Module Based Design for Rigid Body Simulators

Kenny Erleben

Abstract: In this paper we outline a module design for a rigid body simulator. The purpose of the module design is to clarify how many parts a simulator 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 2

2 A Module Design 2
  2.1 The Simulation Component .................................................. 2
    2.1.1 The Time Control Module ............................................... 3
    2.1.2 The Motion Solver Module .............................................. 5
    2.1.3 The Constraint Solver Module ......................................... 6
    2.1.4 The Collision Solver Module ........................................... 7
  2.2 The Collision Detection Component ........................................ 8
    2.2.1 The Broad Phase Collision Detection Module ....................... 9
    2.2.2 The Narrow Phase Collision Detection Module ..................... 9
    2.2.3 The Contact Determination Module .................................. 9
    2.2.4 The Contact Analysis Module ....................................... 10
  2.3 Summary ................................................................. 11

3 Simulator Paradigms 11
  3.1 Analytical Methods ......................................................... 11
  3.2 Penalty Methods .......................................................... 11
  3.3 Impulse based methods ..................................................... 12
  3.4 Hybrids ................................................................. 12

4 Conclusion 13

5 Acknowledgments 13
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 research 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 literature is written as articles describing a small part of the functionality of a simulator, such as a single algorithm or data structure. Even though most of this literature 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, determining 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 general purpose module design [92]. Over the years we have continued our work and our early module design has now evolved into a more mature and pedagogical 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 paradigms.

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

\[1\text{The result was a module design known as QAD, which is an acronym for Quick And Dirty.}\]
2 A MODULE DESIGN

![Diagram showing two components: Collision Detection and Simulation](image)

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

- Motion Solver Module
- Constraint Solver Module

![Diagram showing four modules: Collision Detection, Collision Solver, Time Control, Motion Solver, and Constraint Solver](image)

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 telling the time control to simulate forward in time from the current time, \( t_{\text{cur}} \), to \( t_{\text{next}} = t_{\text{cur}} + \Delta t \). The configuration state at time \( t_{\text{next}} \) 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 \( t_{\text{next}} \). The later group of algorithms is further divided into two subgroups called backtracking and one-sided-approach.

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

What kind of algorithm that is used depends 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 occurred. If a penetration occurred 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 earliest 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 pattern 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 reminiscent of distributed algorithms (see [91]).

![Diagram of simulation modules interaction](image)

**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 continuous 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]):

$$\frac{d}{dt} \begin{bmatrix} \vec{r} \\ \dot{q} \\ \ddot{q} \\ \dddot{q} \end{bmatrix} = \begin{bmatrix} \vec{g} \\ \frac{1}{2} [\vec{\omega}, \vec{\omega}] \dot{q} \\ \vec{F} \\ \tau \end{bmatrix}$$

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

$$Y(t) = \{\vec{r}, q, \vec{\omega}, \dot{\vec{r}}, \dot{q}, \ddot{\vec{r}}, \ddot{q}, \dddot{\vec{r}}, \dddot{q}\}$$

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 $\frac{1000}{10000} \text{sec}$ 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.

![Diagram of the Motion Solver](image)

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 continous 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 succeeding ODE integration steps.

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

One could of course ask the question, which one of these methods are the correct way of doing things? In our opinion 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 continuous 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 so far. 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 penetration 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 penetration.

### 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 discontinuous change of the objects state. One therefore usually have to notify the motion solver about the discontinuous change so it can update any state information it might store (recall that the motion solver handles the continuous 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 quantities changes during a collision. Full deformation laws are not seen in real time rigid body simulation for two reasons. First it is almost impossible 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, whereas 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 already computed by the invocation of the collision 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 walkthrough.

2.2 The Collision 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 collision detection module, the narrow phase collision 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.
2.2.1 The Broad Phase Collision 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 considered 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 hash tables) [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 sent to the narrow phase collision detection module for further 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 useful if one uses a one-sided-approach in the time control module.

2.2.2 The Narrow Phase Collision 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 collision 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. pairs of geometric features one from each object.
The contact determination is a pure geometric problem of determining the contact region. However the problem is not trivial considering uncertainties 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 tractable 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 usually not postponed to the very last minute in the collision detection. But a contact graph is usually kept and constantly updated with information when results come back from both the broad phase, narrow phase and contact determination modules. This means that Figure 13 is a little misleading with respect to the contact group computation but for clarity we have chosen to draw the figure as it is shown.

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

![Diagram](image-url)
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 successfully 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.
Penalty methods have another disadvantage deep penetrations could result in penalty forces which would make the ODE of the objects stiff. Numerical stability is therefore 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 occurs 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 occurring 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.
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 simulactor paradigms.

5 Acknowledgments

I thank the people at Simulhicles 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


Figure 14: Complete General Purpose Module Design.


REFERENCES


Figure 16: A typical penalty method based simulator.

Figure 17: A typical impulse based simulator.


REFERENCES

[72] Jeff Trinkle, Jong-Shi Pang, Sandra Sudarsky and Grace Lo: On Dynamic Multi-Rigid-Body Contact Problems with Coulomb Friction, Tech Report 95-063, Texas A&M University, Department of Computer Science


