Numerically Robust Continuous Collision Detection for Dynamic Explicit Surfaces *

Tyson Brochu† Robert Bridson‡

Abstract

We present a new, provably robust method for continuous collision detection for moving triangle meshes. Our method augments the spatial coordinate system by one dimension, representing time. We can then apply numerically robust predicates from computational geometry to detect intersections in space-time. These predicates use only multiplication and addition, so we can determine the maximum numerical error accumulated in their computation. From this forward error analysis, we can identify and handle degenerate geometric configurations without resorting to user-tuned error tolerances.

1 Introduction

Beginning with the cloth simulation of Bridson et al. [1], physics-based animation with triangle meshes that are guaranteed not to self-intersect has proven valuable. More recently Brochu [2] extended this work to handle adaptive triangle meshes undergoing topological changes (such as merging and pinching) while retaining the guarantee of no self-collision, which now paves the way for interesting applications beyond cloth: indeed, virtually any dynamic surface evolution (whether closed or open) can be effectively handled by Brochu's method. (In comparison, the popular level set method which ably handles deformation and topology change cannot easily handle very thin surfaces, open surfaces, or surfaces which are not supposed to change topology.)

*UBC Computer Science Technical Report TR-2009-03

†e-mail: tbrochu@cs.ubc.ca

‡e-mail: rbridson@cs.ubc.ca

This report introduces a new, provably robust continuous collision detection method useful for tangle-free mesh animation. While the current state-of-the-art in cloth simulation usually works well, it requires some degree of tuning for the various error tolerances; particularly for Brochu's adaptive surfaces which present a much more severe challenge to CCD than simple cloth simulations, this tuning can be at times difficult. We provide a provably robust alternative, adapting computational geometry methods for detecting intersections to space-time meshes.

2 Continuous Collision Detection

For the purposes of this report, Continuous Collision Detection (CCD) is the process of detecting if a mesh moving between initial and final configurations over a time step at some point in between comes into contact with itself (and, naturally, finding when and where these collisions occur). We generally assume the initial configuration is intersection free, so this can be reduced to two primitive tests: does a moving point hit a moving triangle, or does a moving edge hit another moving edge? For algorithms which require non-self-intersecting meshes at all time steps, it is critical that CCD be *robust*: no false negative (a missed collision that could lead to self-intersection) is permitted. Accuracy, in the sense of reducing the number of false positives (near misses that are flagged as collisions), is also desirable but not as high a priority.

The collision queries (point vs. triangle, edge vs. edge) aren't well-posed until the exact nature of the motion over the time step is specified. For example, a rigid body motion with constant translational and angular velocity would lead to points following "screw" paths, and edges

sweeping out helical patches. Zhang et al. [7] provide a recent example of solving CCD under this assumption for articulated rigid models, albeit with the requirement of tuning of a distance threshold for the Conservative Advancement algorithm; robustly solving for an exact collision time and location appears difficult. However, we are here concerned with more general deformable models, where there isn't an obviously "correct" intermediate motion in the time step.

Provot [5] made the simplifying assumption that mesh vertices move on straight line, constant velocity paths, and the edges and triangles between them are linearly interpolated at every intermediate time. The times at which a point and a triangle or two edges become coplanar (a necessary condition for collision) are the roots of a simple cubic equation in this model, which is simple enough to solve; at these times, the proximity of the elements can be evaluated to determine if a collision occurs. Bridson et al. [1] introduced error tolerances in the cubic solver, and an error tolerance on proximity testing at the coplanarity times, to account for rounding errors in the process, making it robust enough to run many large-scale tangle-free cloth simulations.

However, this solution is less than satisfactory, as it again relies on tuned error tolerances. If too small, collisions may be missed but if too large the simulation may grind to a halt under the weight of false positives. For more challenging problems than cloth, such as Brochu's work on adaptive meshes undergoing topological changes [2], this tuning can be difficult to get right. A conservative forward error analysis appears daunting. We thus propose an alternative model of the intermediate motion which admits provably robust CCD without need for any tolerances.

2.1 CCD in Two Dimensions

It is simpler to first understand the new model reduced to two spatial dimensions, where CCD is defined as detecting if a moving point collides with a moving edge in the plane. By viewing time as just a third axis, it's easy to see the trajectory of the point as a curve in 3D and the trajectory of the edge as a quadrilateral patch in 3D. The Provot model uses a linear segment and a bilinear patch, resulting in a quadratic to solve for intermediate collinearity times, and all the attendant difficulties with rounding error.

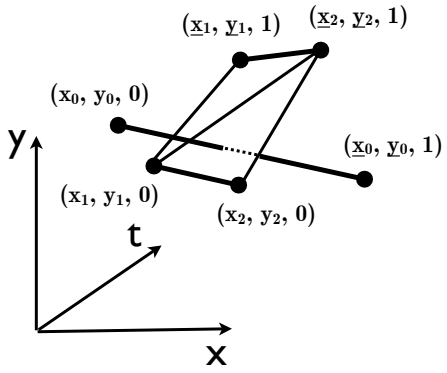

Figure 1: *To test a moving point colliding with a moving edge in 2D, we view it as intersection in 3D and discretize the swept trajectory of the edge with two triangles.*

Instead we propose discretizing the trajectory of the edge with two triangles, breaking up the quadrilateral patch along a diagonal: see figure 1. This further reduces the CCD problem to testing a specially structured line segment against two separate triangles in three dimensions, which is much more tractable for rounding error analysis.

Let (x_0, y_0) and $(\bar{x}_0, \bar{y}_0)$ be the coordinates of the point at the start ($t = 0$) and end ($t = 1$) of the time step, or in our 3D view $(x_0, y_0, 0)$ and $(\bar{x}_0, \bar{y}_0, 1)$. Let the edge's endpoints be (x_1, y_1) and (x_2, y_2) at $t = 0$, and $(\bar{x}_1, \bar{y}_1)$ and $(\bar{x}_2, \bar{y}_2)$ at $t = 1$. Picking one diagonal arbitrarily, the two triangles representing the edge's swept trajectory are $(x_1, y_1, 0) : (x_2, y_2, 0) : (\bar{x}_2, \bar{y}_2, 1)$ and $(x_1, y_1, 0) : (\bar{x}_1, \bar{y}_1, 1) : (\bar{x}_2, \bar{y}_2, 1)$.

Focus on the first triangle. We'll set up the intersection problem as a linear system, looking for $(s, t, \alpha, \beta, \gamma)$ satisfying:

$$
\begin{aligned}
s x_0 + t \bar{x}_0 + \alpha x_1 + \beta x_2 + \gamma \bar{x}_2 &= 0 \quad (1)\\
s y_0 + t \bar{y}_0 + \alpha y_1 + \beta y_2 + \gamma \bar{y}_2 &= 0 \quad (2)\\
t + \gamma &= 0 \quad (3)\\
s + t + \alpha + \beta + \gamma &= 0 \quad (4)\\
s + t &= 1 \quad (5)
\end{aligned}
$$

The last two equations define the pair (s, t) as barycentric

coordinates along the line through the point's trajectory and the triple $(-\alpha, -\beta, -\gamma)$ as barycentric coordinates in the plane of the triangle. The first three equations then express the intersection of the points' trajectory with the triangle, in x, y, and t in order. There is an intersection if and only if there is a solution to this linear system where s and t are both non-negative, and α, β, and γ are all non-positive—i.e. the intersection lies within the triangle and in the time interval $[0, 1]$.

We take the usual Computational Geometry approach to determining the signs of components of the solution. Assuming the matrix is invertible for now, Cramer's rule expresses the components of the solution as ratios of determinants formed from the matrix and right-hand side. Since we only care about signs, we can take out the common denominator to get the solution scaled by the determinant of the matrix. For example, the scaled s is the determinant of the matrix with its first column replaced by the right hand side:

$$\hat{s} = \det \left(\begin{array}{c|cccc} 0 & \bar{x}_0 & x_1 & x_2 & \bar{x}_2 \\ 0 & \bar{y}_0 & y_1 & y_2 & \bar{y}_2 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{array} \right) \tag{6}$$

Similarly $\hat{t}$, $\hat{\alpha}$, $\hat{\beta}$, and $\hat{\gamma}$ can be expressed as 5×5 determinants. These determinants can be substantially simplified by cofactor expansion. (We note these determinants are often given geometric meaning in terms of signed volumes of tetrahedra, or triple products of certain vectors, but extending this to higher dimensions taxes the intuition.) These can be computed using only multiplication, addition and subtraction—no square or cube roots or even divisions are needed. If using fixed-point or integer coordinates with p bits, the exact answers only require $2p + 4$ bits. We instead use floating point arithmetic, and carry out the forward error analysis to conservatively and provably bound the error as a small multiple of unit round-off times the permanent of the absolute values of the matrix entries; if the computed result is further from zero than this error bound, we are guaranteed to have the correct sign, but if smaller we clamp it to zero (thus marking the sign as indeterminate and possibly perturbing a near miss into a collision). Shewchuk's adaptive precision approach would be a worthwhile extension to guaranteeably but efficiently compute the correct sign when the computed value is smaller than the error bound [6].

It is a common occurence in a typical simulation that we will end up a singular matrix, where both $\hat{s}$ and $\hat{t}$ or all of $\hat{\alpha}$, $\hat{\beta}$, and $\hat{\gamma}$ evaluate to zero (or were clamped to zero due to conservative error analysis): geometrically this corresponds to a degenerate configuration, such as parallel motion or a zero-length edge. If using exact evaluation of the determinants, Simulation of Simplicity [3] could be used to resolve the degeneracy in a fully automatic way; since we instead use approximate (but conservatively bounded) floating point arithmetic, this is not available to us. We instead handle the degenerate case robustly by checking the moving point's path against both "time" edges of each triangle. These smaller segment vs. segment checks begin by checking coplanarity with the appropriate already-evaluated determinant. If the two edges are not coplanar in 3D they cannot collide, and otherwise we check for collision by projecting the segments onto the x–t and the y–t planes and running similarly robust 2D segment vs. segment tests. There is a collision if and only if both 2D tests report a collision. If either of the 2D tests hits a degeneracy, we again reduce the dimension by checking the endpoints against the other segment, and so on.

Additional information about the collision is easily collected from the barycentric coordinates, once computed. For example, the time of collision (between 0 and 1) is simply the coordinate t; the normal vector at the collision is either the $t = 0$ edge's normal or the $t = 1$ edge's normal depending on which triangle was hit; the relative normal velocity at the collision is defined from the point's velocity and the velocity of the endpoint of the edge that is entirely contained in the colliding triangle. (Special cases must be made for degenerate situations, sometimes leading to an ambiguity which we arbitrarily resolve: e.g. a degenerate edge has no defined normal.)

2.2 CCD in Three Dimensions

To handle CCD in two dimensions, we essentially built a triangle mesh stretching between the initial and final configurations in 3D space-time, and then ran robust self-intersection specialized to this type of mesh construction. We do exactly the same in three dimensions, building a

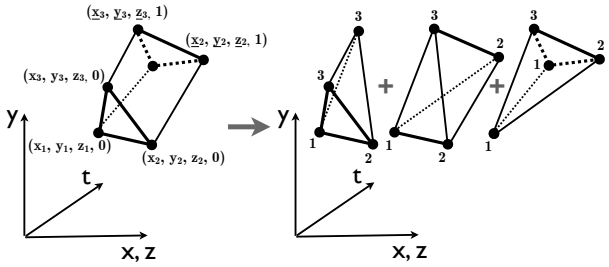

Figure 2: *For collisions with a triangle in 3D, we discretize the triangular space-time prism of the triangle's swept trajectory with three tetrahedra as shown (with x and z axes projected together).*

tetrahedral mesh stretching the the initial and final configurations in 4D space-time. We model a moving point by sweeping out a straight line segment from $t = 0$ to $t = 1$ and a moving edge with two triangles as before. A moving triangle with vertices labeled 1, 2, and 3 sweeps out a triangular prism discretized into three tetrahedra, as in figure 2:

$$\mathbf{x}_1 : \mathbf{x}_2 : \mathbf{x}_3 : \bar{\mathbf{x}}_3$$
$$\mathbf{x}_1 : \mathbf{x}_2 : \bar{\mathbf{x}}_2 : \bar{\mathbf{x}}_3$$
$$\mathbf{x}_1 : \bar{\mathbf{x}}_1 : \bar{\mathbf{x}}_2 : \bar{\mathbf{x}}_3$$

where $\mathbf{x}_i = (x_i, y_i, z_i, 0)$ and $\bar{\mathbf{x}}_i = (\bar{x}_i, \bar{y}_i, \bar{z}_i, 1)$.

Note that to avoid cracks in the tetrahedral space-time mesh, a consistent choice of diagonals for the triangles must be made: in our code, we first sort the vertices by index before constructing the triangles or tetrahedra in a call to make sure this happens.

With this discretization, the moving edge vs. moving edge tests is reduced to four triangle vs. triangle intersection tests in 4D (each edge sweeps out two triangles), and the moving point vs. triangle test is reduced to three segment vs. tetrahedron intersection tests in 4D. The intersection of two such 4D simplex primitives can be described with six barycentric coordinates that solve a linear system analagous to the one presented above, robustly evaluated in exactly the same fashion. Degeneracies are also dealt with in exactly the same fashion: for example, if a segment vs. tetrahedron test is degenerate, we replace it with several segment vs. triangle tests (using the triangular faces of the tetrahedron) which, if coplanar, can be projected onto x–y–t, x–z–t, and y–z–t spaces to be resolved in lower dimensions.

In terms of performance, we have not yet properly profiled and optimized the code, but find it already is comparable to a typical cubic-root-finding based method. Many of the determinant calculations permit better pipelining and vectorization than the root-solver, so we are hopeful superior performance will be obtainable.

2.3 Collision Resolution

We implemented our new CCD algorithm with double precision arithmetic and conservative error bounds, providing what we believe is the first practical robust CCD code for deforming meshes which doesn't require any user-tuned tolerances. (We also implemented and tested an exact integer coordinate version for comparison, but in 3D with 64-bit integers the available precision for the input coordinates is only 19 bits or roughly six decimal digits, which is somewhat limiting for more challenging applications.) However, at this point we can only guarantee that we will detect all collisions: this says nothing about *resolving* them, i.e. finding a physically consistent adjustment to the final configuration of the mesh that eliminates all collisions. Indeed, taking into account that the final positions are quantized to a finite number of bits of precision, provably robust but physically plausible collision resolution probably will involve the solution of a rather daunting large-scale integer programming problem. As an example of the complications involved, simple transformation of an intersection-free mesh with a rotation matrix can potentially create self-intersection once the results are rounded or quantized.

We instead use the more heuristic approach to collision resolution initiated by Provot [5] and later extended by Bridson et al. [1] and Harmon et al. [4]. As in the latter work, we gather overlapping collisions into "impact zones" and solve for the set of impulses which will resolve all collisions in the zone simultaneously. This involves solving a potentially extremely rank-difficient matrix. We have tried several different linear solvers to deal with this issue, including dense versions of SVD, as well as sparse

iterative solvers such as MINRES CR and CGNR and have found that these iterative sparse solvers work well in the presence of matrices which are singular or nearly singular.

References

[1] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 21(3):594–603, 2002.

[2] T. Brochu. Fluid animation with explicit surface meshes and boundary-only dynamics. Master's thesis, University of British Columbia, 2006.

[3] H. Edelsbrunner and E. P. Mucke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph*, 9:66–104, 1990.

[4] D. Harmon, E. Vouga, R. Tamstorf, and E. Grinspun. Robust treatment of simultaneous collisions. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 27(3):1–4, 2008.

[5] X. Provot. Collision and self-collision handling in cloth model dedicated to design garment. *Graphics Interface*, pages 177–89, 1997.

[6] J. R. Shewchuk. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry*, 18(3):305–363, October 1997.

[7] X. Zhang, S. Redon, M. Lee, and Y. J. Kim. Continuous collision detection for articulated models using taylor models and temporal culling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3):15, 2007.