

Precise Generalized Contact Point and Normal Determination for Rigid Body Simulation

ABSTRACT

Modeling contact for rigid body simulation requires accurate determination of time of contact, contact points, and contact normals. Existing collision detection methods for rigid body simulation can be grouped into one of three categories: convexity-based discrete methods, *a posteriori* discrete methods, and continuous methods. Our method combines the advantages of all three of these types by operating on arbitrary geometric representations, running in asymptotic linear time in the number of polyhedral features (for non-convex representations), having a parameterizable precision (a variation is guaranteed to miss no collisions), and avoiding difficult to implement simplex/simplex tests that are frequently not numerically or geometrically robust. Our algorithm is demonstrated on a pathological example involving both polyhedra and polygon soups.

1. INTRODUCTION

Algorithms for modeling contact in rigid body simulation take a set of contact points and normals as input and produce a set of forces or impulses that handle impacts or treat resting contact. However, the difficulty and importance of determining these contact point and normal inputs accurately is often overlooked; this determination is both computationally intensive and plagued by numerical and geometric degeneracies. Many researchers use approximate contact points and normals that are determined without finding the exact time-of-impact. For some applications of rigid body simulation (e.g., animation, games), these approximations may not be deleterious; for applications that place importance on accurate simulation, approximations may be unacceptable.

Approximation of the true contact points and normals can lead to interpenetration, numerical and dynamic instability, and undesirable energy drain from the simulated system [6]. The energy drain effect can also appear when the geometry of rigid bodies with curved surfaces is approximated with polyhedra. We introduce a method for computing time of contact (TOC) and contact points and normals that addresses the above problems. Our algorithm is general with respect to the geometric representation: geometries may be polyhedra, polygon soups, implicit surfaces, parametric surfaces, constructive solid geometry, etc. and can be

non-convex. The only requirements are that a representative subset of points on the surface of the geometry be readily obtainable and that fast intersection queries between the geometry and either a oriented-bounding box (OBB) or a line segment are supported. Our algorithm performs *continuous collision detection*; it is able to determine with certainty whether a pair of bodies contact over a given time interval. Additionally, our method is fast: it runs in time which is linear in the number of polyhedral features, even for non-convex polyhedra. We illustrate this algorithm on a pathological example: a fast moving projectile impacting a thin body.

2. BACKGROUND

Collision detection methods can be broadly grouped, with respect to rigid body simulation, into three categories: *discrete convexity-based methods* (Section 2.1), *discrete a posteriori methods* (Section 2.2), and *continuous collision detection methods* (Section 2.3).

2.1 Discrete convexity-based methods

Convexity-based methods determine whether one (or more) pairs of bodies are intersecting at an instant in time; they are termed *discrete* because, if the bodies follow a trajectory, intersection testing must be performed at discrete points in time. Convexity-based methods can determine the time of contact by using numerical root-finding type approaches (e.g., [17, 2, 22], etc.) until a query reports that two bodies are either separated or interpenetrating smaller than some tolerance. If a pair of geometries are polyhedra with M and N features, the worst-case running time for these methods is $\Theta(M + N)$. When temporal coherence is exploited, these algorithms can achieve near constant-time performance; the coherence caches closest features, which are used to determine the contact points and normals.

The convexity-based algorithms include the GJK algorithm [10, 4, 21], the Lin-Canny [15] and V-Clip [16] algorithms, and the method of Chung and Wang [8]. Only the GJK-based algorithms can be used on geometric representations other than polyhedra; however, even these algorithms converge only asymptotically on non-polyhedral representations [9].

The only means that the convexity-based algorithms have for handling objects with non-convex geometries is decomposition into convex pieces. Even a decomposition into a minimal number of pieces can result in N^2 pieces for a polyhedron with N vertices [7]; collision detection for non-convex polyhedra using decompositions is thus too slow for many interactive applications. Non-manifold geometries (e.g., polygon soups) are unsupported by convexity-based methods.

2.2 Discrete *a posteriori* methods

We denote methods that determine contact points and normals

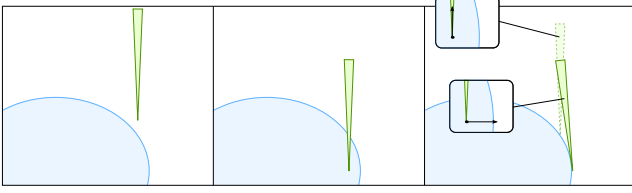


Figure 1: Three successive frames (left to right) depicting a collision. The configuration of the bodies in the rightmost frame is used to determine the contact point and normal; using the minimum penetration depth causes the spike to be pushed outward laterally (shown in the rightmost frame), when it should be pushed out vertically.

after interpenetration has occurred *a posteriori* methods. The *a posteriori* methods, which are discrete methods by their nature, avoid the significant computation requirements of finding the exact time of contact, but only guess at the contact points and normals (an incorrect guess is seen in Figure 1); larger step sizes result in less accurate guesses. The *a posteriori* methods include the work of Moore and Wilhelms [17], Kim et al. [14], Guendelman et al. [12], Hasegawa and Fujii [13], Yamane and Nakamura [23], and Redon and Lin [19]. The approaches of Kim et al., Guendelman et al., and Redon and Lin treat non-convex geometries, though polygon soups are not supported.

2.3 Continuous collision detection

Continuous collision detection methods test whether one (or more) pairs of bodies contact over a given time interval, rather than at a discrete instant of time. Canny [5] first proposed to represent polygonal trajectories (i.e., both translation and rotation) by polynomials, and to express collision terms of contact functions defined over these polynomials. Portions of time in which collisions can occur are found by searching for intervals of time bracketed by the zeros of contact functions. Numerous researchers have proposed methods to perform continuous collision detection at rates competitive with, though invariably slower than, discrete collision detection algorithms. Most relevant to this paper are methods for rigid bodies governed by classical mechanics, including the method of Redon et al. [18] and the work of Zhang et al. [24]. This latter work, in particular, is similar to that introduced in this paper; Zhang et al. use comparable bounds on rigid body motion. However, the work of Zhang et al. is restricted to polyhedral models and requires decomposition into convex polyhedra to handle non-convex geometries. Decomposition into a minimal number of convex pieces in NP-hard [1], and approximate algorithms can lead to potentially many unnecessary convex pieces.

3. METHOD

We determine contact data and times over a time interval $[t_0, t_f]$ by computing the TOC of points sampled from the two bodies; we assume that the bodies are disjoint at t_0 and are intersecting at t_f . The times of impact of the sampled points are computed rapidly using the dynamic states of the bodies at t_0 and t_f ; it is guaranteed that the first time of contact (and associated contact data) is found between a pair of bodies up to a given tolerance.

Algorithm 1 computes the time of contact, contact point, and contact normal for two rigid bodies with arbitrary geometries. (A simple modification can return multiple contact points; all contacts within small tolerance of the first TOC can be returned.) The time

of contact between one solid and a representative subset of the points on the surface of the other solid is determined. The choice of points depends on the primitives involved although, most generally, a random sampling could be used. Algorithm 2, presented in the next section, describes how to compute the time of contact between a point on a rigid body and a stationary solid; the following section describes the extension to a moving solid.

Algorithm 1

DETERMINE-CONTACT($P_A, P_B, v_A(t_0), \omega_A(t_0), v_B(t_0), \omega_B(t_0), t_0, t_f$): given rigid bodies A and B with geometries P_A and P_B , linear and angular velocities $v_A(t), \omega_A(t), v_B(t)$ and $\omega_B(t)$ of the bodies at time t_0 , and interval of integration $[t_0, t_f]$, determines the time of contact, contact point, and contact normal between A and B .

```

 $t_i \leftarrow \infty$ 
for all points  $p_A(t)$  on surface of  $P_A$  do
   $\{t_{min}, p(t_{min}), \hat{n}\} \leftarrow \text{DETERMINE-TOC}(\dots)$ 
  if  $t_{min} < t_i$  then
     $t_i \leftarrow t_{min}$ 
     $c \leftarrow p(t_{min})$ 
     $\hat{n}_c \leftarrow \hat{n}$ 
for all points  $p_B(t)$  on surface of  $P_B$  do
   $\{t_{min}, p(t_{min}), \hat{n}\} \leftarrow \text{DETERMINE-TOC}(\dots)$ 
  if  $t_{min} < t_i$  then
     $t_i \leftarrow t_{min}$ 
     $c \leftarrow p(t_{min})$ 
     $\hat{n}_c \leftarrow \hat{n}$ 
return  $\{t_i, c, \hat{n}_c\}$ 

```

3.1 Determining intersection of point trajectory with a stationary solid

Intersection of a parametric curve with a solid requires algorithms equivalent to numerical root finding. Unfortunately, root finding algorithms work poorly at determining whether a root exists within an interval. Our application requires identification a root within an interval to compute the number of intersections between the curve and the solid. However, we make use of the fact that the motion free of external torques and forces between integration steps in reliably computing such intersections. The bodies must be treated in this way in order for a search between times t_0 and t_f , which proceeds at a fine temporal granularity, to be consistent with the integrator steps with $\Delta t = t_f - t_0$.

In general, points on a rigid body follow a curved path; the following equation determines the position of a point $p(t)$ on a moving rigid body \mathcal{B} :

$$p(t) = x_{\mathcal{B}}(t) + \mathbf{R}_{\mathcal{B}}(t)\mathbf{u}$$

where $x_{\mathcal{B}}(t)$ is the position of the body's center-of-mass, $\mathbf{R}_{\mathcal{B}}(t)$ is the rotation matrix of \mathcal{B} , and \mathbf{u} is the vector from the center-of-mass to the point in \mathcal{B} 's frame. The position of the point over a time interval is determined by the solution to the time derivative of the above equation. The time derivative of this equation is given below, using the identity $\dot{\mathbf{R}}(t) = \tilde{\omega}(t)\mathbf{R}(t)$:

$$\dot{p}(t) = \dot{x}_{\mathcal{B}}(t) + \tilde{\omega}_{\mathcal{B}}(t)\mathbf{R}_{\mathcal{B}}(t)\mathbf{u} \quad (1)$$

where $\dot{x}_{\mathcal{B}}(t)$ is the velocity of \mathcal{B} 's center-of-mass and $\tilde{\omega}_{\mathcal{B}}(t)$ is the antisymmetric matrix formed using the angular velocity of the body in the world frame. We note that integrating the position of $p(t)$ requires integrating $\mathbf{R}_{\mathcal{B}}(t)$ as well; Baraff [3] describes how to perform this integration with quaternions to reduce drift.

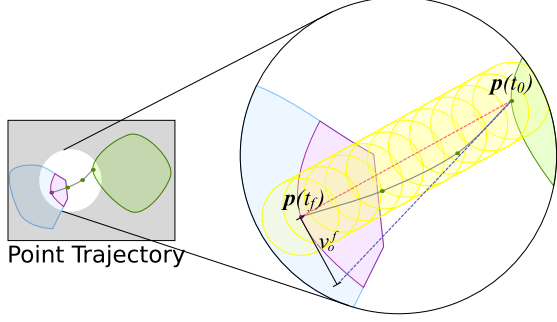


Figure 2: Depiction of the bounds (shaded in yellow) of a trajectory of a point on a rigid body; the true trajectory is drawn in black, and the linearization of the trajectory by connecting end-points is drawn in red, the linearization by expansion around $\mathbf{p}(t_0)$ is drawn in blue.

The magnitude of $\dot{\mathbf{p}}(t)$ remains constant over $[t_0, t_f]$ because the linear and magnitude of the angular velocities remain constant over this interval¹. The rotation matrix in (1) does not change the magnitude of the angular contribution to point velocity. Therefore, the distance that the particle moves over the time interval is equal to the magnitude of the point velocity times the difference in time:

$$\left\| \int_{t_0}^{t_f} \dot{\mathbf{p}}(t) dt \right\| \leq \|\dot{\mathbf{p}}(t_0)\| (t_f - t_0) \quad (2)$$

This equation permits us to evaluate the quality of the approximation of $\mathbf{p}(t)$ over $[t_0, t_f]$ by a line segment; if the approximation is good, then

$$\nu_0^f = \|\dot{\mathbf{p}}(t_0)\| (t_f - t_0) - \|\mathbf{p}(t_f) - \mathbf{p}(t_0)\|$$

will be small. This equation also enables us to bound the values that $\mathbf{p}(t)$ can take over $[t_0, t_f]$. As seen in Figure 2, $\mathbf{p}(t)$ can deviate from the line segment $\overline{\mathbf{p}(t_0)\mathbf{p}(t_f)}$ by at most $\nu_0^f/2$ in any direction over $[t_0, t_f]$. We can thus bound $\mathbf{p}(t)$ by an ellipsoid over this time interval. Alternatively, as used in the algorithm itself, we can bound $\mathbf{p}(t)$ by an oriented bounding box (OBB), which generally permits fast intersection testing with the solid at the expense of looser bounds for $\mathbf{p}(t)$.

3.2 Intersection of point on a rigid body with a moving solid

The preceding intersection of a point trajectory with a stationary body can be extended to consider intersection of trajectory with a moving body through an appropriate change in coordinate system. If the solid \mathcal{S} is moving with linear velocity $\dot{\mathbf{x}}_S(t)$ and angular velocity $\dot{\boldsymbol{\omega}}_S$, then \mathcal{S} can be treated as stationary by incorporating its velocity into that of $\mathbf{p}(t)$. The position of the tracked point in the frame of \mathcal{S} is given by

$${}^S \mathbf{p}(t) = \mathbf{R}_S^T(t)(\mathbf{x}_B(t) - \mathbf{x}_S(t) + \mathbf{R}_B(t)\mathbf{u}).$$

The time derivative of this equation is given below (velocities are again treated as constant), and uses the identity $\dot{\boldsymbol{\omega}}^T = -\dot{\boldsymbol{\omega}}$:

$$\begin{aligned} {}^S \dot{\mathbf{p}}(t) &= \mathbf{R}_S^T(t)(\dot{\mathbf{x}}_B - \dot{\mathbf{x}}_S + \dot{\boldsymbol{\omega}}_B \mathbf{R}_B(t)\mathbf{u}) \\ &\quad - \mathbf{R}_S^T(t)\dot{\boldsymbol{\omega}}_S(\mathbf{x}_B(t) - \mathbf{x}_S(t) + \mathbf{R}_B(t)\mathbf{u}) \end{aligned}$$

¹This assertion is untrue for the underlying continuous system but is true when numerically integrating the rigid body equations of motion.

Algorithm 2

DETERMINE-TOC($\mathcal{S}, t_0, t_f, \mathbf{v}_B, \boldsymbol{\omega}_B, \mathbf{p}(t_0), \mathbf{p}(t_f), \dot{\mathbf{p}}(t)$): given a stationary solid \mathcal{S} , a time interval $[t_0, t_f]$, and a point $\mathbf{p}(t)$ on a rigid body \mathcal{B} with linear and angular velocities \mathbf{v}_B and $\boldsymbol{\omega}_B$ at time t_0 , determines first time-of-impact t_χ of $\mathbf{p}(t)$ with \mathcal{S} , along with $\mathbf{p}(t_\chi)$ and the normal of \mathcal{S} at $\mathbf{p}(t_\chi)$.

```

 $t_\chi \leftarrow \infty$ 
push( $Q, \{t_0, t_f, \mathbf{p}(t_0), \mathbf{p}(t_f)\}$ )
while  $Q$  not empty do
   $\{t_a, t_b, \mathbf{p}(t_a), \mathbf{p}(t_b)\} \leftarrow \text{pop}(Q)$ 
   $\dot{\mathbf{p}}(t_a) \leftarrow \mathbf{x}_B + \dot{\boldsymbol{\omega}}_B \mathbf{R}_B \mathbf{u}$ 
   $n_\alpha \leftarrow \frac{1}{\|\mathbf{p}(t_b) - \mathbf{p}(t_a)\|} (\mathbf{p}(t_b) - \mathbf{p}(t_a))$ 
  form orthonormal basis  $\{n_\alpha, n_\beta, n_\gamma\}$ 
   $\nu_a^b = \|\dot{\mathbf{p}}(t_a)\| (t_b - t_a) - \|\mathbf{p}(t_b) - \mathbf{p}(t_a)\|$ 
   $c \leftarrow \frac{1}{2}(\mathbf{p}(t_a) + \mathbf{p}(t_b))$ 
   $d_\alpha \leftarrow \frac{1}{2}(\|\mathbf{p}(t_b) - \mathbf{p}(t_a)\| + \nu_a^b)$ 
   $d_\beta \leftarrow d_\gamma \leftarrow \frac{1}{2}\nu_a^b$ 
   $\mathcal{O} \leftarrow \text{OBB}(c, \{n_\alpha, d_\alpha\}, \{n_\beta, d_\beta\}, \{n_\gamma, d_\gamma\})$ 
  if  $\mathcal{O}$  intersects  $\mathcal{S}$  then
    if  $\nu_a^b > \epsilon$  then
       $t_i \leftarrow \frac{1}{2}(t_a + t_b)$ 
       $\mathbf{R}_B(t_i) \leftarrow \mathbf{R}_B(t_a) + \text{integrate } \dot{\mathbf{R}}_B(t) \text{ over } [t_a, t_i]$ 
       $\mathbf{p}(t_i) \leftarrow \mathbf{p}(t_a) + \text{integrate } \dot{\mathbf{p}}(t) \text{ over } [t_a, t_i]$ 
      push( $Q, \{t_a, t_i, \mathbf{p}(t_a), \mathbf{p}(t_i)\}$ )
      push( $Q, \{t_i, t_b, \mathbf{p}(t_i), \mathbf{p}(t_b)\}$ )
    else
      for all intersections  $p_x$  between  $\mathcal{O}$  and  $\mathcal{S}$  do
         $t_\alpha \leftarrow t_a + \frac{\|p_x - p_b\|}{\|p_a - p_b\|}$ 
        if  $t_\alpha < t_\chi$  then
           $t_\chi \leftarrow t_\alpha$ 
           $\mathbf{q} \leftarrow \mathbf{p}(t_\chi)$ 
           $\hat{\boldsymbol{\eta}} \leftarrow \hat{\mathbf{n}}(\mathcal{S}^q)$  {Get normal to  $\mathcal{S}$  at  $\mathbf{p}(t_\chi)$ }
      return  $\{t_\chi, \mathbf{q}, \hat{\boldsymbol{\eta}}\}$ 

```

Simplifying, we get:

$$\begin{aligned} {}^S \dot{\mathbf{p}}(t) &= \mathbf{R}_S^T(t)(\dot{\mathbf{x}}_B - \dot{\mathbf{x}}_S - \dot{\boldsymbol{\omega}}_S(\mathbf{x}_B(t) - \mathbf{x}_S(t)) \\ &\quad + (\dot{\boldsymbol{\omega}}_B - \dot{\boldsymbol{\omega}}_S)\mathbf{R}_B(t)\mathbf{u}) \end{aligned}$$

Computing the position of the point with respect to the frame of \mathcal{S} thus requires tracking $\mathbf{R}_S(t)$, $\mathbf{x}_B(t)$, and $\mathbf{x}_S(t)$ in addition to ${}^S \mathbf{p}(t)$ and $\mathbf{R}_B(t)$. On return, Algorithm 2 should be modified to transform \mathbf{q} and $\hat{\boldsymbol{\eta}}$ into the global frame.

3.3 Performance & computational complexity

The number of iterations necessary to reduce the curved trajectory into line-segments depends on the trajectory (essentially a function of $\dot{\mathbf{x}}_B, \dot{\mathbf{x}}_S, \dot{\mathbf{R}}_B, \dot{\mathbf{R}}_S$) and the precision parameter ϵ . In the limit as $\epsilon \rightarrow 0$ the algorithm resolves the query exactly. With a finite ϵ the performance of the algorithm needs further description. It is *not* the case that the method finds the earliest time at which the solid passes within ϵ of the trajectory, but instead the method finds the time at which intersects the volume of a piecewise linear curve of boxes with ϵ sides, and is guaranteed to contain the original trajectory.

The computational complexity of our method will be analyzed for the case of polyhedra (convex or non-convex) P and Q with N and M features, respectively; we assume that a zero level-set is also constructed offline, as in Guendelman et al. [12], for each polyhedron to facilitate constant time signed distance queries.

Suppose Algorithm 2 requires k iterations. Within the worst case intersections with the solid and bounding boxes require the entire

trajectory be reduced to bounding boxes with $\nu_a^b \leq \epsilon$, thus k depends on the curvature of the trajectory, which can be bounded by a linear function of $\|\dot{\mathbf{x}}_B\|$, $\|\dot{\mathbf{x}}_S\|$, $\|\dot{\mathbf{R}}_B\|$, $\|\dot{\mathbf{R}}_S\|$. The nature of physical simulation bounds these quantities, which do not depend on the size of the polyhedra.

Since $\Theta(R)$ time is required to determine the intersection between an oriented bounding box and a polyhedron with R features [11], and algorithm 2 is called N times against the implicit surface corresponding to Q and M times against the implicit surface corresponding to P . Therefore, determining the times of intersection of vertices of each polyhedron inside the other exhibits time complexity $O(kNM + kMN)$.

3.4 Variation sans bounding-boxes

If intersection queries between the solid and an oriented bounding box cannot be computed easily, then an alternative method using a piecewise linear approximation to $\mathbf{p}(t)$ will succeed. This requires that line segments between points $\mathbf{p}(t_a)$ and $\mathbf{p}(t_b)$ be bisected recursively until $\nu_i^j < \epsilon$ for any consecutive i, j . Each line segment is then tested for intersection with the surface. This alternate method works equally correctly to the algorithm above, but would generally require many more intersection queries. This is detailed in Algorithm 3.

Algorithm 3

DETERMINE-TOC-ALT($\mathcal{S}, t_0, t_f, \mathbf{v}_B, \boldsymbol{\omega}_B, \mathbf{p}(t_0), \mathbf{p}(t_f), \dot{\mathbf{p}}(t)$): given a stationary solid \mathcal{S} , a time interval $[t_0, t_f]$, and a point $\mathbf{p}(t)$ on a rigid body \mathcal{B} with linear and angular velocities \mathbf{v}_B and $\boldsymbol{\omega}_B$ at time t_0 , determines first time-of-impact t_χ of $\mathbf{p}(t)$ with \mathcal{S} , along with $\mathbf{p}(t_\chi)$ and the normal of \mathcal{S} at $\mathbf{p}(t_\chi)$.

```

 $t_\chi \leftarrow \infty$ 
push( $Q, \{t_0, t_f, \mathbf{p}(t_0), \mathbf{p}(t_f)\}$ )
while  $Q$  not empty do
   $\{t_a, t_b, \mathbf{p}(t_a), \mathbf{p}(t_b)\} \leftarrow \text{pop}(Q)$ 
   $\dot{\mathbf{p}}(t) \leftarrow \mathbf{x}_B + \tilde{\boldsymbol{\omega}}_B \mathbf{R}_B \mathbf{u}$ 
   $\nu_a^b = \|\dot{\mathbf{p}}(t_a)\| (t_b - t_a) - \|\mathbf{p}(t_b) - \mathbf{p}(t_a)\|$ 
  if  $\nu_a^b > \epsilon$  then
     $t_i \leftarrow \frac{1}{2}(t_a + t_b)$ 
     $\mathbf{R}_B(t_i) \leftarrow \mathbf{R}_B(t_a) + \text{integrate } \dot{\mathbf{R}}_B(t) \text{ over } [t_a, t_i]$ 
     $\mathbf{p}(t_i) \leftarrow \mathbf{p}(t_a) + \text{integrate } \dot{\mathbf{p}}(t) \text{ over } [t_a, t_i]$ 
    push( $Q, \{t_a, t_i, \mathbf{p}(t_a), \mathbf{p}(t_i)\}$ )
    push( $Q, \{t_i, t_b, \mathbf{p}(t_i), \mathbf{p}(t_b)\}$ )
  else
     $p_x \leftarrow \text{intersection between } \overline{\mathbf{p}(t_a)\mathbf{p}(t_b)} \text{ and } \mathcal{S} \text{ closest to } \mathbf{p}(t_a)$ 
     $t_\alpha \leftarrow t_a + \frac{\|\mathbf{p}_x - \mathbf{p}_b\|}{\|\mathbf{p}_a - \mathbf{p}_b\|}$ 
    if  $t_\alpha < t_\chi$  then
       $t_\chi \leftarrow t_\alpha$ 
       $\mathbf{q} \leftarrow \mathbf{p}(t_\chi)$ 
       $\hat{\boldsymbol{\eta}} \leftarrow \hat{\mathbf{n}}(\mathcal{S}^q)$  {Get normal to  $\mathcal{S}$  at  $\mathbf{p}(t_\chi)$ }
return  $\{t_\chi, \mathbf{q}, \hat{\boldsymbol{\eta}}\}$ 

```

This variation has the advantage that the first TOC t_α between a line segment and an implicit surface can be found rapidly using methods for ray tracing level sets; for example, the method of Singh and Narayanan [20] achieves super-interactive frame rates using GPUs. Thus allowing constant time checks, and allowing the algorithm complexity $O(kN + kM)$, i.e., linear in the polyhedral features.

However, it must be noted that the role of the ϵ has a different interpretation despite the fact that in the limit of small ϵ the two algorithms are equivalent. A solid intersecting the original curve

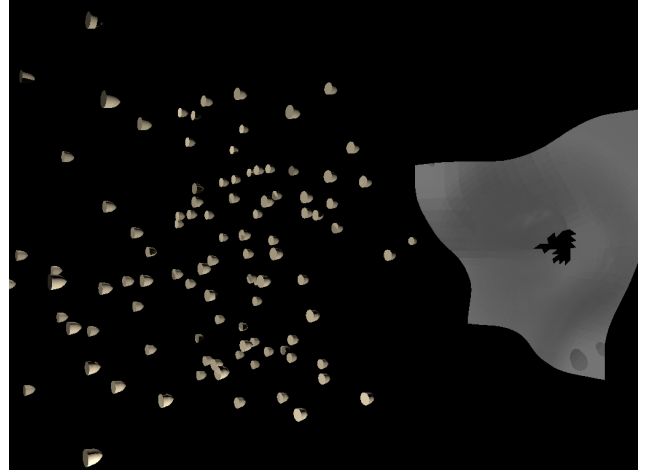


Figure 3: 100 bullets (each a 640 triangle polyhedron) moving at 1000 m/s about to impact a stationary warped plane (a 2010 triangle polygon soup) with a hole in the center.

with depth less than ϵ can be missed by this variation whereas it would be detected by Algorithm 2. (Generally the detection algorithm that operates as part of the integrator, which is responsible for detecting penetration at time t_f , has a numerical accuracy that can be used to select a suitable ϵ .)

3.5 Broad phase collision detection

We speed the collision detection process considerably by quickly eliminating pairs of bodies from consideration that have no possibility of intersecting (i.e., *broad phase collision detection*; this speedup uses oriented bounding boxes built around the individual geometries. During a collision check between a pair of bodies, a sphere of radius $v_i(t)$ is determined around the eight vertices $v_i(t)$ of each bounding box; each bounding box is then enlarged such that the eight spheres are contained. Only if these enlarged OBBs intersect are the algorithms presented previously (i.e., *narrow phase collision detection*) applied. Even tighter bounds on object motion could be readily computed, at the possible cost of greater bounding volume/bounding volume intersection tests.

4. EXAMPLE

We illustrate our method using a pathological scenario that is problematic for discrete collision detection methods: a small object with high velocity coming into contact with a very thin object. Using a discrete collision detection method on this example requires the use of extremely small step sizes, thus precluding interactive simulation.

The small object with high velocity is a bullet in our example, and the thin object is a warped plane with a hole in the center. To illustrate that our method has no difficulty with the hole in the warped plane, we use 100 bullets moving at 1000 m/s; each bullet is subject to gravity (the plane is immobile) and additionally incorporates an angular velocity around its longitudinal axis of $2e^{-3}$ rad/sec, consistent with a rifling effect. The example is rendered in Figure 3.

The bullets are represented by convex polyhedra (640 triangles) while the warped plane is represented using a polygon soup (2010 triangles). We did not use implicit surface representations; line segments were tested directly against triangles to determine con-

tact points and times of contact. Oriented bounding boxes were employed to reduce the number of triangles necessary for testing. This approach exhibits slower asymptotic complexity than that discussed in Section 3.3 (i.e., as poor as $\Theta(MN)$, but more inline with $\Theta(M \lg N + N \lg M)$ in our experience), but does permit the use of the polygon soup.

The simulation was able to run effectively at high step sizes; a step size of 0.1 was used with an explicit Euler integrator on this example. An impulse-based contact algorithm resolved impacts with the warped plane. Approximately 201 seconds on a 3.0 GHz Core Duo processor was required to compute the one second of simulation time in which the 100 bullets impacted the plane.

This particular example represents a worst-case scenario for our method. The rigid bodies move with extremely high velocity, so the narrow phase is triggered with high frequency; tightening the bounding volumes used in the broad phase would have no effect as the motion of the bullets is mostly linear. Additionally, the geometries are composed of a large number of triangles, and therefore vertices, so a great number of intersection tests are performed.

5. DISCUSSION

Standard collision detection algorithms have proven to be less than perfectly suited for rigid body simulation. The convexity-based methods, which permit ready determination of the contact data, exhibit quadratic complexity for non-convex polyhedra (due to the necessary decomposition) and are unusable on polygon soups, general implicit surfaces, etc. In contrast, the *a posteriori* methods, which can handle more varied geometric representations, frequently determine inaccurate contact points and normals. Our algorithm is generally sufficiently fast for rigid body simulation, even if its worst case running times may be prohibitive for some interactive applications.

6. REFERENCES

- [1] C. L. Bajaj and T. K. Dey. Convex decomposition of polyhedra and robustness. *SIAM J. Comput.*, 21(2):339–364, 1992.
- [2] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics*, 23(3), July 1989.
- [3] D. Baraff. An introduction to physically based modeling: Rigid body simulation II— nonpenetration constraints. Technical report, Robotics Institute, Carnegie Mellon University, 1997.
- [4] S. Cameron. Enhancing GJK: Computing minimum and penetration distances between convex polyhedra. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Albuquerque, NM, USA, April 1997.
- [5] J. Canny. Collision detection for moving polyhedra. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 8(2), 1986.
- [6] N. Chakraborty, S. Berard, S. Akella, and J. Trinkle. An implicit time-stepping method for multibody systems with intermittent contact. In *Proc. of Robotics: Science and Systems*, 2007.
- [7] B. Chazelle. Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm. *SIAM J. Comput.*, 13:488–507, 1984.
- [8] K. Chung and W. Wang. Quick collision detection of polytopes in virtual environments. In M. Green, editor, *Proc. of the ACM Symp. on Virtual Reality Software and Technology (VRST)*, pages 125–131, 1996.
- [9] C. Ericson. *Real-Time Collision Detection*. Morgan Kaufmann, 2005.
- [10] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE J. of Robotics and Automation*, 4(2):193–203, April 1988.
- [11] N. Greene. Detecting intersection of a rectangular solid and a convex polyhedron. In *Graphics Gems IV*, pages 74–82. Academic Press, 1994.
- [12] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex rigid bodies with stacking. *ACM Trans. on Graphics*, 22(3):871–878, 2003.
- [13] S. Hasegawa and N. Fujii. Real-time rigid body simulation based on volumetric penalty method. In *Proc. of the 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS)*, 2003.
- [14] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha. Fast penetration depth computation for physically-based animation. In *Proc. of Symposium on Computer Animation (SCA)*, 2002.
- [15] M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1008–1014, 1991.
- [16] B. Mirtich. V-Clip: fast and robust polyhedral collision detection. *ACM Trans. on Graphics*, 17(3):177–208, 1998.
- [17] M. Moore and J. Wilhelms. Collision detection and response for computer animation. In *Proc. of Intl. Conf. on Computer Graphics and Interactive Techniques*, pages 289–298, 1988.
- [18] S. Redon, A. Kheddar, and S. Coquillart. Fast continuous collision detection between rigid bodies. In *Proc. of Eurographics (Computer Graphics Forum)*, 2002.
- [19] S. Redon and M. Lin. A fast method for local penetration depth computation. *J. of Graphics Tools*, 11(2):37–50, 2006.
- [20] J. M. Singh and P. Narayanan. Real-time ray tracing of implicit surfaces on the gpu. Technical Report IIIT/TR/2007/72, Intl. Inst. of Information Technology, July 2007.
- [21] G. van den Bergen. A fast and robust GJK implementation for collision detection of convex objects. *J. of Graphics Tools*, 4(2):7–25, 1999.
- [22] K. Vlack and S. Tachi. Fast and accurate spacio-temporal intersection detection with the GJK algorithm. In *Proc. of the Intl. Conf. on Artificial Reality and Telexistence (ICAT)*, pages 79–84, 2001.
- [23] K. Yamane and Y. Nakamura. Stable penalty-based model of frictional contacts. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Orlando, FL, USA, May 2006.
- [24] X. Zhang, M. Lee, and Y. J. Kim. Interactive continuous collision detection for non-convex polyhedra. In *The Visual Computer (Proc. of Pacific Graphics)*, 2006.