Speeding up line/line CCD checks with thickness

Last updated: April 22, 2011

1 Introduction: line/line CCD without thickness

Let us consider four points, P_1, P_2, Q_1 and Q_2, that are moving through space as a function of time. A necessary condition for the segments P_1P_2 and Q_1Q_2 to intersect (at some time t) is that P_1, P_2, Q_1 and Q_2 must be coplanar; or equivalently, that the volume of the tetrahedron $P_1P_2Q_1Q_2$ equals zero.[1] This can be expressed using the triple product of adjacent edges:

$$0 = 6\,V_{P_1P_2Q_1Q_2} = \overrightarrow{P_1P_2} \cdot (\overrightarrow{P_1Q_1} \times \overrightarrow{P_1Q_2}) = (\vec{P}_2 - \vec{P}_1) \cdot \big((\vec{Q}_1 - \vec{P}_1) \times (\vec{Q}_2 - \vec{P}_1)\big)$$

After some rearranging and a change of sign,[2] we obtain the more symmetric-looking

$$(\vec{Q}_1 - \vec{P}_1) \cdot \big((\vec{P}_2 - \vec{P}_1) \times (\vec{Q}_2 - \vec{Q}_1)\big) = 0 \tag{1.1}$$

(It's worth remembering that all of these vectors are implicitly functions of time.)

If we assume that the segment endpoints are moving linearly in time, we can express the triple product above as a cubic polynomial $T(t)$. Because of linearity, $\vec{P}_1(t) = \vec{P}_{10} + \vec{P}_{1\Delta}t$, where $\vec{P}_{10} = \vec{P}_1(0)$ and $\vec{P}_{1\Delta} = \vec{P}_1(1) - \vec{P}_1(0)$. Likewise, $\vec{P}_2(t) = \vec{P}_{20} + \vec{P}_{2\Delta}t$, etc. Eventually, after working through the vector math (or letting e.g. `LinearMotion` classes do it for you), you end up with a (scalar) 3rd-degree polynomial expression for the triple product.

That means that in most cases,[3] we can find the roots of $T(t)$ analytically and obtain the time(s) of possible impact. (This is what the current "cubic solver" CCD implementation does.)

We can also (subject to the same caveats) obtain the image of any time interval $[t_1, t_2]$ under T, i.e. determine the set of values for the triple product as the argument varies between t_1 and t_2. If the result (which is an interval[4]) includes zero,

$$T\big([t_1, t_2]\big) = [T_{\min}, T_{\max}] \ni 0 \tag{1.2}$$

then there *may* be collisions (one or more) between t_1 and t_2, but if not, then there are certainly no collisions within that time interval. We can use this condition in a recursive search to find the time of impact to an arbitrary precision. (This is what the current "interval arithmetic" CCD implementation does.)

In either case, once we have established the time of would-be impact, we can find $\vec{P}_1$, $\vec{P}_2$, $\vec{Q}_1$ and $\vec{Q}_2$ at that time, and use "regular" (non-continuous) collision detection to see whether a collision truly occurs and obtain the details.

[1] This condition is necessary but obviously not sufficient! The condition really checks whether the infinite lines defined by P_1P_2 and Q_1Q_2 are intersecting, but it's of course possible for the lines to be intersecting but for the segments to be positioned so they don't.

[2] The volume as expressed here has a sign that depends on the order of vertices, but the sign doesnt matter for our purposes.

[3] If the two segments remain coplanar throughout their motion, the polynomial will be identically zero and cant be usefully solved etc. We handle those cases specially.

[4] Because T is continuous.

2 Checking for line/line collisions with thickness

2.1 A generalization from the no-thickness case

Equations (1.1) and (1.2) obviously no longer hold if you want to assume that the segments P_1P_2 and Q_1Q_2 are actually cylinders with half-thickness $r_P > 0$ and $r_Q > 0$, respectively. The "thickened" lines (i.e. infinitely long cylinders) will come into contact *before* the volume of the tetrahedron reaches zero, at the point where the perpendicular distance between the lines is $d = r_P + r_Q$.

The volume of the tetrahedron at the point of contact can be calculated using the formula for the volume of the tetrahedron given two non-adjacent edges and their distance:

$$V_{\text{contact}}(t) = \frac{d \left\| \left(\vec{P}_2(t) - \vec{P}_1(t)\right) \times \left(\vec{Q}_2(t) - \vec{Q}_1(t)\right) \right\|}{6} \tag{2.1}$$

This is a standard formula,[5] but see sec. B.1 for a derivation.

A necessary condition for the lines P_1P_2 and Q_1Q_2 to be interpenetrating[6] is $\left|V_{P_1P_2Q_1Q_2}(t)\right| \leqslant \left|V_{\text{contact}}(t)\right|$. At the very moment of initial contact,[7] we will have

$$V_{P_1P_2Q_1Q_2}(t) \pm V_{\text{contact}}(t) = 0$$

$$(\vec{Q}_1 - \vec{P}_1) \cdot \left((\vec{P}_2 - \vec{P}_1) \times (\vec{Q}_2 - \vec{Q}_1)\right) \quad \pm \quad (r_P + r_Q) \left\| (\vec{P}_2 - \vec{P}_1) \times (\vec{Q}_2 - \vec{Q}_1) \right\| \quad = \quad 0 \tag{2.2}$$

(Every vector above is a function of t, but that takes up way too much space. Just imagine I wrote (t) after each of them.)

2.2 . . . but this isn't directly usable yet.

The term to the left of the $\pm$ in eq. (2.2) is just the left-hand side from eq. (1.1), which is well-behaved. But the presence of the vector norm operator on the right makes things very very nasty.

As far as I know, there's no way to solve eq. (2.2) analytically for t (a naive approach ends up with a 6th-degree polynomial). So a "cubic solver"-like approach based on this seems doomed. (Julien's "cubic solver with thickness" code tends to apply the no-thickness code first and then look for nearby solutions with thickness, but I'm not sure how easily that can be made fully robust.)

Doing what we do in the no-thickness "interval arithmetic" solver also won't work. That code base calculates the image of an interval through a polynomial by locating the zeros of the polynomial's derivative. Once again, I don't think we can solve the derivative.

But in order to do a search for the intersection time, as we do in the "interval arithmetic" approach, we don't necessarily need to calculate the *exact* image of the interval through the left-hand side of eq. (2.2). An exact image like in the no-thickness case is certainly nice, because it results in an optimal recursive search (we only

[5] See e.g. http://en.wikipedia.org/wiki/Tetrahedron#Distance_between_the_edges

[6] The condition actually checks for interpenetration of the infinitely long cylinders whose centers are defined by the segments and whose radii equal the half-thickness. Just like the tetrahedron check in the no-thickness case, this is is necessary but not sufficient for segment collision.

[7] In the common case where the motion isn't (nearly) coplanar throughout. The coplanar case needs to be handled separately.

descend the parts of the call graph where the condition is actually true, meaning no work is wasted.) But having an approximate result that is guaranteed to be a superset of the real image (but not too much larger!)[8] could still work well.

2.3 Making the math do what we need

To check for collisions between times t_1 and t_2, we'll be constructing an interval that is known to contain all of the values of $V_{P_1P_2Q_1Q_2}(t) \pm V_{\text{contact}}(t)$ for $t \in [t_1, t_2]$. (This is not strictly an image, since it will end up with too much in it.) Just like we did with eq. (1.2), we can check whether the result includes zero, and use that as a basis for a recursive search.

Starting from eq. (2.2), we make the following approximations:

- We treat each of the two terms separately, then add the two resulting intervals together. Note that this is an approximation in that it may artificially broaden the result.
- As in the no-thickness case, we express $V_{P_1P_2Q_1Q_2}(t)$ as a cubic polynomial $T(t)$. This allows us to find the image of $[t_1, t_2]$ under the polynomial and use it. (OK, this isn't actually an approximation.)
- We can calculate an upper bound C for the length of the cross product over the entire interval. Then $\pm V_{\text{contact}}(t)$ is guaranteed to fall within $[-(r_P + r_Q)C,\ (r_P + r_Q)C]$.

  All this gives us a collision check

  $$T\big([t_1, t_2]\big) \quad + \quad [-(r_P + r_Q)C,\ (r_P + r_Q)C] \quad \ni \quad 0$$

  which may include false positives (i.e. the check may be true when it shouldn't be), but which should definitely cover all of the actual collision cases.
- We can construct the upper bound C in several ways. In fact, we can actually calculate *several* different upper bounds, and use the smallest one we obtain as our C.
- For example, the cross product is equal to the product of the lengths of the segments times the sine of their angle. We can compute the maximum length of P_1P_2 during the time interval (let's call it $\lceil \text{len}_P \rceil$), do the same with Q_1Q_2 ($\lceil \text{len}_Q \rceil$), and place an upper bound of $\lceil \text{len}_P \rceil \cdot \lceil \text{len}_Q \rceil$ on the cross product, ignoring the angle. (This bound will work well if the segments are close to perpendicular, and may be very loose if they are close to parallel.)
- Alternately, we can express the x, y and z components of $\overrightarrow{P_1P_2} \times \overrightarrow{Q_1Q_2}$ as polynomials, then find the values reached by each polynomial for times in $[t_1, t_2]$. We can compute the maximum norm of the resulting triplet of intervals (by squaring each interval, summing the upper bounds and taking the square root), and use it as the upper bound C. (This bound will work well if the segments are not moving much.)
- We may want to think of some more upper bounds that work well in cases that are effective in cases that haven't yet been covered well...

Putting all of this together gives us the collision check

$$T\big([t_1, t_2]\big) \quad + \quad [-(r_P + r_Q)C,\ (r_P + r_Q)C] \quad \ni \quad 0 \tag{2.3a}$$

[8] See sec. A.1 for some detail on why too large is bad...

where

$$C = \min(C_1, C_2) \qquad \text{(more could be added)} \tag{2.3b}$$

$$C_1 = \lceil \mathrm{len}_P([t_1, t_2]) \rceil \ \lceil \mathrm{len}_Q([t_1, t_2]) \rceil \tag{2.3c}$$

$$C_2 = \left\lceil \left\| X([t_1, t_2]) \right\| \right\rceil = \sqrt{\lceil X([t_1, t_2])^2 \rceil} \qquad \text{with } X(t) = \overrightarrow{P_1 P_2} \times \overrightarrow{Q_1 Q_2} \tag{2.3d}$$

A Things we aren't doing

There are a few other general ideas that can be applied to introduce line thickness into the line/line calculations. I'm going to mention even the ones that turned out to be dead ends, in case they become relevant at some later point.

A.1 Old code: Evaluating the triple product using interval arithmetic

One possible approach—and in fact the approach that was used in the previous implementation—is to use interval arithmetic to simply evaluate the triple product in eq. (1.1). To do so, you express the values reached by each axis component (x,y,z) of $\overrightarrow{P_1P_2}$ over $[t_1, t_2]$ as an interval. You do the same with $\overrightarrow{Q_1Q_2}$ and $\overrightarrow{P_1Q_1}$. You broaden each interval to account for thickness. When you simply plug these intervals into the usual triple product math, out falls an interval result, and you can check if it includes zero.

The problem with this approach is that the interval formulation of the triple product can produce results that are over-broad, and so include 0 in the interval far more often than necessary. The biggest reason is that the "phase" information—the relationships between the motions of each point axis—is not being captured in any way, so the motion in each axis is treated as being completely independent. This broadness causes far too much recursion during the search and awful performance. Changing line/line collisions to the approach described in this document resulted in an >1000x speed improvement for certain bad cases!

It's also important to note that **we still use triple product calculations based on "raw" interval arithmetic in more than a few places.** We should really **attempt to remove** every occurence of the `Interval_nD` class from the collision code...

A.2 Other ideas that may or may not be useful

A.2.1 Direct evaluation of C

When we attempt to figure out the upper bound on the cross product length for use in eq. (2.3), we're currently completely ignoring a frontal attack that should be able to give us a tight bound.

The cross product $\vec{P} \times \vec{Q}$ can be expressed as a 3-vector of quadratic polynomials in t. It follows that the *square* of the cross product can be expressed as a single 4th-degree polynomial. It should be possible to find images of intervals through this polynomial directly! (In order to compute the image, we do need to find extrema (minima/maxima) of the polynomial by locating the zeros of its derivative. But since the derivative is cubic, that should be possible.)

Thus we can find the upper bound for the square of the cross product (it's just the upper end of the interval we get for the image). The upper bound C for the *length* of the cross product is the square root of that.

Mostly, I didn't do it that way because I was a bit wary (perhaps unwarrantedly) of the robustness of using a cubic solver in this way. But if we find that the "multiple upper bound" approach spends too much work looking for bounds that aren't good enough, we should possibly revisit this.

A.2.2 Polynomials with interval coefficients

Using intervals to make the segment endpoints "thick", as described in sec. A.1, can be developed in another possible way. Instead of evaluating the triple product using intervals over the entire time interval $[t_1, t_2]$, you can keep the time-dependence of each point as is, and just add the appropriate thickness interval $[-r, r]$ to each of the vectors. If you turn the resulting triple product into a polynomial along the lines of eq. (1.2), you end up with a polynomial whose coefficients are intervals rather than scalars.

You can do a number of useful things with such polynomials: evaluate, take derivatives, find zeros using the standard formulas,[9] etc. You should also be able to find minima/maxima of the polynomials and construct images of interval arguments under the polynomial.

The big problem is mostly that all of the interval stuff makes things a bit exotic and harder to comprehend intuitively, so I find I have to think hard about things that really ought to be pretty simple. That's primarily why I gave up this line of attack.

B Derivations

B.1 Tetrahedron volume from non-adjacent edge length and distance

Consider a tetrahedron $ABCD$, and pick a pair of non-adjacent edges, let's say AB and CD. To find the distance between the straight lines through AB and CD, we start by constructing a line that is perpendicular to both. The direction of this straight line is the same as the direction of the cross product $\overrightarrow{AB} \times \overrightarrow{CD}$; the unit vector in that direction is

$$\hat{n} = \frac{\overrightarrow{AB} \times \overrightarrow{CD}}{\left\| \overrightarrow{AB} \times \overrightarrow{CD} \right\|}$$

The distance between the lines AB and CD is the same as the projection onto $\hat{n}$ of the distance from any point on AB to any point on CD, which is

$$\begin{aligned}
d &= \left| \overrightarrow{KL} \cdot \hat{n} \right|, \qquad \text{where} \quad \vec{K} = \vec{A} + \lambda \overrightarrow{AB} \ \text{ and } \ \vec{L} = \vec{C} + \mu \overrightarrow{CD} \\
\pm d &= \overrightarrow{KL} \cdot \hat{n} = \\
&= (\vec{C} - \vec{A}) \cdot \hat{n} + \mu (\overrightarrow{CD} \cdot \hat{n}) - \lambda (\overrightarrow{AB} \cdot \hat{n}) = \\
&= \overrightarrow{AC} \cdot \frac{\overrightarrow{AB} \times \overrightarrow{CD}}{\left\| \overrightarrow{AB} \times \overrightarrow{CD} \right\|} + \mu \left(\overrightarrow{CD} \cdot \frac{\overrightarrow{AB} \times \overrightarrow{CD}}{\left\| \overrightarrow{AB} \times \overrightarrow{CD} \right\|} \right) - \lambda \left(\overrightarrow{AB} \cdot \frac{\overrightarrow{AB} \times \overrightarrow{CD}}{\left\| \overrightarrow{AB} \times \overrightarrow{CD} \right\|} \right) = \\
&= \frac{\overrightarrow{AC} \cdot \left(\overrightarrow{AB} \times \overrightarrow{CD} \right)}{\left\| \overrightarrow{AB} \times \overrightarrow{CD} \right\|} \\
d &= \left| \frac{\overrightarrow{AC} \cdot \left(\overrightarrow{AB} \times \overrightarrow{CD} \right)}{\left\| \overrightarrow{AB} \times \overrightarrow{CD} \right\|} \right| = \frac{\left| \overrightarrow{AC} \cdot \left(\overrightarrow{AB} \times \overrightarrow{CD} \right) \right|}{\left\| \overrightarrow{AB} \times \overrightarrow{CD} \right\|}
\end{aligned}$$

[9] The zeros you obtain in this way will, of course, be *intervals*, which complicates finding minima/maxima.

Note that the distance between the lines doesn't depend on which K and L we picked, which is what's expected. The triple product in the numerator is six times the volume of the tetrahedron (see eq. (1.1)), so

$$V_{ABCD} = \frac{1}{6}\left|\overrightarrow{AC}\cdot\left(\overrightarrow{AB}\times\overrightarrow{CD}\right)\right| = \frac{d\left\|\overrightarrow{AB}\times\overrightarrow{CD}\right\|}{6} \tag{B.1}$$

which is where eq. (2.1) came from.