Continuous Collision Detection in 3-D

Thomas Jansen

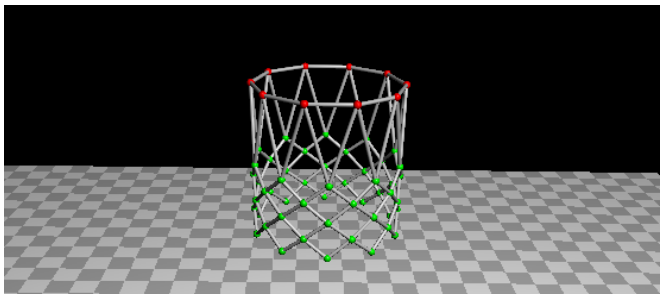

Figure 1: *Simulated Basketball Hoop*

Abstract

In this paper, we describe a way to robustly check for and respond to collisions between edges in three dimensions. All of the math involved in finding co-planarity, solving for collision times, taking roots of complex numbers, and finding intersections is explained in full detail to make the process straightforward to understand and replicate. All of these algorithms work for edges in any combination and configuration, although efficiency depends on the number of edges that exist in the system. A class that builds discretized cables is described, although bending and torsion have not yet been simulated.

1 Introduction

Creating robust collision detection algorithms is an essential piece of the now rapidly growing field of computer simulation. Computer simulations provide the means to test real-world systems without the risk involved. Whether it's to practice flying an aircraft, or to test an idea for a new roller-coaster, simulations are fundamental for avoiding potentially dangerous or costly situations in the real world. One of the key pieces in simulations is accurately responding to collisions between objects, and a continuous detection system allows for those collisions to be seen and accounted for before they actually occur.

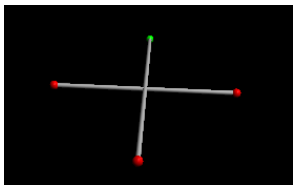

Figure 2: *An example collision between two edges*

Continuous Collision Detection (CCD) systems, if done properly, can assure that all collisions will be found and resolved accordingly. This is especially important for collision detection between infinitesimally thin lines, such as the on that can be observed in figure 2. Discrete collision detection systems, although perhaps more efficient, would be almost guaranteed to miss infinitesimally thin edge-edge collisions, seeing as they only check for collisions at current time steps (this is especially true at high velocities). Therefore CCD algorithms, such as the one examined in this paper, are instrumental in simulations that need to be of utmost accuracy.

2 Related Work

In related work, [Bridson et al.] describe an algorithm that efficiently and robustly processes collisions for cloth. The algorithm relies mostly on repulsion forces in order to reduce the number of collisions that are actually processed. Then, for certain cases where repulsion forces don't work (e.g. high velocity movement), geometric collision detection and response was used. They describe how to incorporate friction into their model, as well as an efficiency increasing trick by which they represent areas with many collisions as rigid bodies called "rigid impact zones".

[Spillman et al.] propose a way to represent deformable elastic rods, which would be an extension of the cable class we began to implement in this paper. They represented the elastic rods by a set of discretized elements, but calculated the energies associated with each element continuously, thereby decoupling the energy from the existing representation of the rods. The rods were one-dimensional, but had a configuration space of size two which included the orientation of the centreline along the rod. They then modelled bending and torsion along with kinetic, potential, and dissipation energy to accurately represent the behaviour of the elastic rods.

Continuous Collision Detection has been extended to rigid bodies as well. [Redon et al.] describe a way to efficiently detect collisions continuously on very complicated polyhedra. Their method involves finding the expected location of all rigid bodies at the next time step, and interpolating the movement over the space to find a volume of where the rigid body has been. Then, if any volumes intersect, those volumes are subdivided until it is found that they shared the space at different times, or that they did indeed collide [3].

3 Method

In three dimensions, collisions occur when two objects composed of a total of four points are coplanar (either two edges or a point and a plane). So in the case of systems composed solely of edges, it suffices to check when the points of two edges will become coplanar based on their current states. From this, the method behind edge-edge continuous collision detection follows quite logically.

First, each edge-edge combination is checked to find when they will become coplanar. Then, if they are expected to become coplanar within the next time step, the edges are checked to see if they will in fact intersect, and will not coincidentally be coplanar but nowhere near each other. Finally if the edges will become coplanar and will

intersect, it means that a collision will occur within the next time step. If a collision is found, an appropriate impulse is calculated and applied to either edge along the normal direction of the collision, which, in this case, is the normal direction of the plane defined by the two intersecting edges.

3.1 Finding Co-planarity

Every point in a system has a current position and a current velocity associated with each direction in the three dimensional world space (x, y, z). Therefore, assuming no change in velocity, we can find exactly where a point will be at any given time in the future (an example is shown below where A_{x0} represents the current x position). In between time steps, no forces are applied so this assumption can be made.

$$A_x = A_{x0} + \dot{A}_x t \tag{1}$$

Any plane in 3D space can be defined by three points (call them A, B, and C), and its normal can be defined by the cross product of two different lines in the plane (e.g. $\vec{AB} \times \vec{AC}$). If a fourth point (call it D) were also in the plane, then the line connecting it to any other point in the plane would have to be perpendicular to the normal of the plane. Therefore, the following must hold in order for four points to be coplanar

$$\vec{AD} \cdot (\vec{AB} \times \vec{AC}) = 0 \tag{2}$$

By plugging all twelve equations of the type seen in equation (1) (for A, B, C, D and all their x, y, z coordinates) into equation (2), a cubic equation for the expected time of co-planarity can be made

$$at^3 + bt^2 + ct + d = 0 \tag{3}$$

3.2 Solving the Cubic Formula

If a in the cubic equation (3) is zero, then the equation becomes quadratic and solving it is straight forward. However if a is not zero, then solving for t is not so trivial. To begin, we divide through the equation by a to remove the coefficient in front of t^3. we then replace t by $(x - \lambda)$, expand, and simplify to get

$$x^3+(b-3\lambda)x^2+(c-2b\lambda+3\lambda^2)x+(d-c\lambda+b\lambda^2-\lambda^3) = 0 \tag{4}$$

Then, by setting $\lambda = \frac{b}{3}$, the x^2 term is removed, leaving an equation of the form

$$x^3 + \frac{3c - b^2}{3}x - \frac{9cb - 27d - 2b^3}{27} = 0 \tag{5}$$

Setting a constant $p = \frac{3c-b^2}{3}$, and a constant $q = \frac{9cb-27d-2b^3}{27}$ reduces the equation to the following

$$x^3 + px - q = 0 \tag{6}$$

For ease in the following steps, we replace p by $Q = \frac{p}{3}$ and q by $R = \frac{q}{2}$

$$x^3 + 3Qx - 2R = 0 \tag{7}$$

Let M and N be arbitrary constants, where the following equation is satisfied

$$x^3 - M^3 = (x - M)(x^2 + Mx + M^2) \tag{8}$$

but since equation (7) has an x term, we need to account for it by adding a $N(x - M)$ term to (8)

$$x^3-M^3+N(x-M) = (x-M)(x^2+Mx+M^2+N) = 0 \tag{9}$$

Then, by rearranging the terms and solving for N, we get equations for M and N of the form

$$N = 3Q \tag{10}$$

$$M^3 + 3QM = 2R \tag{11}$$

As it turns out, the following equation satisfies (11)

$$M = \sqrt[3]{R + \sqrt{Q^3 + R^2}} + \sqrt[3]{R - \sqrt{Q^3 + R^2}} \tag{12}$$

Then, by plugging these values back into (9) we get

$$(x - M)(x^2 + Mx + M^2 + 3Q) = 0 \tag{13}$$

Where $x = M$ can be taken as a solution, and then $(x - M)$ can be factored out leaving

$$x^2 + Mx + M^2 + 3Q = 0 \tag{14}$$

Using the quadratic formula gives us the following values for x

$$x = -\frac{1}{2}M \pm \frac{1}{2}\sqrt{3}i\sqrt{M^2 + 4Q} \tag{15}$$

Defining

$$P = \sqrt[3]{R + \sqrt{Q^3 + R^2}} - \sqrt[3]{R - \sqrt{Q^3 + R^2}} = \sqrt{M^2 + 4Q} \tag{16}$$

Simplifies (15) to

$$x = -\frac{1}{2}M \pm \frac{1}{2}\sqrt{3}iP \tag{17}$$

Then, by defining $G = Q^3 + R^2$, $S = \sqrt[3]{R + \sqrt{G}}$, and $T = \sqrt[3]{R - \sqrt{G}}$, we get

$$M = S + T \tag{18}$$

$$P = S - T \tag{19}$$

Which, recalling $x = t - \frac{b}{3}$, gives us three equations for our roots

$$t_1 = -\frac{1}{3}b + S + T \tag{20}$$

$$t_2 = -\frac{1}{3}b - \frac{S + T}{2} + \frac{i\sqrt{3}(S - T)}{2} \tag{21}$$

$$t_3 = -\frac{1}{3}b - \frac{S + T}{2} - \frac{i\sqrt{3}(S - T)}{2} \tag{22}$$

3.3 Cube Roots of Complex Numbers

One part that is difficult to solve for on computers is finding the values for S and T, specifically if G is negative which would result in taking the cube root of a complex number. The complex number would be of the form $C_a + C_b i$ where $C_a = R$ and $C_b = \sqrt{-G}$. Then, by using Euler's Formula, the complex number can be represented using r and θ such that

$$r = \sqrt{C_a^2 + C_b^2} \tag{23}$$

$$\cos(\theta) = \frac{C_a}{r} \tag{24}$$

$$\sin(\theta) = \frac{C_b}{r} \tag{25}$$

Furthermore, Euler's Formula allows us to take the n^{th} root of a complex number by simply setting our new radius to $r_n = r^{\frac{1}{n}}$ and our new angle to $\theta_n = \frac{\theta}{n}$. It is important to note, however, that a complex number has n roots where n is which root is being taken (in this case three). The other roots have the same radius, but the angles are equal to the following

$$\theta_k = \frac{\theta}{n} + \frac{2\pi k}{n} \qquad \text{for } k = 0,..., n-1 \tag{26}$$

However, these extra roots are redundant, because their values are repeated in the equations for the t's. Then, by plugging the new radius and angle values back into equations (23), (24), and (25), we get our complex numbers for S and T which we then plug into the equations for our expected times of co-planarity.

3.4 Finding Intersections

Once we find when two edges will become coplanar, and it is found that it will happen within the next time step, we need to decide whether or not the edges actually intersect. In order to do so, we solve for f_1 and f_2 in the following equation

$$A + f_1\vec{AB} = C + f_2\vec{CD} \tag{27}$$

Where one edge runs from point A to point B and the other edge runs from point C to point D. The constants f_1 and f_2 represent how far along the axis of the edge the intersection occurs, where one signifies the end point of the edge (B for edge $\vec{AB}$ and D for edge $\vec{CD}$). The equation can be solved because each point and vector corresponds to three Cartesian coordinates (x, y, z), resulting in three equations for only two unknowns. Therefore the equations that don't include any zero's in the denominator should be used. Then, once f_1 and f_2 have been found, they must both lie between zero and one in order for a collision to have occurred.

3.5 Computing Impulses

To compute the impulse from the collision, the relative velocity in the normal direction of the edges at the point of contact needs to be found. Since the points of contact were found using the f_1 and f_2 values from (27), the relative velocity in the normal direction of the collision is computed in the following way

$$v_{rn} = (\vec{AB} \times \vec{CD}) \cdot ((1-f_2)\dot{C} + f_2\dot{D} - (1-f_1)\dot{A} - f_1\dot{B}) \tag{28}$$

The impulse is then found by scaling v_{rn} by the restitution (ϵ) and by the sum of the masses scaled by the squares of their respective distances from the collision

$$j = \frac{-(1+\epsilon)v_{rn}}{\frac{(1-f_1)^2}{m_a} + \frac{f_1^2}{m_b} + \frac{(1-f_2)^2}{m_c} + \frac{f_2^2}{m_d}} \tag{29}$$

Where the m's correspond to the masses of each of the particles (A, B, C, and D). The impulses are then applied to each of the particles (scaled by their distance from the collision) in order to correctly resolve the collisions.

3.6 Planar Collisions

A note about planar collisions is that edges that share a particle may collide if they become parallel and therefore occupy the same space. As a result, edges that share particles should be checked and handled appropriately using two dimensional continuous collision methods.

3.7 Cables

The foundation for a cable class was built, however complex equations solving for bending and torsion were not yet implemented due to time constraints. As of now, the cable class builds a cable connecting two points through a set of discretized springs. The number of springs used is an argument for the constructor of the class and can change based on the preferences of the user.

4 Results

The math from the equations follows quite logically, but when incorporated into an actual computer algorithm a few problems arose. Firstly, the algorithm runs in $O(kn^2)$ time, where n represents the number of edges and k represents the limit of iterations for detecting collisions (because in particularly messy situations the number of collisions may never resolve as edges bounce off of one another). Therefore, depending on the amount of computation per iteration, the algorithm can become very inefficient very quickly. This is especially true for algorithms that have not been optimized, such as the one that was used in this paper.

For relatively small numbers of edges (30 edges or less), highly detailed systems can be run at interactive rates. However, when the number of edges is significantly larger (such as in the basketball and basketball hoop test system where there are over 500 edges), the time it takes to check for all collisions along with with the time it takes to render the image severely slows the application in our implementation (with a frame rate of about 0.25 frames per second).

Secondly, the floating point errors in the computer tended to build up on themselves as more and more calculations were made. This problem could be solved by finding more direct mathematical solutions in the algorithm that would avoid this build up. Finding short cuts to the math equations would also be a way to increase the efficiency of a continuous collision detection algorithm.

Finally, there were certain scenarios where collisions were missed, possibly due to errors in the system or collision anomalies that were difficult to foresee or understand. With some more debugging and a deeper understanding of the process, these anomalies can be found and accounted for in a very strong CCD algorithm.

5 Conclusion

In conclusion, the logic behind CCD makes sense and when used properly can create systems with incredibly robust collision detection and reaction. The main issue with CCD, however, is that it is not very efficient compared to other collision detection schemes that currently exist (specifically discrete collision detection). With more research into this field, CCD algorithms may eventually reach interactive rates in highly complex systems. This would guarantee their use over discrete detection systems due to the unparalleled collision detection robustness that CCD produces.

References

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. 21*, 3 (July), 594–603.

REDON, S., KHEDDAR, A., AND COQUILLART, S. 2002. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum 21*, 3 (Sept.), 279–287.

SPILLMANN, J., AND TESCHNER, M. 2007. Corde: Cosserat rod elements for the dynamic simulation of one-dimensional

elastic objects. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '07, 63–72.