Realtime Rigid Body Simulation Using Impulses

Scott Lembcke
University of Minnesota Morris
600 E. 4th St.
Morris, Minnesota
lemb0029@morris.umn.edu

ABSTRACT

Using impulses for rigid body simulation is rapidly becoming a standard method for realtime simulations. This paper discusses how impulses can be used for modeling both collision and contact between rigid bodies when using the coulomb friction model. Time stepping techniques are also discussed including an example method optimized specifically for handling large numbers of simultaneous collisions and contacts. Lastly, several advanced optimizations that can be applied when solving contacts and do not sacrifice the visual quality of the simulation are discussed.

General Terms

Rigid body simulation

Keywords

realtime, rigid body, physics, simulation, animation

1. INTRODUCTION

Computer generated animation is becoming more common in everyday life in movies, television shows, and video games. Creating computer animation is often a time consuming and laborious chore, especially when trying to recreate natural motion. For instance, creating a convincing animation of a falling tower of blocks by hand would be very difficult despite the simplicity of the rules governing the motion of the individual blocks. Blocks bounce off of each other and gravity pulls them down. Anyone who has taken a physics course should be able to calculate the amount of friction generated as a block slides across a table, but other problems can't be done so easily. Will the friction force cause the block to tip over? Will a tower of blocks fall over if you take away one of the blocks in its base? How will it fall? Where will all the blocks land? These are questions that you probably could not figure out on paper.

Creating computer simulations of physical systems can also be difficult despite the computer's strength at solving mathematical problems. Even something as ordinary as a pile of blocks is a surprisingly complicated simulation. Despite the challenges, realistic animation is important as it helps complete the sense of immersion in a virtual scene. Realtime simulations are also especially important in video games and other interactive programs. Allowing users to manipulate objects in a realistic and intuitive way can be a powerful yet simple way to interact with a computer.

2. RIGID BODY DYNAMICS

Real objects can interact in many complicated ways. For instance, they can be broken, crushed, bent, cut, melted, and burnt. Modeling any of these interactions in a computer simulation is difficult and computationally expensive. In order to achieve realtime simulation speeds, many simplifications must be made in the allowed behaviors of the objects.

2.1 Particle dynamics

The simplest way to represent an object is by reducing it to a point mass, a singular point with an associated mass. This is referred to as particle dynamics. It is most often used to simulate the motions of small objects, such as the particles in a cloud of dust. However, the use of simple constraints such as distance constraints or spring forces allows particle dynamics to be used for coarsely simulating other systems such as fluids or flexible materials like cloth or rope. The limitation of particle dynamics is that a point mass has no concept of rotation. So simulating large solid objects is unrealistic as they are unable to rotate.

2.2 Rigid body dynamics

In rigid body dynamics, objects are allowed to rotate, but are composed of an infinitely stiff material that can not deform or break. This is a reasonable simplification as many of the solid objects we see around us everyday do not deform noticeably. Using rigid body dynamics, it is possible to simulate a ball rolling across the floor or a box tumbling down a hill.

Much like using multiple linked particles to simulate more complicated objects, such as cloth, using multiple linked rigid bodies can approximate more complicated objects as well. For instance, a semi-stiff object like a mattress could be simulated by connecting several several rigid bodies with damped spring joints. Complicated interactions between rigid bodies such as breaking can be approximated by replacing a single rigid body with several new ones if too much stress is applied to the object.

3. HANDLING COLLISION AND CONTACT

Simulating a physical system in which bodies did not collide would be very easy. A simple differential equation that models the system could be created and the initial value problem for every body in the system solved to obtain equations that could describe the system at any given point in time. However, collisions and user interactions makes the motion of the objects unpredictable, so a general equation is not possible. This is what makes rigid body simulation difficult.

The simplest and most prevalent ways for objects to interact are through collisions and contact. Both have a very distinct effect and must be differentiated in a simulation. Collisions occur when two objects in free flight impact each other and bounce apart. In rigid body simulations, collisions are usually handled by applying impulses that discontinuously modify the velocities of the bodies at the time of collision. A rigid body collision should not to be confused with collision detection, the latter is a problem of determining if and how two objects are intersecting. Contact occurs when two objects are resting on one another, such as a book laying on a table. Handling contacts is a more difficult problem than handling collisions. In addition to the difficulty of solving contats, being able to differentiate between collision and contact is not an easy problem when given only the state of the objects and the information given by the collision detection algorithms. Some methods for handling contact include using penalty forces, impulses, and using analytic contact.

3.1 Computing collision impulses

Collisions are the easiest interaction to model because colliding objects are only in contact for an instant. Additionally, collisions between rigid objects have been studied for a long time and the problem is well understood. This has produced the idea of the *ideal impulse*, an instantaneous transfer of momentum between two colliding rigid bodies. While collisions between real objects are not instantaneous because they deform on impact, for most objects the time is short enough to have little effect on the outcome.

Before calculating a collision impulse, there are several other values that must be found first. The collision point (p) is the point at which the bodies are touching while the collision normal ($\vec{n}$) is the direction that the bodies will separate in at the collision point after the collision. Determining these two quantities is the job of the collision detection algorithms, and is not discussed in this paper. The coefficient of restitution (ϵ) is the amount of bounce in the simulation while the frictional coefficient (μ) determines the ratio of the normal force to the frictional force. These two values differ depending on the type of materials in contact. Lastly, we need to know some information about the objects in contact, their velocity ($\vec{v}$), their rotational velocity ($\vec{\omega}$), their mass (m), and their moment of inertia ($\mathbf{I}$). The moment of inertia is best explained by the analogy mass is to velocity as the moment of inertia is to rotational velocity.

Consider the collision in *figure 1*. A collision is occurring at point P with normal $\vec{n}$.

The normal impulse, j_n, stops the collision of the two objects so that $\vec{v}_r \bullet \vec{n} = 0$. j_n is computed as follows:

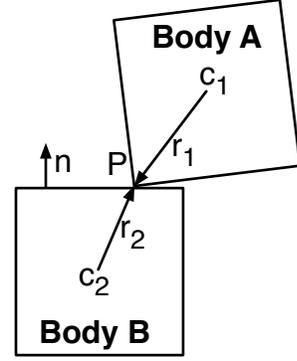$$j_n = -\frac{(\vec{v}_r \bullet \vec{n})\,(\epsilon + 1)}{k_n}$$

Figure 1: Two colliding boxes.

where:

$$\vec{v}_r = (\vec{v}_2 + \vec{\omega}_2 \times \vec{r}_2) - (\vec{v}_1 + \vec{\omega}_1 \times \vec{r}_1)$$

$$k_n = \frac{1}{m_1} + \frac{1}{m_2} + \left[\mathbf{I}_1^{-1}\,(\vec{r}_1 \times \vec{n}) \times \vec{r}_1 + \mathbf{I}_2^{-1}\,(\vec{r}_2 \times \vec{n}) \times \vec{r}_2 \right] \bullet \vec{n}$$

If j_n is negative, the bodies are already separating at the collision point. Applying this impulse would push the objects back together. This means j_n should be clamped so that $j_n \geq 0$.

The normal impulse does not include the effects of friction. Using the coulomb model of friction, it is fairly straightforward to find the frictional or tangential impulse. The formulas are similar to the normal impulse, but the final impulse is clamped to a proportion of the normal impulse based on the frictional coefficient.

$$j_t = clamp\left(-\frac{(\vec{v}_r \bullet \vec{t})\,(\epsilon + 1)}{k_n}, [-\mu j_n, \mu j_n] \right)$$

where:

$$\vec{t} = \frac{(\vec{n} \times \vec{v}_r) \times \vec{n}}{|\vec{v}_r|}$$

$$k_t = \frac{1}{m_1} + \frac{1}{m_2} + \left[\mathbf{I}_1^{-1}\,(\vec{r}_1 \times \vec{t}) \times \vec{r}_1 + \mathbf{I}_2^{-1}\,(\vec{r}_2 \times \vec{t}) \times \vec{r}_2 \right] \bullet \vec{t}$$

The collision impulse, $\vec{j}$, would then be the sum of the normal and tangent impulses. So $\vec{j} = (j_n \cdot \vec{n}) + (j_t \cdot \vec{t})$. This impulse is then applied to body A in the following manner:

$$\vec{v}_1' = \vec{v}_1 + \frac{\vec{j}}{m_1}$$

$$\vec{\omega}_1' = \vec{\omega}_1 + \left(\vec{r}_1 \times \vec{j} \right) \mathbf{I}_1^{-1}$$

The opposite impulse is applied to body B:

$$\vec{v}_2' = \vec{v}_2 - \frac{\vec{j}}{m_2}$$

$$\vec{\omega}_2' = \vec{\omega}_2 - \left(\vec{r}_2 \times \vec{j} \right) \mathbf{I}_2^{-1}$$

3.2 Handling contact

Compared to collisions, contact can be hard to solve effectively. This is because continuous contact forces need to be

modeled in a discrete simulation. Most naive implementations that try to handle contact as collisions lead to objects that vibrate wildly, sink into one another, or both.

3.2.1 Penalty force method

One way to handle the contact problem is to apply forces to the contact points that are integrated as continuous throughout the time step. These forces can be thought of as being generated by stiff springs that are attached at the contact points. Interpenetration is often resolved by simply increasing the spring forces based on the penetration depth.

The penalty force method performs relatively well, but determining correct forces to apply be difficult. In fact, Baraff proves that computing correct frictional contact forces can be NP complete in some cases [1].

3.2.2 Impulse based methods

Using impulses exclusively for both collision and contact is becoming much more popular for realtime simulations. This is because impulses are relatively easy to work with and apply. Also, using impulses for both collision and contact makes the implementation simpler and easier to understand and optimize. There are a number of ways to use impulses effectively, but all require making changes to the time stepping algorithm. One example of a time stepping algorithm is given in the next section.

3.2.3 Computing impulses for contact

Impulses used for contact can not be computed in the same way that collision impulses are calculated. Collision impulses are good at making objects bounce, this is not desirable when objects are resting on one another. Treating the contacts as inelastic collisions will help, but there is another problem. The impulse applied to any given contact point affects the all the other impulses applied at other contact points in a group of bodies that are resting on each other. This means that applying contact impulses to each contact point sequentially will not really work as it did when resolving collisions. Consider, for example, a stack of boxes. Applying contact impulses between the ground and the bottom box will stop its motion towards the ground, but when impulses are applied between the two boxes, the top box will push the bottom box towards the ground again. The contact constraints form a linear system of equations, and the impulses to be applied can be found by solving the equation.

3.2.4 Handling interpenetration

For highly accurate physical simulations, interpenetration would probably be considered an error and would be dealt with by reducing its occurrence in the first place. In realtime simulations, it is simply a byproduct of using approximations. Resolving collisions only at discrete time intervals, and treating contact with impulses instead of forces are the main causes of interpenetration. Because interpenetration is a nonrealistic behavior, there is no correct or realistic way to handle it. The best that can be done is to reduce the visual impact.

There are many possible ways of handling interpenetration. One possibility is to simply reduce the relative velocities as much as possible during the contact phase. This is certainly the most complicated method, but produces the best results as there shouldn't be any interpretation to fix. Guendelman's Shock Propagation method [3] takes this

route. The problem with this method is that interpenetration can still happen and may become a problem over time if nothing else is done to correct it. Another possibility is to modify the positions of the bodies. This method is simpler, but can cause the velocity to get out of sync with the position. A final possibility is to bias the velocities of the contacts so that the bodies will separate over time. This method is the simplest, but will cause objects to bounce unnaturally.

4. TIME STEPPING TECHNIQUES

Because of collisions, it is not possible to determine the position of an object at any time from a simple equation. It is also not possible to continuously simulate a system for every moment in time on a computer. Consequently, time has to be broken down into discrete steps. The simplest method is to apply the equations of motion, and then solve for collisions and contact. However, in all but the simplest cases, this time stepping algorithm performs very poorly, producing neither a physically accurate result or even realistic looking animation. This brings up an important distinction: An animation can look physically plausible without being physically accurate. Fortunately, realtime simulations are not used for their accuracy which allows many optimizations to be made as long as the resulting animation looks plausible.

4.1 Integration

The motion of an object in free flight can simply be reduced to solving a simple 2nd order, linear, constant coefficient, non-homogenous differential equation. The exact solution, $\Delta \vec{x} = \vec{v} \cdot \Delta t + (\vec{a} \cdot \Delta t^2)/2$, can be found easily enough using simple integration. However, an exact solution is not always desirable because the acceleration term can cause the object to move even when $\vec{v} = 0$. This can be a problem because an object resting on the ground will penetrate slightly into the ground after updating the object's position. Using a simpler, less accurate integration method such as Euler integration can help provide a more stable simulation because an object at rest will stay put, reducing the amount of penetration that needs to be fixed later.

An even simpler integration technique often used in video games with low system requirements is vertlet integration [4]. In vertlet integration, instead of explicitly storing the velocity for an object, its current and past position are stored. When integrating, the velocity is approximated as the distance it moved in the last time step. Resolving inelastic collisions and resting contacts simply becomes a problem of resolving penetrations. This method can produce fairly realistic looking animation when realistic friction and elasticity are not needed.

A problem common to almost any time stepping method is that simulations can gain energy due to coarse time steps and rounding errors. For instance, imperfections in a simulation of a super ball could cause the ball to jump higher and higher on each bounce. To overcome this, a damping coefficient is often applied when updating velocities. This slowly reduces the velocity of objects to prevent them gaining energy in unnatural ways. When used in small enough amounts it can correct the problem without becoming visible and appears to simulate objects in a viscous fluid when used in large amounts.

4.2 Guendelman's time stepping algorithm

In [3] Guendelman et al proposed an algorithm for time integration that is optimized for systems with many collisions and contacts. Using their method, all the collisions that will happen throughout the time step are resolved first. The collisions are resolved by calculating impulses to apply at their contact points. For efficiency reasons collisions are processed in the order that they are found, not in the order that they occur. This step can be done iteratively to allow objects to collide multiple times as changing their velocities changes where they will go, opening the possibility for new collisions. Iterating the collisions in this manner can enable the simulation of a Newton's cradle, as the impulses can be passed from object to object within a single time step. Once the objects have bounced around for a few iterations, the velocities of all the objects in the system are integrated.

Next, contacts are handled in a very similar fashion to collisions. First, find all contacts that will occur in the remainder of the time step and handle them as though they were inelastic collisions. Treating the contacts as inelastic collisions will keep the objects from bouncing when they should be resting. Contact can also be applied iteratively. Applying the contacts iteratively approximates the linear system for the contact constraints. Using more iterations increases the accuracy of the solution.

An example is given in [3], describing the benefits of this time stepping algorithm. Imagine a very bouncy cube ($\epsilon \approx 1$) sitting on an inclined plane with enough friction to stop its motion. Using the normal update and collide time stepping algorithm, the velocity of the cube would be integrated first. Then when a low velocity collision would be registered with the ground plane and impulses would be applied to the cube that would cause it to bounce at an angle down the inclined plane. Subsequent bounces would only make the problem worse, and the box would continue to gain energy. Using Guendelman's new time stepping scheme, the time step is started by finding and resolving all collisions. Doing so, a collision between the cube and the ground would be found, but the cube is still at rest so the collision impulse would be zero. Next, the velocity of the cube would be integrated, and the collision detection rerun. Once again, a collision between the ground plane and cube would be found. Using $\epsilon = 0$, the impulse calculated would be just big enough to stop the motion of the cube and not make it bounce. Finally, when the cube's position is integrated, it would stay in place because its velocity is zero.

5. ADVANCED CONTACT TECHNIQUES

In addition to the methods presented earlier, there are a number of advanced techniques that can be used to reduce the amount of computation required to solve contacts. Some selected methods include shock propagation [3], freezing [5], and contact persistence [2].

5.1 Shock propagation

In [3] Guendelman et al, they create a contact graph to aid in the contact resolution stage in a unique way. First of all, a contact graph is a directed graph that shows the relation relation of resting contacts between objects. For a stack of objects, the graph would start at the ground plane (or another static object) and have directed edges pointing from one object to the next up the stack. Using such a

graph, the contacts can be solved in an order that allows the solution to converge in fewer iterations.

They modified the general contact graph algorithm by processing the graph into a tree. Cycles in the graph are handled by transforming them into flat levels of the tree. Contacts are resolved by starting iterating over the objects in the first level of the tree, solving the impulses. After a certain number of iterations, they move on to the next level of the tree and repeat the process. This process is then iterated over the tree several times. The advantage of of resolving the contacts in this order is that the impulse chain is started at the ground, always moving the velocity of objects in stacks upwards. If the impulses were applied in a random order, objects could get pushed up and down, wasting some of the work done by earlier impulses.

The other modification made to this algorithm is a final iteration over the tree performing what Guendelman calls shock propagation. Even after a large number of iterations, objects in a stack will still have residual downward velocities. The shock propagation step minimizes the velocities as much as possible. Shock propagation is performed by making the objects in a tree level static (giving them infinite mass) before moving on to the next level. Giving the objects infinite mass forces the objects above to move out of the way. Using shock propagation, fewer iterations are needed for the contact solution. To appear physically plausible, not all of the momentum needs to be transfered between objects in a realistic way. The shock step takes care of stopping the objects from moving towards each other. Shock propagation even works well enough to prevent interpenetration that Guendelman did not feel that adding an explicit penetration resolution step was necessary.

5.2 Freezing

In [5], Schmidl and Melenkovic present another method for increasing speed when resolving contacts. The basis of their idea is that objects that have come to rest should not need to be simulated any more until they begin moving again. They call their method *freezing*.

In their simulation, they classify objects as three different types: fixed, dynamic, and frozen. Walls or the ground plane are examples of fixed objects. Dynamic objects are objects that are still in motion, and frozen objects are objects that are at rest. Dynamic objects can become frozen if they stop moving and are flagged to be at rest by a heuristic. Similarly, frozen objects can become dynamic again if they are hit with enough force or if an object supporting it moves.

The heuristic they give to determine if an object is at rest simply compares the amount of kinetic energy that the object has (linear and rotational) and compares that to the amount of energy it would gain due to gravity in a time step. If the energy is below the threshold for a certain number of contiguous frames, then the object is determined to be at rest.

Frozen objects never have their velocities or positions updated, and they do not have to check collisions with other frozen objects. They are effectively removed from the simulation. This will have little to no visible impact on the simulation unless the resting heuristic is poorly implemented and freezes objects too soon. In fact, for simulations that are optimized for speed over accuracy, it can actually improve the realism as objects will be frozen instead of drifting due to a poor contact solution.

5.3 Contact persistence

In [2], Catto presents a caching scheme for iterative solutions. The basis of the idea is that the contact impulses for objects at rest will be very similar to the time step before. If the final computed impulse was cached, it could be used again during the next time step as the starting point when solving the contact impulse again. Thus the contact solution can use fewer iterations per step because the solution will continue to converge over time. Using contact persistence, often an order of magnitude fewer iterations can be used and will still provide a better contact solution than a plain iterative solver. However, if the number of iterations is decreased too much, the time the solution takes to converge will become visibly noticeable. This causes objects to mush into and slide over one another for a large number of frames before coming to rest.

6. CONCLUSION

In conclusion, impulses can work very well when applied to realtime simulations. Although impulses do not produce physically accurate results, they do produce visually plausible results. For most realtime simulations this is more than adequate. Furthermore, when the results only need to be visually plausible,a great deal of optimizations can be made in order to speed the computation up without affecting the visual quality of the simulation. Simple optimizations include treating collisions and contacts in the order that they are found or in an order that will help the solution to converge faster. When combined with an advanced technique such as freezing, contact persistence or shock propagation modern computers can simulate thousands of objects in a scene even on consumer hardware in realtime.

7. REFERENCES

[1] D. Baraff. Coping with friction for non-penetrating rigid body simulation. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 31–41, New York, NY, USA, 1991. ACM Press.

[2] E. Catto. Iterative dynamics with temporal coherence, 2005.
http://www.gphysics.com/files/IterativeDynamics.pdf.

[3] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex rigid bodies with stacking. *ACM Trans. Graph.*, 22(3):871–878, 2003.

[4] T. Jakobsen. Advanced character physics, 2001.
http://www.teknikus.dk/tj/gdc2001.htm.

[5] H. Schmidl and V. J. Milenkovic. A fast impulsive contact suite for rigid body simulation. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):189–197, 2004.