Technical Report DIKU-TR-02/06
Department of Computer Science
University of Copenhagen
Universitetsparken 1
DK-2100 KBH Ø
DENMARK

July 2002

Module Based Design for Rigid Body Simulators

Kenny Erleben

Abstract: In this paper we outline a module design for a rigid body simualtor. The purpose of the module design is to clarify how many parts a simualtor consist of and how these parts work together. The module design can be used as a guideline for how one should implement a simulator. The paper contributes to the field of dynamic simulation by introducing the concepts of a time control module, a motion solver and a contact analysis module.

Contents

1 Introduction

It is widely accepted that implementing a rigid body simulator is both a difficult and time consuming task with a steep learning curve. Why is this so? Many people believe it is due to the large amount of mathematics and physics which one has to learn. However we ourselves have a firm background in mathematics and physics and still experienced the same problems when implementing our first simulator several years ago. This is why we believe that the mathematics and physics are only partly to blame. In our opinion there are two more important reasons.

- Rigid body simulation (and dynamic simulation in general) covers a wide range of reaserch areas: Algorithms, graphics, computational geometry, numerical methods etc. Meaning that a newcomer not only has to learn a lot of mathematics and physics, but all around knowledge of computer science is also required.
- Most of the litterature is written as articles describing a small part of the functionality of a simulator, such as a single algorithm or data structure. Eventhough most of this litterature is well written and fairly easy to understand, it is not always easy as a newcomer to figure out how the bits and pieces in an article fit into the "big picture".

In this article we are going to look at the "big picture", that is the glue which binds all the smaller pieces into a large simulator. Our hope is that this can guide newcomers in a way that makes it easier for them to gain an overview of how all the theory fits together and implement their first simulator.

A rigid body simulator is often broken down into smaller pieces, each responsible for handling a specific task in the simulator, e.g. computing a collision impulse, determing the time of impact and so on. In this paper we call such small pieces modules. There is really nothing new in breaking down a simulator into modules and many thesis's and articles have been written explaining what the modules of specific simulators looks like. However not much work has been done towards presenting an unified general purpose module design. For several years we have worked on such a unified genral purpose module design [92][1]. Over the years we have continued our work and our early module design has now evolved into a more mature and phedagogical version with several improvements. In this paper we will present this latest version of our module design.

The paper is organized into two parts, the first part takes the reader on a detailed tour through a single frame computation. The last part of the paper discusses the variations on the module design depending on the different kinds of simulator paradims.

The paper is intended for people that are relatively new to rigid body simulation, perhaps perfect for those who are just about to start on implementing their very own simulator for the first time.

2 A Module Design

At the highest level the simulator appears to be split into two components, a collision detection component and a simulation component as you can see on Figure 1. This section outlines the general purpose module design by walking through a single frame computation and explain the modules as we encounter them in the frame computation.

2.1 The Simulation Component

Our tour begin at the simulation component, which consists of four modules: A collision solver module, a time control module, a motion solver module and a constraint solver module (see Figure 2).

- Collision Solver Module
- Time Control Module

[1]The result was a module design known as **QAD**, which is an acronym for **Q**uick **A**nd **D**irty.

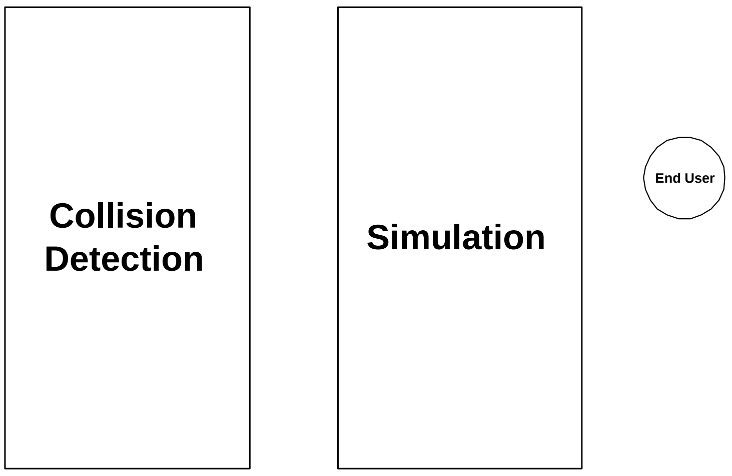

Figure 1: At the top level a simulator consist of two components.

- Motion Solver Module
- Constraint SolverModule

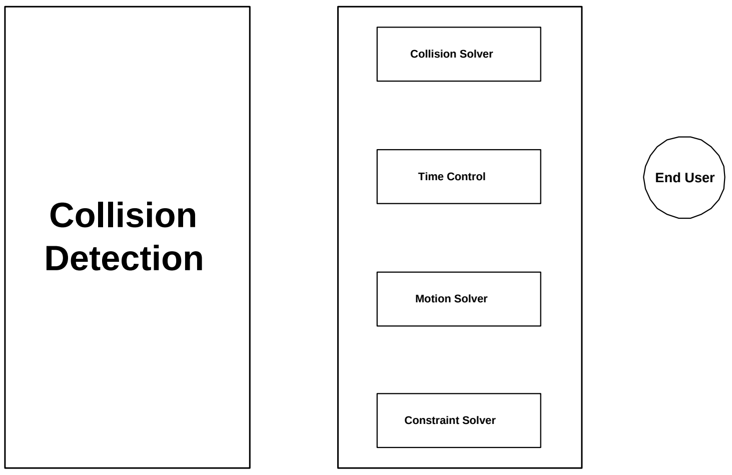

Figure 2: The four modules of the simulation component.

2.1.1 The Time Control Module

The time control module is the central part of the simulator. This module controls when and how all the other modules in the simulator should be invoked. Our frame computation starts out by the end user tells the time control to simulate forward in time from the current time, t_{cur}, to $t_{next} = t_{cur} + \Delta t$. The configuration state at time t_{next} is then returned as the computed frame (see Figure 3).

The time control module can use fixed-time-stepping and/or adaptive-time-stepping algorithms to simulate forward to t_{next}. The later group of algorithms is further divided into two subgroups called backtracking and one-sided-approach.

- Fixed-time-stepping

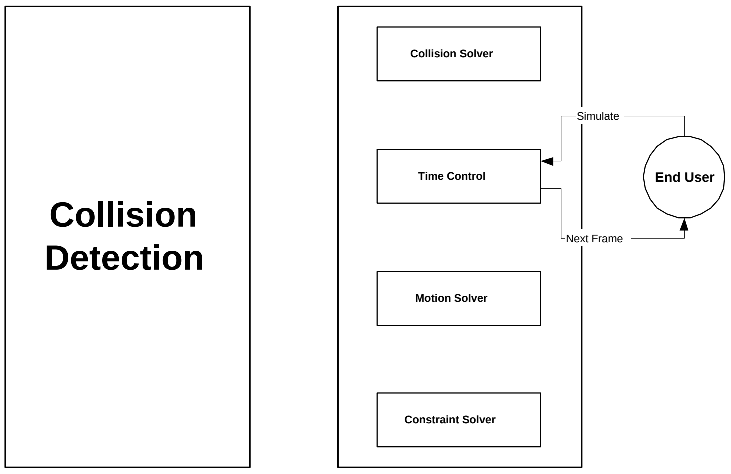

Figure 3: The end users interaction with the simulator.

- Adaptive-time-stepping
  - Backtracking
  - One-sided-approach

What kind of algorithm that is used dependens on the sort of simulator paradigm[2] one has picked. The time control interacts with the motion solver and the collision solver, however the interaction varies depending on the sort of simulator paradigm and time control algorithm one has picked. For now we will overlook the implications of the simulator paradigms and instead shortly review the main ideas of the three algorithm types.

With fixed-time-stepping (see [55, 53, 56, 57]) the time control module would blindly ask the motion solver module to simulate forward by some small fixed stepsize, $t_{fixed} \ll \Delta t$, until t_{next} is reached. It is quite obvious that with fixed-time-stepping both deep penetrations and overshooting[3] could occur if t_{fixed} is not picked small enough.

The ideas of both backtracking and the one-sided-approach stems from traditional root search algorithms. The idea is to search for the "root", which corresponds to the point in time where the objects in the configuration first touch each other. Both algorithms make use of the collision detection component to do this.

The backtracking algorithm (see [60, 82]) will ask the motion solver to simulate forward some timestep, then it will ask the collision detection whatever a penetration occured. If a penetration occured then the time control has to ask the motion solver to backtrack to the last known "good" state afterwards the backtracking algorithm will try to simulate forward once again, but this time with a smaller stepsize. The backtracking algorithm continues to do so until it reaches a state where the objects either are totally separated or are in touching contact. It will then ask the collision solver to handle any collisions and afterwards it will repeat the entire process until it reaches the time t_{next}.

The one-sided-approach works a little different (see [75, 11, 16, 33]), the main idea is that no penetrations are allowed instead the point of contact is reached by moving the objects closer and closer until they touch. The one-sided-approach does so by computing a lower estimate for the earlist time of impact (TOI). In order to compute these TOIs the time control has to ask the collision detection for the closest distances between all objects in near proximity of each other. Knowing the velocities and accelerations of the objects one can compute a lower conservative estimate for the time of impacts. Afterwards the time control picks the smallest TOI and ask the motion solver to simulate forward to the time of impact.

[2]A simulator paradigm means a certain way of doing simulation. It will be elaborated in section 3.

[3]Also called tunneling, it basically means that objects fly through each other without detecting a collision.

Afterwards the collision solver is invoked to handle any collisions and the patteren repeats itself until the time t_{next} is reached.

The main thing to take notice of is that the frame computation happens through several iterations of the time control. The states that are computed before the time t_{next} is reached are frequently called inbetween states. Figure 4 illustrates the interaction between the time control module and the other parts of the simulator. As a final remark we should mention that it is possible to make hybrid time control algorithms (see [60]), this basically means that one has a time control module, which shifts time control algorithm depending on different criteria. There are also other new tendencies remindscient of distributed algorithms (see [91]).

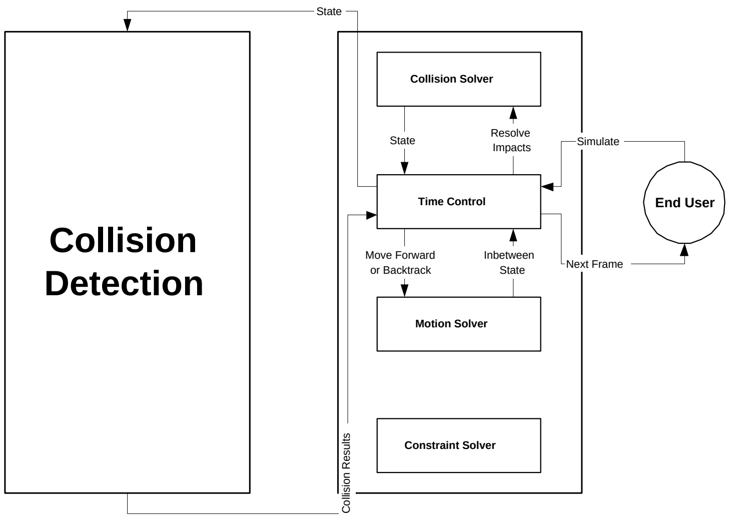

Figure 4: The time controls interaction with the other modules.

2.1.2 The Motion Solver Module

Now let us look at the next module which we encounter in our frame computation, this is the motion solver module. The motion solver is responsible for taking care of the continous movement of all objects in the configuration. Basically there are two ways to describe the movement, either by an ordinary differential equation (ODE) (see [94]):

$$\frac{d}{dt}\begin{bmatrix} \vec{r} \\ q \\ \vec{P} \\ \vec{L} \end{bmatrix} = \begin{bmatrix} \vec{v} \\ \frac{1}{2}\left[0,\vec{\omega}\right] q \\ \vec{F} \\ \vec{\tau} \end{bmatrix}$$

Or by a scripted motion of some sort (see [75, 95]):

$$Y(t) = \{\vec{r}, q, \vec{v}, \vec{\omega}, \vec{a}, \vec{\alpha}\}$$

Both describes the state of a single object. When the time control request the motion solver to move the objects forward in time then the motion solver has to use an ODE solver of some kind to compute the new state of those objects who motion is described by an ODE. For each integration step it will have to first compute the new state of the scripted bodies and afterwards all external forces and finally any constraint forces working on the objects. In order to compute the constraint forces, the motion solver sends the current (inbetween) state of the configuration to the constraint solver, the constraint solver then computes any constraint forces and returns them to the motion solver, which apply them to the ODEs.

It is important to notice the order of computation in each integration step. Constraint forces can depend both on external forces and scripted bodies. Finally scripted bodies are totally independent of any kind of forces.

Typically the motion solver will track (record) all the inbetween states (see [75] as an example). This state information could be valuable for the time control module. One usage is to validate TOI limits, to our knowledge there does not today exist a method for computing conservative TOIs for multibodies, so by using the state tracking information the time control can check whatever the TOIs previously computed were violated or not.

Recall that the time control might also tell the motion solver to backtrack, that is reestablish the previous state. This means that the motion solver also have to store state information for backtracking purpose.

By now it should be apparent that there is a two-loop structure in a single frame computation: The outer loop, which consist of the time control iterations and the inner loop which consist of the ODE integration steps of the motion solver.

Many newcomers typically ask about how big the integration steps should be? This is of course highly dependent on both the configuration and the accuracy one wants. In our experience integration steps typically goes as low as $\frac{1}{100} - \frac{1}{1000} sec.$ and in some difficult cases even lower. So if real time simulation should be possible the inner loop should be run at a rate of 1000 times per second. This sugest a place where one can look for a performance bottleneck.

Now let us summarize some of the facts we have learned. The motion solver:

- is responsible for computing the configuration state.
- contains state tracking information for the last time control iteration.

Figure 5 shows how the motion solver module interacts with other modules in the simulator.

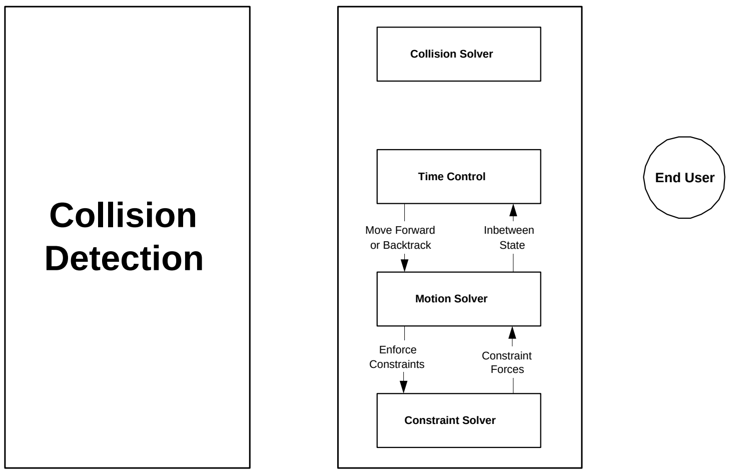

Figure 5: The interaction between the motion solver and the other modules.

2.1.3 The Constraint Solver Module

The motion solver constantly needs to invoke the constraint solver in order to compute the constraint forces that is needed in the ODE's that describes the objects contineous motion. Constraint forces are needed in order to prevent objects from interpenetrating each other when they rest upon each other, these kind of forces are known as contact forces (see [87, 72, 84, 76, 71, 70, 93, 54, 81, 77, 82, 65, 94]). In order to compute the contact forces the constraint solver has to query the collision detection for the contact regions between the objects. Links between objects could also be modelled by constraint forces.

The main difference is that these sort of constraint forces are bilateral whereas the contact forces are unilateral.

Today there exist two different approaches to compute these constraint forces: analytically or by penalty methods. In the first case a system of equations and constraints are set up and solved. Penalty methods works by adding penalty forces where interpenetrations occur (notice that penetrations are allowed with penalty methods).

- Analytical
- Penalty Based

There are numerous variations on how and when the constraint solver is invoked by the motion solver:

Eager: Invoke collision detection and recompute constraint forces for each ODE integration step.

Partly Lazy: Run only collision detection for the first ODE integration step and use the computed contact regions for the constraint force computation in all succeding ODE integration steps.

Lazy: Compute only constraint forces for the first ODE integration step and reuse the same constraint forces in the succeding ODE integration steps.

One could of course ask the question, which one of these methods are the correct way of doing things? In our opion it is highly dependent on the usage of the simulator. If very accurate simulation is required then the first strategy would be preferred, unfortunately it is quite expensive mostly due to running the collision detection in every integration step and real time simulation can be hard to obtain.

The other variations are computationally more tractable but also more likely to produce wrong simulation results, because the contact regions can change tremendously during a contineous movement of an object and constraint forces will therefore also change. Imagine a high speed ball rolling of a table top. If the lazy variation is applied the normal force from the table top is applied to the ball even when it no longer touches the table top. However if the eager variation is used then the contact region change will be detected and the ball will drop under gravity when it no longer touches the table top. We prefer the eager variation since the simulation results are more accurate. However many people use the other two variations in practice (see [93]).

Figure 6 shows the interactions outlined sofar. Due to the fact that the constraint solver invokes the collision detection with a much higher rate than the time control module it might occur that a pentration is detected in the constraint solver, which the time control module has overlooked. If this ever occur the constraint solver should notify the time control about the "problem" so the time control can take the appropriate actions against the pentration.

2.1.4 The Collision Solver Module

In our frame computation we are now back at the time control after it has invoked the motion solver and we are ready for invoking the collision solver before the time control can continue with the next iteration. So let us investigate what happens in the collision solver.

When the collision solver is invoked it should compute collision impulses and apply these to all the colliding objects in the configuration. Applying an impulse to an object means making a discontineous change of the objects state. One therefore usually have to notify the motion solver about the discontineous change so it can update any state information it might store (recall that the motion solver handles the contineous movement of the objects). In Figure 7 we have omitted this notification for clarity reasons.

Computing impulses can be done either by using an algebraic law, an incremental law or a full deformation law (see [86]). A full deformation law means that one has to solve a partial differential equation describing how the physical quanties changes during a collision. Full deformation laws are not seen in real time rigid body simulation for two reasons. First it is almost imposible to determine the starting conditions for the partial differential equations and second even if one could the partial differential equations would be far to computational expensive to solve. Incremental laws uses a microscopic collision model, which is used to integrate over the collision, where as algebraic laws solves a system of equations. Incremental laws are more realistic accurate than algebraic laws but also computationally more expensive.

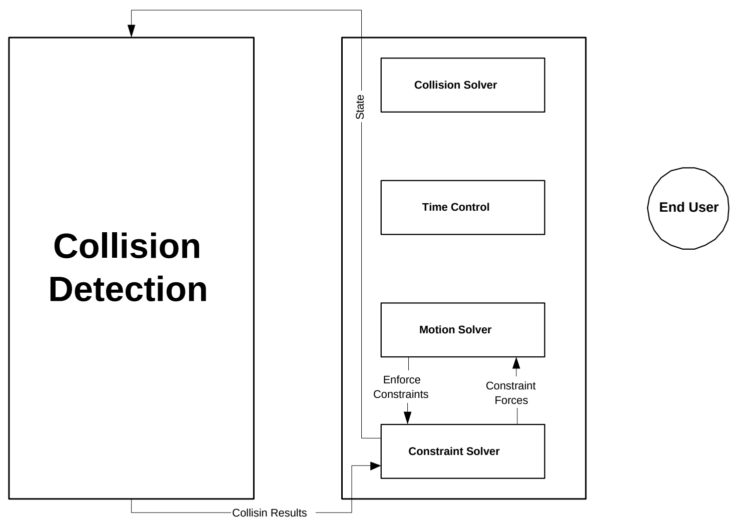

Figure 6: The interactions of the constraint solver module.

- Full Deformation Law
- Incremental Law
- Algebraic Law

Impulses can be applied to the objects either through simultaneous impulses or by sequential (also called propagating) impulses (see [59, 86, 75]). Simultaneous impulses means that all impulses are computed in a single step, where sequential means the impulses are computed one by one.

- Simultaneous Impulses
- Sequential Impulses

Notice that there is really no reason for invoking the collision detection in order to get the contact regions where the impulses should be applied, these contact regions was allready computed by the invokation of the collsion detection in the time control and can therefor be passed along to the collision solver. Figure 7 shows how the collision solver module interacts with the time control module.

By now we have a complete overview of how the simulation of our simulator works Figure 8 shows a complete overview of our walkthorugh.

2.2 The Collison Detection Component

Having taken care of the simulation, we can now concentrate on the collision detection. The collision detection does also consist of four modules: The broad phase collison detection module, the narrow phase collison detection module, the contact determination module and the contact analysis module.

- The Broad Phase Collision Detection Module
- The Narrow Phase Collision Detection Module
- The Contact Determination Module
- The Contact Analysis Module

Figure 9 shows these modules.

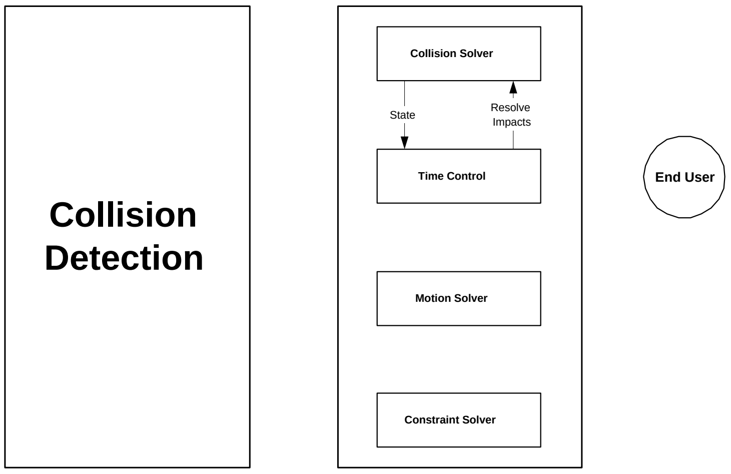

Figure 7: The interactions of the collision solver module.

2.2.1 The Broad Phase Collison Detection Module

The first module we encounter in the collision detection is the broad phase collision detection module (see [15, 16, 17, 18]). The purpose of the broad phase module is to reduce the amount of work which need to be done. The general idea is to find pairs of objects which are close enough to be consider for further collision testing. Typical algorithms used for this stage are exhaustive search, sweep n' prune [1, 94] (also called coordinate sorting) and (multilevel) grids (also known as hierarchical hashtabels) [75, 93]. The result of the broad phase collision detection module is a list of pairs of objects in close proximity, each of these pairs are send to the narrow phase collision detection module for furhter processing.

In order to determine close proximity, objects are often approximated by bounding volumes, in some cases the broad phase collision detection algorithm might even benefit from using information about the motion of the objects. From this information sweeping volumes can be constructed. These are sort of bounding volumes tracing out the motion of the objects in the near future (see [75]), alternatively one could use space-time bounds (see [33, 16]). How far one looks into the future is usually determined by passing along a look-ahead time step to the broad phase collision detection module (not shown in Figure 10). Using sweeping volumes or space-time bounds means that one can get proximity information about objects which are about to collide in the near future this is extremely usefull if one uses a one-sided-approach in the time control module.

2.2.2 The Narrow Phase Collison Detection Module

The next module we encounter in the collision detection is the narrow phase collision detection module. This module is responsible for testing whatever two pairs of objects are colliding or not. The algorithms to do this is usually capable of returning much more information than a simple yes-no answer. Often contact points, penetrating features and other proximity information is returned as the result of the narrow phase collision detection algorithm (see [55, 1, 18, 35, 31, 4, 83, 40, 27, 45, 9, 94, 38]). See Figure 11.

2.2.3 The Contact Determination Module

The proximity information returned from the narrow phase collison detection module can often be used with great advantage in the contact determination module (see [82, 50, 51, 94]). The contact determination module is responsible for computing the contact regions between those objects that are touching or penetrating. In mathematical terms one can think of the contact region as the union of the two object surfaces in touching or penetrating contact. For polygonal objects one often represents contact regions as contact formations consisting of principal contacts, i.e. paris of geometric features one from each obejct.

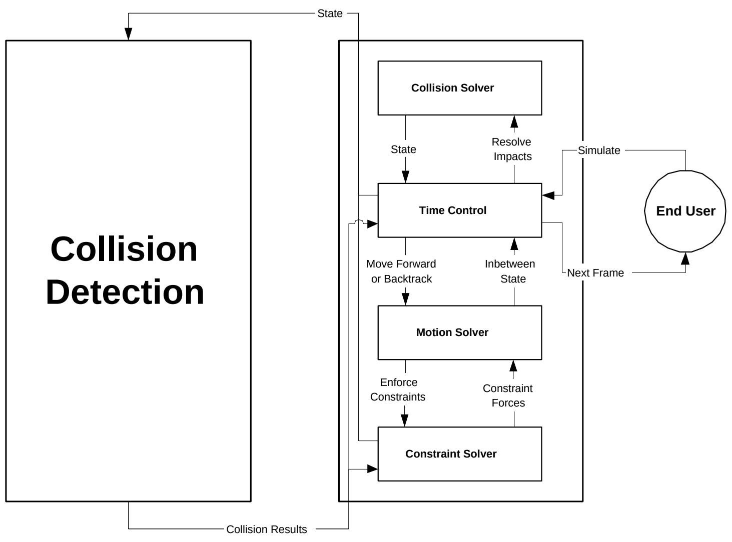

Figure 8: The entire simulation component of the simulator.

The contact determination is a pure geometric problem of determing the contact region. However the problem is not trivial considering uncertaincies and nonuniqueness of representation. Figure 12 illustrates the contact determination module.

2.2.4 The Contact Analysis Module

The final module in the collision detection we call the contact analysis module. The contact analysis module includes several things, first of all knowing that we work with physical based simulation we can turn the computed contact regions into support regions (see [48, 87]). There typically exist multiple solutions for the constraint forces of a contact region by computing the support region one eliminates the multiple solutions and the constraint forces becomes unique. This is computational tracktable in terms of computing constraint forces (it means fewer constraints to solve for).

The contact analysis module is also the place where contact groups are computed, that is clusters of objects which all are either in touching or penetrating contact. Again this information can be exploited both by the constraint solver and collision solver. Each cluster consist of an independent chunk of objects in the configuration. For each of these clusters we can compute both impulses and constraint forces independently from the other clusters.

The usage of contact groups can be taken even further by applying time warping to the simulator (see [91]). This basically means that each cluster can be simulated independently of other clusters and only when clusters interact one has to synchronize the simulation between them.

The contact group computation is usally not postponed to the very last minute in the collision detection. But a contact graph is usually keept and constantly updated with information when results comes back from both the broad phase, narrow phase and contact determination modules. This means that Figure 13 is a little misguiding with respect to the contact group computation but for clarity we have chosen to draw the figure as it is shown.

It has been discused (see [48]) whatever the support region computation should be placed in the collision detection or in the simulation components. Afterall it has more to do with physics than with geometry. We prefer the collision detection component as we have explained.

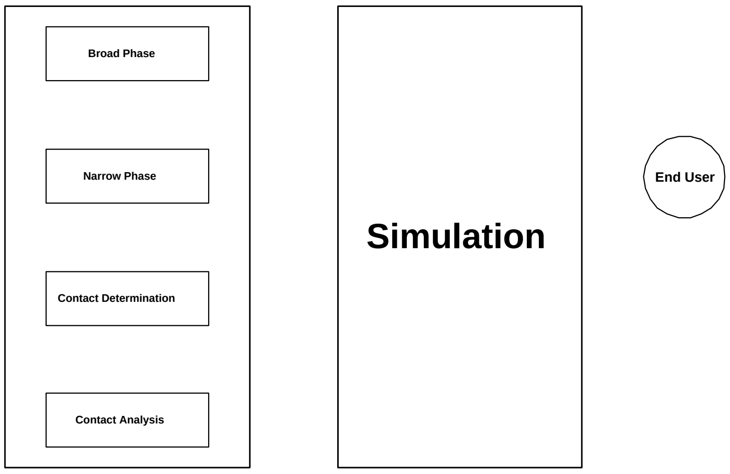

Figure 9: The four modules of collision detection.

2.3 Summary

This completes our walktrough of our general purpose module design. In Figure 14 you can see a schematic view of the entire module design.

We have succesfully applied this module design to rigid body simulation, and in the next section we will elaborate on the differences on the general purpose module design due to each of the simulator paradigms in rigid body simulation. However there is no reasons why the module design is not applicable to other kinds of simulation such as particle systems and soft body simulation. The major difference from rigid body simulation lies in the sort of collision detection algorithms one has to use.

3 Simulator Paradigms

In rigid body simulation there exist four different kinds of simulator paradigms: Impulse based methods, analytical methods (also sometimes called constraint based methods), penalty methods and hybrids. We will look closer on each paradigm in the following subsections.

3.1 Analytical Methods

An analytical simulator is the most complex of the simulator paradigms and all the modules of the general module design are needed as can be seen by comparing Figure 15 with Figure 14. There is really nothing more to say about this paradigm, since we have treated every aspect of it in our frame computation walkthrough.

Analytical simulators are typically very good at handling complex configurations with static contacts.

3.2 Penalty Methods

This paradigm is simpler than the previous one. Often one uses a simple fixed-time-stepping algorithm. This makes good sense since this paradigm allows penetration of the objects (neither analytical nor impulse based simulators general allow this). The greatest difficulty with this method lies in the requirements to the collision detection: It is not enough to determine whatever a penetration occurs, but both penetration depth and penetration points must also be computed.

Penalty forces can be computed in different ways, some methods are based on energy functions and penalty forces are found as the gradient of the energy functions. Other methods are more simple and simply use springs to push the penetrating objects away from each other.

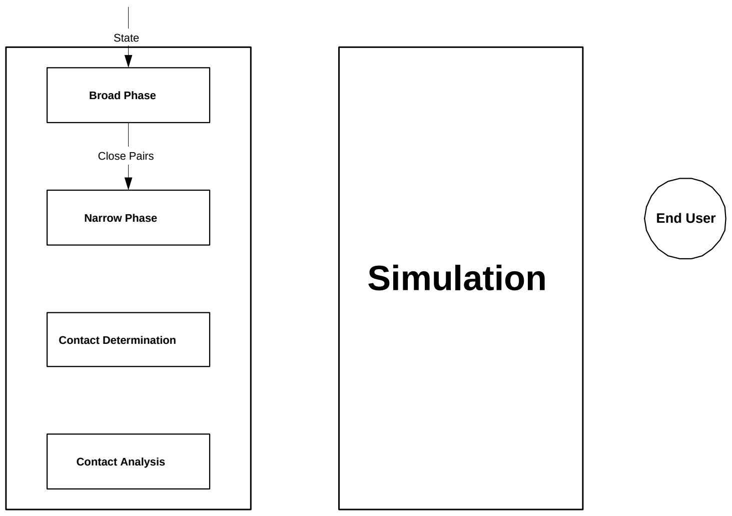

Figure 10: The broad phase collision detection module.

Penalty methods have another disadvantage deep penetrations could result in penalty forces which would make the ODE of the objects stiff. Numerical stability is therefor of major concern with penalty methods.

Figure 16 shows a typical penalty based simulator, notice that there is no collision solver. This is because if collisions are ignored then a penetration will occur in the near future. When the penetration occur it will be handled by the constraint solver, which will add a penalty force to reduce the penetration. This strategy requires very small time steps in the time control module. Some people have tried to avoid this problem by adding a collision solver to the simulator.

Penalty based simulators have got a lot of attention and is the preferred choice by many, mainly due to the simplicity of the modelling of physical interactions (springs) and they are easily extended to handle soft bodies as well.

3.3 Impulse based methods

Impulse based simulation was originally introduced by James K. Hahn (see [58]) and afterwards developed further by Brian Mirtich (see [75]). The idea behind this paradigm is to simulate all physical interactions between the objects in the configuration as collision impulses. Static contacts (i.e. resting contact) is modelled as a series of small microcollisions occuring at a very high frequency.

As Figure 17 indicates, this kind of paradigm is particular simple to implement, typically a one-sided-approach (see section 2.1.1) is used together with sequential impulses based on some sort of incremental law (see section 2.1.4). Impulses need only be applied at the closest point between two objects, so there is really no need for doing contact determination or contact analysis.

Impulse based simulators are rather bad at handling static contact, but they are effective for configurations with lots of objects moving at high speeds. Impulse based simulators also have a high performance and are therefore a good choice for real time simulation.

3.4 Hybrids

The final paradigm is hybrids. The general idea behind hybrids is to combine some of the other paradigms such that the weakness of one paradigm can be handled by another paradigm that does not suffer from the same weakness. Since hybrids are combinations of the paradigms we already have explained they can surely be handled by our general purpose module design.

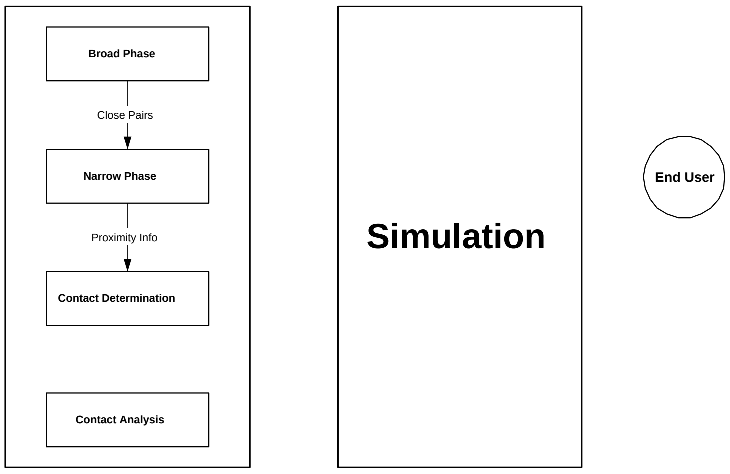

Figure 11: The narrow phase collision detection module.

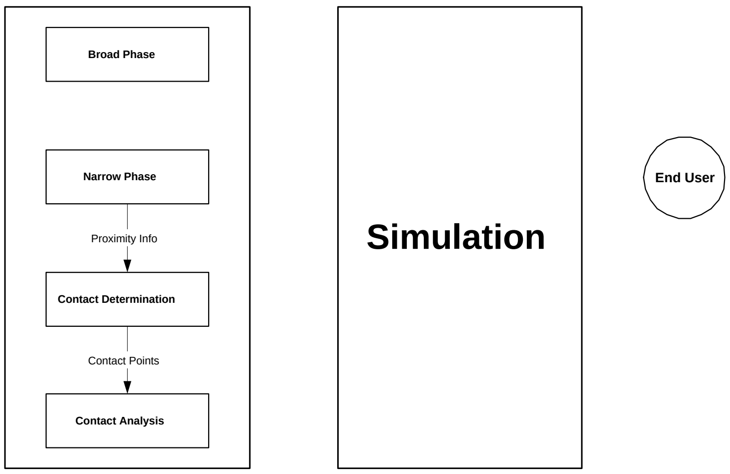

Figure 12: The contact determination module.

4 Conclusion

In this paper we have outlined a general purpose module design for a rigid body simulator. We have shown how different types of algorithms used in rigid body simulation fit into our module design. Several details on variations both on methods, heuristics and algorithms have been explained. Finally we have elaborated on how the module design applies to different simualtor paradigms.

5 Acknowledgments

I thank the people at Simulnicles for patiently trying out the module design in practice. I thank Kasper Hornbæk for his valuable comments and correcting typos.

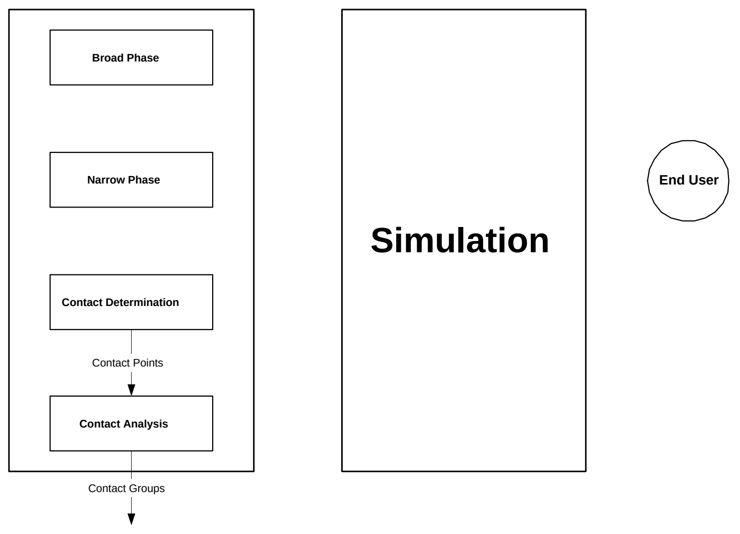

Figure 13: The contact analysis module.

References

[1] J. D. Cohen, M. K. Ponamgi, D. Manocha and M. C. Lin: *Interactive and Exact Collision Detection for Large-Scaled Enviroments*, Technical report TR94-005, Department of Computer Science, University of N. Carolina, Chapel Hill. http://www.cs.unc.edu/ dm/collision.html.

[2] D. Manocha and M. C. Lin: *Efficient Contanct Determination Between Geometric Models*, International Journal of Computational Geometry and Applications, 1995. http://www.cs.unc.edu/ dm/collision.html.

[3] M. K. Ponamgi, D. Manocha and M. C. Lin: *Incremental algorithms for collision detectin between solid models*, IEEE Transactions on Visualization and Computer Graphics , 1997. http://www.cs.unc.edu/ dm/collision.html.

[4] S. Krishnan, A. Pattekar, M. Lin and D. Manocha.: *Spherical shell: A higher order bounding volume for fast proximity queries*, In Proc. of Third International Workshop on Algorithmic Foundations of Robotics, pages 122-136, 1998.

[5] M. Lin and S. Gottschalk: *Collision Detection between Geometric Models: A Survey*, Appeared in the Proceedings of IMA Conference on Mathematics of Surfaces 1998. http://www.cs.unc.edu/ dm/collision.html

[6] S. Gottschalk, M. C. Lin and D. Manocha: *OBB-Tree: A Hierarchical Structure for Rapid Interference Detection*, Technical report TR96-013, Department of Computer Science, University of N. Carolina, Chapel Hill. Proc. of ACM Siggraph'96. 1996. http://www.cs.unc.edu/ geom/OBB/OBBT.html

[7] S. Gottschalk: *Seperating axis theorem*, Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill, 1996 ftp://cs.unc.edu/pub/users/Gottsch.

[8] S. Gottschalk: *Good-Fit Box for 3D Triangles in O(n lg(n)) Time*, erratta for [7]. http://www.cs.unc.edu/ geom/OBB/OBBT.html.

[9] S. Gottschalk: *Collision Queries using Oriented Bounding Boxes*, Ph.d. thesis, Department of Computer Science, UNC Chapel Hill, 2000. http://www.cs.unc.edu/ stefan/.

[10] David Eberly: *Dynamic Collision Detection using Oriented Bounding Boxes*, Magic Software, Inc. http://www.magic-software.com.

[11] David Eberly: *Intersection of Objects with Linear and Angular Velocities using Oriented Bounding Boxes*, Magic Software, Inc. http://www.magic-software.com.

[12] David Eberly: *Centers of Simplex*, Magic Software, Inc. http://www.magic-software.com.

[13] David Eberly: *Least Squares Fitting of Data*, Magic Software, Inc. http://www.magic-software.com.

[14] David Eberly: *Intersection of Cylinders*, Magic Software, Inc. http://www.magic-software.com

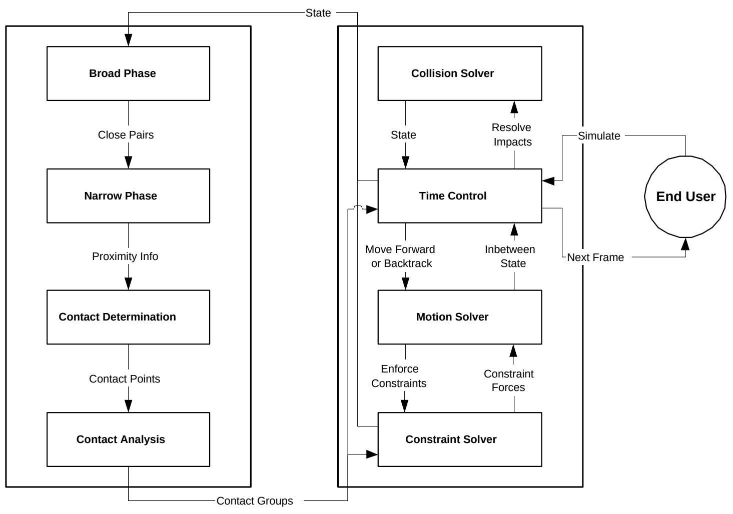

Figure 14: Complete General Purpose Module Design.

[15] P. M. Hubbard: *Interactive Collision Detection*, Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality, October 25-26, 1993, pp. 24-31. http://siesta.cs.wustl.edu/ pmh/research.html.

[16] P. M. Hubbard: *Collision Detection for Interactive Graphics Applications*, IEEE Transactions on Visualization and Computer Graphics , 1(3), September 1995, pp. 218-230. http://siesta.cs.wustl.edu/ pmh/research.html.

[17] P. M. Hubbard: *Real-Time Collision Detection and Time-Critical Computing*, Proceedings of the First ACM Workshop on Simulation and Interaction in Virtual Environments, July 1995, pp. 92-96. http://siesta.cs.wustl.edu/ pmh/research.html.

[18] P. M. Hubbard: *Approximating Polyhedra with Spheres for Time-Critical Collision Detection*, ACM Transactions on Graphics , 15(3), July 1996, pp. 179-210. http://siesta.cs.wustl.edu/ pmh/research.html.

[19] S. Suri, P. M. Hubbard and J. F. Hughes: *Analyzing Bounding Boxes for Object Intersection*, http://siesta.cs.wustl.edu/ pmh/research.html.

[20] S. Suri, P. M. Hubbard and J. F. Hughes: *Collision Detection in Aspect and Scale Bounded Polyhedra*, Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, January 1998, pp. 127-136.

[21] I.J.Palmer and R.L.Grimsdale: *Collision detection for animation using sphere-trees*, Computer Graphics Forum, 14(2), 1995, pp105-116.

[22] I.J.Palmer: *Collision detection for animation: the use of the sphere-tree data structure*, presented at The Second Departmental Workshop on Computing Research, University of Bradford, June 1995.

[23] Costas Tzafestas and Philippe Coiffet: *Real-Time Collision Detection using Spherical Octrees: Virtual Reality Application*, IEEE Internaltional Workshop on Robot and Human Communication, 1996.

[24] C. O'Sullivan and J. Dingliana: *Real-Time Collision Detection and Response Using Sphere-Trees*, Proceedings of the Spring Conference in Computer Graphics, Bratislava April 28-May 1st 1999 (ISBN 80-223-1357-2) pp 83-92. http://isg.cs.tcd.ie/Timecrit.html

[25] C. O'Sullivan and J. Dingliana: *Graceful Degradation of Collision Handling for Physically Based Animation*, Computer Graphics Forum Vol.20 Num.3 (Eurographics 2000 Proceedings) 2000. http://isg.cs.tcd.ie/Timecrit.html

[26] Gino van den Bergen: *Efficient Collision Detection of Complex Deformable Models using AABB Trees*, Department of Mathematics and Computer Science Eindhoven University of Technology, 1998. http://www.win.tue.nl/ gino/solid.

[27] Gino van den Bergen: *A Fast and Robust GJK Implementation for Collision Detection of Convex Objects*, Department of Mathematics and Computer Science Eindhoven University of Technology, 1999. http://www.win.tue.nl/ gino/solid.

[28] Gabriel Zachman: *Exact and Fast Collision Detection*, Diploma thesis Technical University Darmstadt, Dept. of Computer Science, 1994. http://www.igd.fhg.de/ zach.

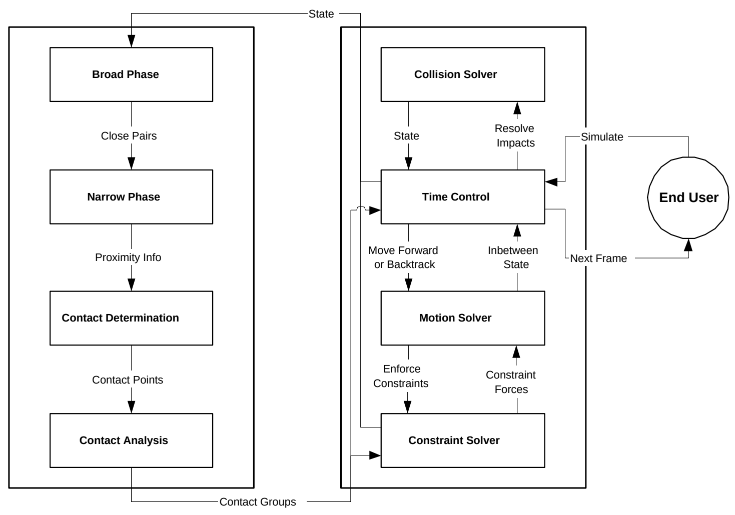

Figure 15: A typical analytical simulator.

[29] Gabriel Zachman: *The BoxTree: Exact and Fast Collision Detection of Arbitrary Polyhedra*, SIVE95 (First Workshop on Simulation and Interaction in Virtual Environments), University of Iowa, July 1995. http://www.igd.fhg.de/ zach.

[30] Gabriel Zachman: *Real-time and Exact Collision Detection for Interactive Virtual Prototyping*, Proc. of the 1997 ASME Design Engineering Technical Conferences, September 14-17, 1997, Sacramento, California. Paper # CIE-4306. http://www.igd.fhg.de/ zach.

[31] Gabriel Zachman: *Rapid Collision Detection by Dynamically Aligned DOP-Trees*, Proc. of IEEE Virtual Reality Annual International Symposium; VRAIS '98. Atlanta, Georgia; March 1998. http://www.igd.fhg.de/ zach.

[32] S. Cameron and R. K. Culley: *Determining the Minimum Translational Distance between Two Convex Polyhedra*, IEEE Conf. Robotics and Automation, San Francisco, April 1986. ftp://ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Stephen.Cameron.

[33] S. Cameron: *Collision Detection by Four Dimensional Intersection Testing*, IEEE Transactions on Robotics and Automation 6(3): pp. 291-302, June 1990.

[34] C. Qin, S. Cameron and A. Mclean: Towards Efficient Motion Planning for Manipulators with Complex Geometry, ISATP'95, August 1995. ftp://ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Stephen.Cameron.

[35] S. Cameron: *Enhancing GJK: Computing Minimum and Penetration Distances between Convex Polyhedra*, Int Conf Robotics and Automation, April 1997. ftp://ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Stephen.Cameron.

[36] S. A. Ehmann and M. C. Lin: *Accelerated Proximity Queries Between Convex Polyhedra By Multi-Level Voronoi Marching*, In Proc. International Conf. on Intelligent Robots and Systems, 2000. http://www.cs.unc.edu/ geom/SWIFT/

[37] S. A. Ehmann and M. C. Lin: *SWIFT: Accelerated Proximity Queries Between Convex Polyhedra By Multi-Level Voronoi Marching*, Technical Report, Computer Science Department, University of North Carolina at Chapel Hill, 2000. http://www.cs.unc.edu/ geom/SWIFT/

[38] S. A. Ehmann and M. C. Lin: *Accurate and Fast Proximity Queries Between Polyhedra Using Convex Surface Decomposition*, EUROGRAPHICS 2001, volume 20 (2001), Number 3. http://www.cs.unc.edu/ geom/SWIFT++/

[39] James Thomas Klosowski: *Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments*, Ph.D. Dissertation, State University of New York at Stony Brook, May 1998. http://www.ams.sunysb.edu/ jklosow/publications

[40] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral and K. Zikan: *Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs*, IEEE Transactions on Visualization and Computer Graphics, March 1998, Volume 4, Number 1. http://www.ams.sunysb.edu/ jklosow/publications

[41] E. Welzl: *Smallest enclosing disks (Balls and ellipsoids*, New Results and New Trends in Computer Science. Lecture Notes in Computer Science, vol 555, pages 359-370. Springer-Verlag, 1991.

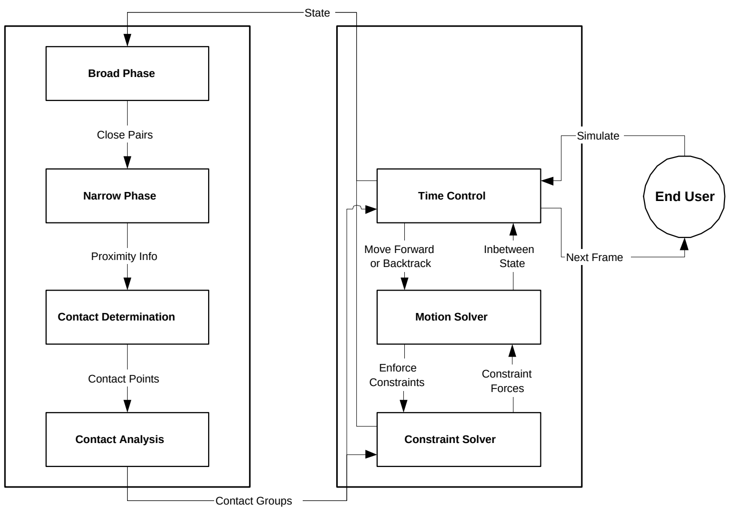

Figure 16: A typical penalty method based simulator.

[42] Thomas MÖller: *A Fast Triangle-Triangle Intersection Test*,

[43] David E. Johnson and Elaine Cohen: *A Framework For Efficient Minimum Distance Computations*, Proc. IEEE Intl. Conf. Robotics & Automation, Leuven, Belgium, May 16-21, 1998, pp. 3678-3684

[44] Tim Culver, John Keyser, and Dinesh Manocha: *Accurate Computation of the Medial Axis of a Polyhedron*, In Proc. of ACM Solid Modeling, 1999. UNC Chapel Hill Computer Science Technical Report TR98-034, 1998 http://www.cs.unc.edu/ geom/MAT

[45] Eric Larsen, Stefan Gottschalk, Ming C. Lin, Dinesh Manocha: *Fast Proximity Queries with Swept Sphere Volumes*, Technical report TR99-018, Department of Computer Science, University of N. Carolina, Chapel Hill. 1999. http://www.cs.unc.edu/ geom/SSV

[46] Pascal Volino and Nadia Magnenat Thalmann: *Collision and Self-Collision Detection, Efficient and Robust Solutions for Highly Deformable Surfaces*, Sixth Workshop on Computer Animation and Simulation, EGCAS'95.

[47] Joel Brown, Stephen Sorkin, Cynthia Bruyns, Jean-Claude Latombe, Kevin Montgomery and Michael Stephanides: *Real-Time Simulation of Deformable Objects: Tools and Application*, Computer Animation, Seoul, Korea, November 2001.

[48] William J. Bouma and George Vanecek Jr.: *Modelling Contacts in a Physical Based Simulation*, Proc. Solid Modeling, 409-419, 1993, 1, 3.

[49] A. Joukhadar, A. Wabbi and Ch. Laugier: *Fast Contact Localisation Between Deformable Polyhedra in Motion*, Conference Paper, In Proc. of the IEEE Computer Animation Conf., pp. 126-135, Geneva (CH), June 1996.

[50] A. Joukhadar, A. Scheuer and Ch. Laugier: *Fast Contact Detection between Moving Deformable Polyhedra*, Conference Paper, In Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems, Vol. 3, pp. 1810-1815, Kyongju (KR), October 1999.

[51] K. Sundaraj and Ch. Laugier: *Fast Contact Localisation of Moving Deformable Polyhedras*, Conference Paper, In Proc. of the Int. Conf. on Control, Automation, Robotics and Vision, Singapore (SG) (December 2000).

[52] John J. Craig: *Introduction to Robotics, mechanics and control*, Addision-Wesley Publishing Company, Inc, Second Edition.

[53] D. Terzopoulos, J.C. Platt, A.H. Barr and K. Fleischer, *Elastic deformable models*, Computer Graphics (proc. SIGGRAPH), vol. 21, pp 205-214, 1987.

[54] Roy Featherstone: *Robot Dynamics Algorithms*, Kluwer Academic Publishers, Second Printing 1998.

[55] M. Moore and J. Wilhelms, *Collision Detection and Response for Computer Animation*, Computer Graphics (proc. SIGGRAPH), vol. 22, pp 289-298, 1988.

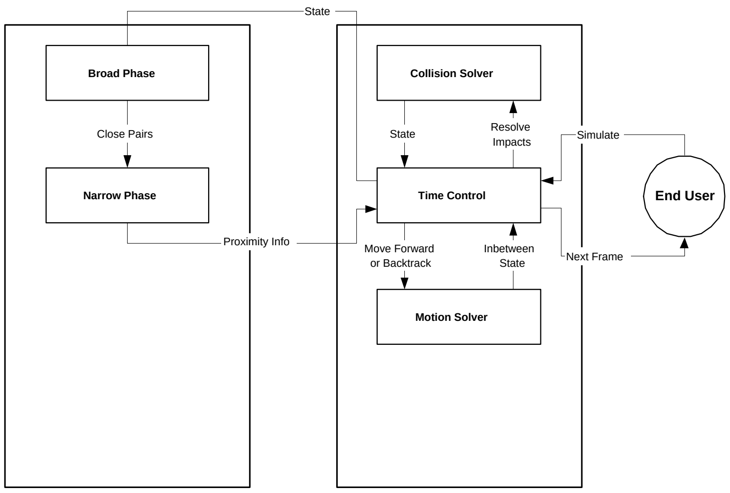

Figure 17: A typical impulse based simulator.

[56] J.C. Platt and A.H. Barr, *Constraint methods for flexible bodies*, Computer Graphics (proc. SIGGRAPH), vol. 22, pp 279-288, 1988.

[57] R. Barzel and A.H. Barr, *A Modeling System Based On Dynamic Constraints*, Computer Graphics (proc. SIGGRAPH), vol. 22, pp 179-187, 1988.

[58] J. K. Hahn, *Realistic Animation of Rigid Bodies*, Computer Graphics (proc. SIGGRAPH), vol. 22, pp 299-308, 1988.

[59] David Baraff: *Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies*, (SIGGRAPH 1989) Computer Graphics, Volume 23, Number 3, July 1989.

[60] David Baraff: *Curvec Surfaces and Coherence for Non-penetrating Rigid Body Simulation*, (SIGGRAPH 1990) Computer Graphics, Volume 24, Number 4, August 1990.

[61] David Baraff: *Coping with Friction for Non-penetrating Rigid Body Simulation*, (SIGGRAPH 1991) Computer Graphics, Volume 25, Number 4, July 1991.

[62] David Baraff: *Dynamic simulation of non-penetrating Rigid Bodies*, ph.d. thesis 1992.

[63] David Baraff, Raju Mattikalli, Bruno Repetto and Pradeep Khosla: *Finding Stable Orientations of Assemblies with Linear Programming*, CMU-RI-TR-93-13.

[64] David Baraff and Raju Mattikalli: *Impending Motion Direction of Contacting Rigid Bodies*, CMU-RI-TR-93-15.

[65] David Baraff: *Fast Contact Force Computation for Nonpenetrating Rigid Bodies*, (SIGGRAPH 1994) Computer Graphics Proceedings, Annual Conference Series 1994.

[66] David Baraff, Raju Mattikalli and Pradeep Khosla: *Minimal Fixturing of Frictionless Assemblies: Complexity and Algorithms*, CMU-RI-TR-94-08.

[67] John Canny and Brian Mirtich: *Impulse-based Dynamic Simulation*, Proceedings of Workshop on Algorithmic Foundations of Robotics, February 1994. http://www.cs.berkeley.edu/ mirtich/impulse.html.

[68] Brian Mirtich: *Hybrid Simulation:Combining Constraints and Impulses*, Proceedings of First Workshop on Simulation and Interaction in Virtual Environments, July 1995. http://www.cs.berkeley.edu/ mirtich/impulse.html.

[69] John Canny and Brian Mirtich: *Impulse-based Dynamic Simulation of Rigid Body Systems*, Proceedings of 1995 Symposium on Interactive 3D Graphics, April 1995. http://www.cs.berkeley.edu/ mirtich/impulse.html.

[70] Mihai Anitescu, James F. Cremer and Florian A. Potra: *Formulating 3D Contact Dynamics Problems*, Reports on Computational Mathematics, No 80/1995, Department of Mathematics, The University of Iowa.

[71] Mihai Anitescu, James F. Cremer and Florian A. Potra: *Properties of Complementary Formulations for Contact Problems with Friction*, Reports on Computational Mathematics, No 83/1995, Department of Mathematics, The University of Iowa.

[72] Jeff Trinkle, Jong-Shi Pang, Sandra Sudarsky and Grace Lo: *On Dynamic Multi-Rigid-Body Contact Problems with Coulomb Friction*, Tech Report 95-003, Texas A&M University, Department of Computer Science

[73] Devendra Kalra: *A General Formulation of Rigid Body Assemblies for Computer Graphics Modeling*, Tech Report: HPL-95-70.

[74] D.E. Stewart and J.C. Trinkle: *Dynamics, Friction, and Complementarity Problems*, International Conference on Complementarity Problems, Johns Hopkins University, Baltimore, Nov. 1995.

[75] Brian Mirtich: *Impulse-based Dynamic Simulation of Rigid Body Systems*, Ph.D. thesis, University of California, Berkeley, December, 1996, http://www.cs.berkeley.edu/ mirtich/impulse.html.

[76] Mihai Anitescu and Florian A. Potra: *Formulating Dynamic Multi-rigid-body Contact Problems with Friction as Solvable Linear Complementary Problems*, Reports on Computational Mathematics, No 93/1996, Department of Mathematics, The University of Iowa.

[77] F. Pfeiffer and M. Wösle: *Dynamics of Multibody Systems Containing Dependent Unilateral Constraints with Friction*, Journal of Vibration and Control 2, 161-192, 1996.

[78] P. R. Kraus and V. Kumar: *Compliant Contact Models for Rigid Body Collisions*, Proceedings of the 1997 IEEE International Conference on Robotics and Automation, Albuquerque, NM, 1997.

[79] P. Wolfsteiner and F. Pfeiffer: *Dynamics of a Vibratory Feeder*, In: Procedings of the 1997 ASME 16th Biennal Conference on Vibration and Noise, Sacramento, California.

[80] F. Pfeiffer and P. Wolfsteiner: *Relative Kinematics of Multibody Contacts*, In Proceedings of the International Mechanical Engineering Congress and Exposition, 16-21 Nov. 1997, Dallas, Texas.

[81] M. Anitescu and F.A. Potra, *Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementary problems*, Nonlinear Dynamics, 14:231-247,1997

[82] Brian Mirtich: *Rigid Body Contact: Collision Detection to Force Computation*, MERL, Technical Report, TR-98-01, March 1998.

[83] Brian Mirtich: *V-Clip: Fast and Robust Polyhedral Collision Detection*, ACM Transactions on Graphics. Vol. 17. No. 3. July 1998. Pages 177-208.

[84] Mihai Anitescu, Florian A. Potra and David E. Stewart: *Time-stepping for three-dimensional rigid body dynamics*, Comp. Methods Appl. Mech. Engineering, 1998.

[85] J. Sauer and E. Schömer: *A Constraint-Based Approach to Rigid Body Dynamics for Virtual Reality Applications*, in ACM Symposium on Virtual Reality Software and Technology, VRST'98, S. 153-161.

[86] Anindya Chatterjee and Andy Ruina: *A New Algebraic Rigid Body Collision Law Based On Impulse Space Considerations*, Journal of Applied Mechanics, Last edited August 6 1998, http://www.tam.cornell.edu/Ruina.html.

[87] M. Buck and E. Schömer: *Interactive rigid body manipulation with obstacle contacts*, in Journal of Visualization and Computer Animation 1998, Vol.9, S. 243-257.

[88] M. Buck and E. Schömer: *Interactive rigid body manipulation with obstacle contacts*, in 6th International Conference in Central Europe on Computer Graphics and Visualization, WSCG'98, S. 49-56.

[89] M. Wösle and F. Pfeiffer: *Dynamics of Multibody Systems with unilateral constraints*, International Journal of Bifurcation and Chaos, Vol. 9, No. 3 (1999) 473-478.

[90] C. Lennerz, E. Schömer and T. Warken: *A Framework for Collision Detection and Response*, in 11th European Simulation Symposium, ESS'99, S. 309-314.

[91] Brian Mirtich: *Timewarp Rigid Body Simulation*, MERL - A Mitsubishi Electric Research Lab.

[92] Kenny Erleben: *En introducerende lærebog i dynamisk simulation af stive legemer*, Speciale, Datalogisk Institut Københavns Universitet. Maj 2001 (nr. 00-09-1).

[93] Murilo G. Coutinho: *Dynamic Simulations of Multibody Systems*, Springer-Verlag, 2001.

[94] David Baraff: *Physical Based Modeling: Rigid Body Simulation*, Pixar Animation Studios, ONLINE SIGGRAPH 2001 COURSE NOTES.

[95] Knud Henriksen og Kenny Erleben: *Scripted Motion and Spline Driven Motion*, To appear, Technical Report, Department of Computer Science University of Copenhagen, July 2002.