the resultant 'ghost' elements are shown at the bottom.*

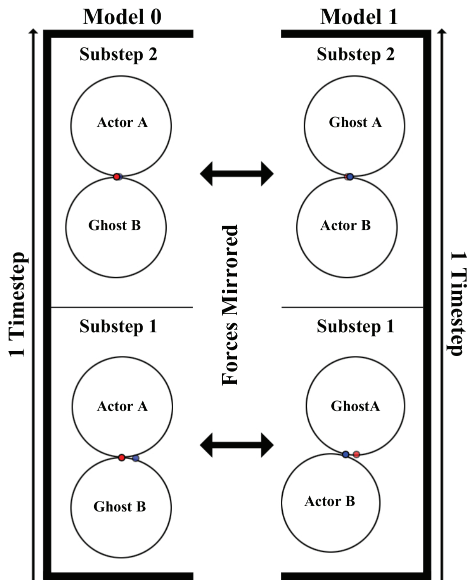

Figure 2: *Simplification of force mirroring schema for each substep of an overall timestep between two scenes in a parallelised physics engine based DEM*

ments in the scene in which they reside. After each time-step, any force that they have experienced is mirrored to the element that they represent. Similarly, any forces on their parent element is mirrored to the ghost. If an existing ‘ghost’ is found to be no longer required, it is deleted. This system is illustrated in figures 1 and 2.

The forces that are copied are obtained from the physics engine directly at each iteration of a substep. Forces are introduced to all bodies using a specific position rather than at their centre of mass, thus ensuring the resultant torque introduced is as accurate as the engine initially calculated. A requirement of this method is that it is possible to identify each element and be able to define whether the interaction pair should be mirrored or not, for example we do not wish to mirror forces generated between two ghosts, rather only between ghosts and normal elements. Some engines, such as PhysX, offer a built-in bit-masking system, allowing for quick and easy segmentation of each element in the system, however for engines without this ability, it would be necessery to manually identify between which two actor types a contact has occured.

While the ghosting model allows multiple DEMs to operate independently but still acting as a continuous system, it is important to then split the overall DEM into its smaller sub models according to the limitations of the engine being used. Within PhysX, it is possible to add 21,333 same sized spherical elements to each scene (due to the pair contact limit), of which there can be 64 within each instance of the engine. This means we are able to create a model with 1,365,312 elements using one instance of the PhysX engine.

A better model however is to distribute over a multi-nodal computing cluster, utilising parallelism at both the local and distributed levels. If, for example, we have a 1,000,000 element model and have access to 8 compute nodes, each with 8 local processing cores, then we can create 8 instances of the PhysX engine, 1 on each node, and then create 8 scenes within each node. This would mean we have divided the 1,000,000 element model into 64 models, each with 15,625 elements. This can be seen in figure 3.

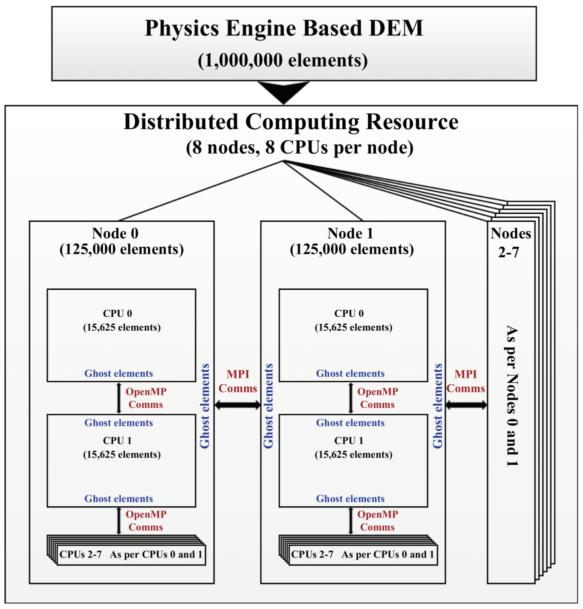

Figure 3: *Example of a parallel distribution of a physics engine based DEM. This model assumes an MPI scheduler that allows fine grained parallelism using OpenMP or local threads. In the absence of this capability, the OpenMP communication channels are simply replaced by MPI or the available equivalent and the distributed resource considered on a per CPU basis rather than a per node basis. The model shown in the figure is optimal.*

Clearly if we have enough processors to ensure each model contains less than the maximum number of elements supported by each PhysX scene, then we have an optimal case in which each model is operating on its own processor. If the overall number of elements is increased to 2,000,000 then each node would contain 250,000 elements and we would need 14 scenes to be calculated on each node, assuming we allow for a 10% reduction in elements per scene to allow for ghosts to be created. While this is still highly parallelised, it is not as optimal as with 1,000,000 elements, as each processor effectively has the task of solving 1.75 models.

One interesting outcome of this form of parallelisation is that it is possible to guarantee determinism of the model. When traditional parallelisation techniques are applied, such as for loop splitting, the allocation of work from the for loop is often different on each run, depending on multiple factors such as the work load of each processor at that exact moment. If a loop were to contain a calculation which relied on any other iteration of that loop, then it may be that on one run, due to floating point rounding errors, a slightly different result is obtained. The parallelisation technique presented here ensures that each subset of elements of the overall model is only ever calculated by one processor. Therefore, as long as a run is performed on the same hardware, therefore ensuring the model is split in the same fashion and floating point operation is the same, the results from one run will be identical to another.

4. Example Geological Case Study

In order to demonstrate the parallel usage of a physics engine in the design of a DEM, this section presents results from a partial run of a DEM representing a weak rock structure approximately 100km in length, 60km in depth and 30km in height. The DEM presented is relatively coarse; containing 97,038 equally sized spherical elements. It was run on a single PC containing 2 AMD Athlon FX-70 CPUs, providing a total of 4 processing cores. The overall model was split into 6 individual models, each containing 16,173 elements. Therefore this was not an optimal parallelisation case.

The model consisted of a simple triangulated box 115km in length, 60km in depth and 60km in height. The elements were initially packed into the box in a close hexagonal formation. Within the box, at 100km, a convex hull in the shape of a flat plate acted as an immovable barrier between the free space in the box and the elements packed to the 100km point. The plate was designated as a 'kinematic' actor; therefore it could interact with the model elements but its movement could only be affected by direct manipulation of its position. Each element in the model began by being attached to all of its neighbours by a weak spring-damper joint; each joint in the system had a slightly randomised breaking length but identical spring and damping values. Gravity was set to -9.81m/s. The box, plate and all elements had negligible values for friction and each element had a density of 3,300 kg/m^{-3}.

Images taken from the 120th iteration of the model can be seen in figures 4 and 5, this is equivalent to 1200 years of model evolution. The kinematic plate has been moved at each iteration so that it provides a smooth compressional force against all of the elements on the rightmost side of the model.

The primary parallelisation overhead that this method introduces is the dynamic creation and deletion of ghost elements, as well as transferring the data required to re-position

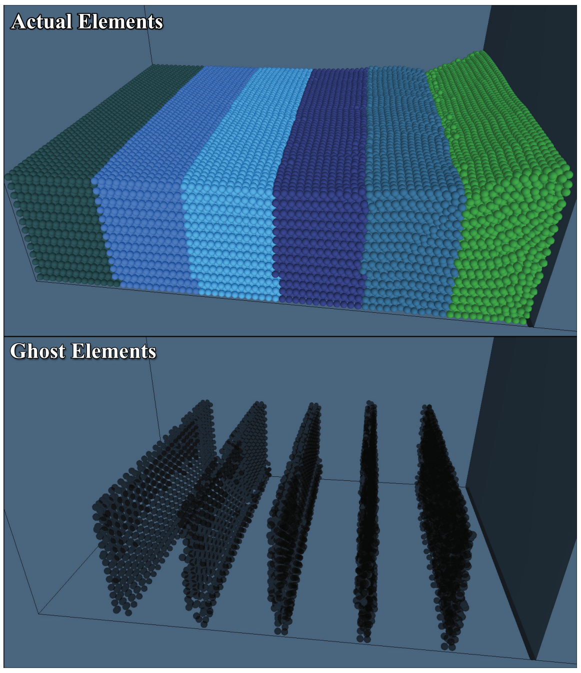

Figure 4: *Geological DEM model at t=1200 years. Actual model elements can be seen at the top of the figure; differing colours represent each elements membership to a particular sub model. The resultant 'ghost' elements that allow inter-model interactivity can be seen at the bottom of the figure. Compressional force is being applied from the right of the model to the left.*

each ghost after every iteration of the model. Detailed research to obtain definitive results as to how the overhead scales with the size of the DEMs are yet to be determined, however for this test case, at t=1200, there existed 3458 ghost elements in total. This is a 3.56% increase in the number of elements that exist in the system over the initial amount.

As this example occurred within a single shared memory system, each sub-model utilised a shared memory resource to access every other sub-models boundaries and ghosted elements. In a scenario utilising MPI for parallelism over a nodal computing cluster, there will be a further overhead in the form of a broadcast of the sub-boundary and force details. Given a 100,000 element model and a distributed system utilising a modern high bandwidth interconnect, it is estimated that this transfer overhead will be negligible.

As one sub-model encroaches further into another, this overhead will increase, with a worst case scenario being that one sub-models boundary entirely surrounds another, therefore meaning each element in each model will likely be tested against a theoretical maximum number of elements, with all elements haveing a ghost. Effectively each model

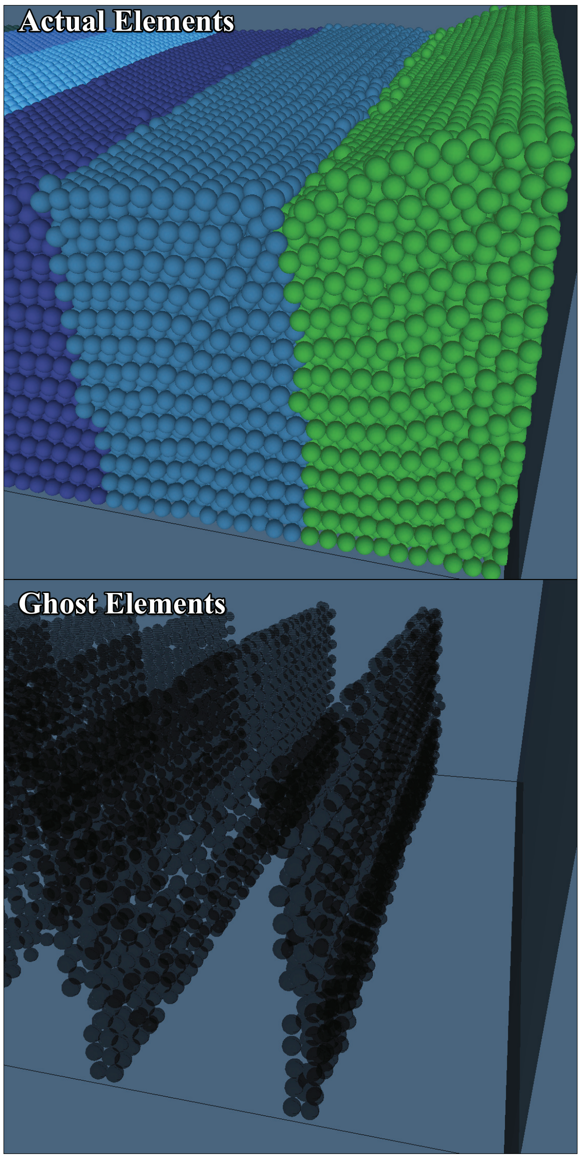

Figure 5: *Close-up of geological DEM model at t=1200 years. Actual model elements can be seen at the top of the figure; differing colours represent an elements membership to a particular sub model. The resultant ghost elements can be seen at the bottom of the figure. Of particular interest in this figure is the accurate continuation of movement between the rightmost scene, with green elements, and the next scene to the left, with grey elements. This movement occurs purely due to the ghost element structure.*

would become mirrored in the other. Due to the scale of geological DEMs, this case is unlikely to happen, with model overlap being kept close to each sub models boundaries. However there may be some uses for the DEM technique in which this worst case is likely and therefore should become a consideration.

5. Conclusion

This paper has presented a method to allow modern physics engines to be utilised in the design and implementation of large scale scientific Discrete Element Models and other suitable simulations. The implications of the reduction in numerical accuracy that a physics engine will introduce to a DEM have been discussed as have the usage limitations that the most notable engines currently enforce.

The benefits of using a physics engine for scientific model development, especially for those models that rely on physical interaction between different bodies, are numerous. Physics engines provide a robust and well defined framework with which to build a simulated version of a real world scenario. This paper has shown how the NVIDIA PhysX engine can be utilised to implement a large scale geological DEM, based loosely around the Lattice Solid Model [MP93].

In order to utilise PhysX for DEMs with large numbers of elements, it was necessary to define a method to use multiple instances of the engine to calculate an overall model. This paper presents the design of a model which allows multiple discrete DEMs, that utilise a physics engine, to be calculated simultaneously across large distributed computing systems. The design is currently only being used for geological DEM purposes, but with further testing should be suitable for other modelling scenarios where a physics engine can be used.

The technique of splitting a DEM into multiple individual DEMs, as seen in this paper, allows a limited black box physics engine to be used to produce large-scale scientific simulations. The effects of the ghosting model on general accuracy needs further exploration, as does the extent of the overhead introduced when a model is parallelised over a large distributed computing cluster.

Within Computer Science, it is generally accepted that producing a standardised framework of code for the most common calculations is good practice. The area of computer graphics already has two main standardised libraries in the form of OpenGL and DirectX, which are accepted and utilised for most rendering purposes. However, physics calculations are normally re-coded every time they are required. This leaves new simulation code open to individual programming errors and discrepancies. While the current generation of physics engines, due to their bias towards game design, are perhaps not numerically accurate enough for some simulations, the basic premise of a highly accurate homogenised library of physical constructs, with which a researcher can produce a simulated real-world, is a compelling one which may be explored by the current commercial engines in the future and is already being explored by the open source developers of engines like Bullet [Cou09].

The goal of a physics engine is to provide the developer with a generic set of joints and collision detection routines, so that they are able to produce a physical scenario that can be likened to any real-world scenario. This not only simplifies the process of handling collision detection between many uniform and non-uniform rigid and soft bodies, it also allows complicated jointing systems. As physics engines improve in performance and accuracy, the possibility of easily allowing a researcher, to design and produce their own high performance simulation code becomes a reality. The integration of the ghosting model presented by this paper would also allow models created using the engines to be processed in parallel over clustered computing resources.

6. Future Work

While the techniques presented in this paper are currently being used to produce large DEMs of fault evolution in the Earth's crust, they should be applicable to other forms of physical modelling.

Immediate work to be completed involves analysis of the ghosting technique, measuring the overhead that it introduces, assessing its applicability to physics engines other than NVIDIAs PhysX, and determining any reduction in calculation error that utilising such a technique introduces into very large systems over a large simulated time period. Also, there currently exists a situaton where too many ghosts may be introduced into a model, with a possible worst case scenario of there being as many ghosts as there are elements. While this is unlikely to happen during the evolution of a geological DEM due to the slow and relatively small movement in the system, if the method was utilised to design a simulation for another purpose, it may become a tangible issue.

Further application of the method will be used to produce large DEMs of fault evolution, the capabilities of the PhysX engine will be explored, including utilising non-geometrical, or even deformable rigid bodies in place of the current spheres. Replacement of the simple rigid box structure that holds the elements in place will also be examined. Use of existing techniques found within PhysX, such as variable force fields and complex triangulated convex hulls may also be explored.

As a physics engine effectively provides a closed loop within simulation code where the majority of the simulation calculation occurs, it also becomes easier to design model code that involve forms of computational steering, that is allowing model parameters to be altered by the user as it evolves, while also offering a form of visual or numerical feedback informing the user as to the current state of their simulation. The inclusion of steerable parameters into a simulation using a physics engine will be explored, including how user input should be propagated across multiple models all running simultaneously and how best to offer visual feedback.

References

[BB07] BOEING A., BRÄUNL T.: Evaluation of real-time physics simulation systems. In *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia* (2007), ACM, pp. 281–288.

[CMD*00] CHANDRA R., MENON R., DAGUM L., KOHR D., MAYDAN D., MCDONALD J.: *Parallel Programming in OpenMP*. Morgan Kaufmann, 2000.

[Cou09] COUMANS E.: Bullet. http://www.bulletphysics.com, 2009.

[CS79] CUNDALL P. A., STRACK O. D. L.: A discrete numerical model for granular assemblies. *Geotechnique 29* (1979), 47–65.

[Cun71] CUNDALL P. A.: A Computer Model for Simulating Progressive Large Scale Movements in Blocky Rock Systems. In *Proc. Sympo. Int. Soc. Rock Mech.* (1971), pp. 129–136.

[FHG03] FINCH E., HARDY S., GAWTHORPE R.: Discrete element modelling of contractional fault-propagation folding above rigid basement fault blocks. *Journal of Structural Geology 25*, 4 (2003), 515–528.

[Hav09] HAVOK: Havok physics. http://www.havok.com [, 2009.

[KD08] KOZICKI J., DONZÉ F. V.: A new open-source software developed for numerical simulations using discrete modeling methods. *Computer Methods in Applied Mechanics and Engineering 197* (2008), 4429–4443.

[MP93] MORA P., PLACE D.: A lattice solid model for the non-linear dynamics of earthquakes. *International Journal of Modern Physics C 4*, 6 (1993), 1059–1074.

[NVI09] NVIDIA: Nvidia physx. http://www.nvidia.com/object/nvidia_physx.html, 2009.

[PM00] PLACE D. G., MORA P. R.: LSMearth: a Virtual Earth Simulator for eathquake micro-physics. In *2000 Fall Meeting, American Geophysical Union* (2000), American Geophysical Union, pp. 15–19.

[Smi09] SMITH R.: Open Dynamics Engine (ODE). http://www.ode.org, 2009.

[Sol09] SOLUTIONS D.: Edem. http://www.dem-solutions.com, 2009.

[WLS99] WILLIAM G., LUSK E., SKJELLUM A.: *Using MPI, 2nd Edition: Portable Parallel Programming with the Message Passing Interface*. MIT Press, 1999.