

Robust Continuous Collision Detection Between Arbitrary Polyhedra Using Trajectory Parameterization of Polyhedral Features

March 10th 2005

J.M.P. van Waveren

© 2005, id Software, Inc.

Abstract

A continuous collision detection algorithm is presented which detects collisions between arbitrary polyhedra in motion by evaluating trajectory parameterizations of polyhedral features. The presented algorithm strictly prevents interpenetration and does not report any collisions until polyhedra are in contact within a small floating-point epsilon. While using trajectory parameterization for collision detection between polyhedra is not new, a novel approach is presented for dealing with numerical error due to rounding in floating-point collision detection calculations. Furthermore, several optimizations are presented that significantly reduce the computational cost. The presented algorithm has been shown to be fast, robust and accurate and has been successfully implemented and employed in the computer game DOOM III.

1. Introduction

The problem of collision detection is encountered in many different fields like robotics, computer graphics, virtual reality environments and computer games. Different applications may have different requirements when it comes to the various aspects of a collision detection system, like accuracy and computational cost. Ideally a collision detection system is fast, accurate, robust and does not restrict the shape and geometric complexity of collision primitives.

The computational cost of collision detection is always an important issue because in today's simulations many objects with a variety of geometric detail may be in motion at the same time requiring the detection of many collisions. Therefore many collision detection algorithms gain speed at the cost of losing precision. Many approaches allow objects to interpenetrate or collisions are detected while objects are still significantly separated. Many collision detection algorithms also restrict the shapes of the objects that can be used, or complex objects are approximated with simple shapes like boxes, spheres, ellipsoids and cylinders. However, with the ability to visualize simulations with high detail real-time lighting and shadowing on today's computers, the accuracy of collision detection algorithms has become a more important issue in a variety of interactive applications.

The collision detection algorithm described in this paper has been developed for the computer game DOOM III. In this computer game the real-time physically simulated objects are displayed with high detail real-time lighting and shadowing. Therefore the collision detection does not only have to be robust but also particularly accurate. Premature detection of collisions may keep objects visibly separated causing unrealistic lighting effects such as light passing through the spaces in between stacked objects. On the other hand, late detection of collisions may cause objects to interpenetrate or overlap which causes rendering artifacts such as Z-fighting where rendered pixels on screen rapidly change color as the view point changes gradually. Not only accuracy, but also performance is particularly important in DOOM III because a lot of computer resources are already dedicated to rendering at high frame rates.

1.1 Previous work

Many algorithms for 3D collision detection have been proposed in literature. These algorithms can generally be grouped into four different approaches based on solving strategy [2]: space-time intersection tests, swept volume interference detection, multiple interference detection and trajectory parameterization.

Approaches based on space-time intersection tests solve the collision detection problem in its most general form. These approaches use 4D extruded volumes of 3D objects in motion to calculate collisions [4, 7]. Such an extruded volume is the spatio-temporal set of points that represents the spatial occupancy of the object as it moves along its trajectory. For more general motions than translations the extruded volumes are bounded by complex hypersurfaces which makes it difficult to implement algorithms to calculate intersections between such volumes.

Approaches based on swept volume interference detection use volumes that contain all the points occupied by a moving object during a time period [8, 9]. Such a swept volume is in essence a projection of a 4D extruded volume onto a lower dimensional subspace. If the swept volumes of two moving objects do not intersect there is no collision. However, when the swept volumes do intersect there is not necessarily a collision. Only for a single moving object colliding with one or more stationary objects, a swept volume intersection is both a sufficient and necessary condition for a collision to take place.

Approaches based on multiple interference detection are most widely used to find collisions [10, 12, 14, 18, 19, 30, 32, 33]. Objects are considered to be in a colliding state when they interpenetrate. The object trajectories are sampled in time and static interference tests are applied repeatedly. A too coarse sampling may cause collisions to be missed and a too fine sampling can be computationally expensive. Stepping back in time and recalculating interference may be required to find more accurate points of collision. In most cases, intersection tests between simple geometric entities are used for the interference detection. Some algorithms, like the ones based on GJK [10] require the collision primitives to be convex. Other algorithms only use simple mathematical shapes like boxes, spheres, and cylinders.

Approaches based on trajectory parameterization determine collisions analytically by expressing the object trajectories as functions of the parameter time [39, 40]. The degrees of the polynomials used to describe complex motions can be arbitrarily high. Polynomials of order 5 and above cannot be solved analytically and the computation of collisions can be very expensive. Therefore the motion of an object between two positions along its trajectory is often replaced by an arbitrary motion which can be described with a low degree polynomial.

The algorithm presented in this paper is based on trajectory parameterization. Similar formulations for the basic collision detection calculations were first introduced in [39] and later in [43]. This paper, however, presents a novel approach for dealing with numerical error in floating-point collision detection calculations, and describes several optimizations that significantly reduce the computational cost.

1.2 Layout

Section 2 describes the basic collision detection calculations. In section 3 a method is presented for dealing with numerical error in collision detection calculations on computers. Section 4 suggest several optimizations for the basic collision detection calculations. A broad phase for minimizing the exact collision detection calculations by first localizing possible collision regions is described in section 5. The results of implementing and using the described collision detection system in DOOM III are presented in section 6 and several conclusions are drawn in section 7.

2. Collision detection between arbitrary polyhedra

During a simulation multiple polyhedra can usually move simultaneously. However, for the described algorithm the polyhedra are assumed to be moving separately, one by one, for short periods of time. Therefore the collision detection problem is reduced to a single moving polyhedron colliding with one or more stationary polyhedra. Moving polyhedra one by one may cause certain collisions to be missed when the polyhedra move over relatively large distances, but in many applications this is not a problem. Besides, the collision detection problem of two polyhedra in motion can always be reduced to one polyhedron in motion and one stationary polyhedron by transforming the motion of the one polyhedron into the motion space of the other. In other words, the polyhedron in motion is moving relative to the motion of the other polyhedron.

The motion between two positions along the path of a polyhedron can be a simultaneous rotation and translation. However, in many simulations the exact motion is unknown or ill-defined. If a parameterized formulation of this motion does exist, it often involves polynomial equations of degree 3 or higher. Solving such polynomial equations on a computer is not only costly but also introduces considerable numerical error in the results, even if an explicit formulation of the roots exists. For this reason any motion between two positions along the path of a polyhedron is replaced with a separate translation and a separate rotation. For a translation and a rotation about an arbitrary axis, the polynomial equations to be solved have at most degree two. The exact motion between two positions along the path of a polyhedron is usually not described by this separate translation and rotation. However, when no collisions occur, the initial and final position of a polyhedron are still the same, and if the movement between the two positions is small there is usually little difference even if there are collisions. Most simulations are also visualized by rendering distinct frames where the visual representation of a polyhedron is only updated at certain positions along its path. Therefore using a more accurate parameterization of the motion of a polyhedron may be important for the accuracy of the simulation of discontinuities, but it is typically not particularly important for making a simulation look visually plausible.

The continuous surfaces of a polyhedron consist of planar faces (polygons) with straight boundaries (edges). Each edge is defined by two points (vertices) that mark the start and end of an edge. Although the polyhedra can have any shape, the presented algorithm requires polygons to be convex. To find collisions between polyhedra, only the motion of a vertex and an edge need to be parameterized because only these features of one polyhedron can collide with the polygons or edges of another polyhedron. Collisions are classified as either vertex-polygon or edge-edge collisions. The degenerate cases, vertex-vertex and vertex-edge, are not considered and are always classified as vertex-polygon collisions.

2.1 Plücker coordinates

The presented algorithm is described in terms of Plücker coordinates [48, 49, 50, 51]. More general Grassman coordinates or Geometric Algebra could be used instead but, in the context of the described algorithm, Plücker coordinates provide a concise representation without the need for a more involved algebra.

To find a parameterization of the motion of an edge, and to be able to easily verify boundaries for collisions, each edge of a polygon is described with a Plücker coordinate. Additionally, a vertex in combination with its direction of motion is represented by a Plücker coordinate.

Any ordered pair of distinct points in three-space, $a = (a_x, a_y, a_z)$ and $b = (b_x, b_y, b_z)$ defines a directed line in three dimensions. This line corresponds to a projective six-tuple $p = (p_0, p_1, p_2, p_3, p_4, p_5)$ of which each component is the determinant of a 2x2 minor of the 2x4 matrix which has the homogenous coordinates of the points a and b as rows:

$$\begin{matrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \end{matrix}$$

This six-tuple, which relates to a point in Plücker space, is defined by:

$$\begin{aligned} p_0 &= a_x b_y - b_x a_y \\ p_1 &= a_x b_z - b_x a_z \\ p_2 &= a_x - b_x \\ p_3 &= a_y b_z - b_y a_z \\ p_4 &= a_z - b_z \\ p_5 &= b_y - a_y \end{aligned}$$

The six coordinates are not independent and they must satisfy the following equation whose solution constitutes the Plücker hypersurface or Grassman manifold.

$$0 = p_0 p_4 + p_1 p_5 + p_2 p_3$$

If $p = (p_0, p_1, p_2, p_3, p_4, p_5)$ and $q = (q_0, q_1, q_2, q_3, q_4, q_5)$ are the Plücker coordinates for two directed lines a - b and c - d respectively, then the following permuted inner product describes a sidedness relation between these lines.

$$s = p_0 q_4 + p_1 q_5 + p_2 q_3 + p_4 q_0 + p_5 q_1 + p_3 q_2$$

This sidedness relation is equivalent to the determinant of the following 4x4 matrix:

$$\begin{matrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ d_x & d_y & d_z & 1 \end{matrix}$$

The absolute value of this determinant represents the volume of the tetrahedron whose vertices are: a , b , c and d . The sign of the determinant is positive if the lines a - b and c - d have the same orientation as the reference frame chosen in three-space. When the volume vanishes the 4 points are coplanar and therefore the two lines a - b and c - d intersect. In other words, the sidedness relation describes how the lines are oriented in space relative to each other as shown in Fig. 1.

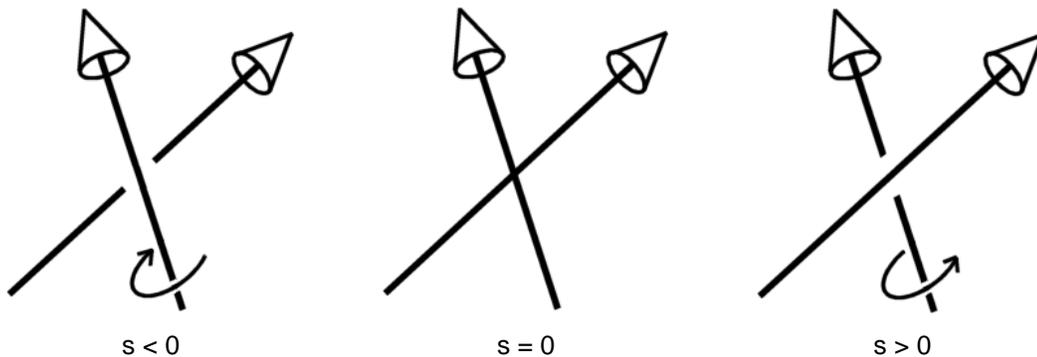


Fig. 1: sidedness relation between two lines in three dimensional space.

The position in combination with the direction of motion of a point in three-space can also be represented by a Plücker coordinate. The position $a = (a_x, a_y, a_z)$ and the direction of motion $v = (v_x, v_y, v_z)$ of a point correspond to the following Plücker coordinate:

$$\begin{aligned} p_0 &= a_x v_y - v_x a_y \\ p_1 &= a_x v_z - v_x a_z \\ p_2 &= -v_x \\ p_3 &= a_y v_z - v_y a_z \\ p_4 &= -v_z \\ p_5 &= v_y \end{aligned}$$

The permuted inner product of this Plücker coordinate with the Plücker coordinate of a directed line can be used to determine at which side the point passes the line.

The polygons of a polyhedron are planar faces. A polygon plane $p = (p_a, p_b, p_c, p_d)$ is represented by the plane equation:

$$p_a x + p_b y + p_c z + p_d = 0$$

The normal vector of this plane represented by (p_a, p_b, p_c) points towards the half-space at the front of the polygon plane, and towards the outside the polyhedron. Collisions are assumed to only take place with the front side of polygons. A polygon must be convex and the edges are stored counter clockwise when the polygon is viewed from the front. The Plücker coordinates for these edges also describe lines in a counter clockwise fashion.

2.2 Translational vertex-polygon collisions

During a translation a vertex of one polyhedron can collide with a polygon of another polyhedron. If $a = (a_x, a_y, a_z)$ is the position of a vertex before translation and $v = (v_x, v_y, v_z)$ is the translation vector and $p = (p_a, p_b, p_c, p_d)$ is the polygon plane, then α and β are defined as follows:

$$\begin{aligned} \alpha &= p_a a_x + p_b a_y + p_c a_z + p_d \\ \beta &= p_a v_x + p_b v_y + p_c v_z \end{aligned}$$

When $\beta = 0$ the translation is parallel to the plane and there will be no collision. The fraction f of the translation completed before colliding with the polygon plane is given by:

$$f = \alpha / \beta$$

Only if $f \in [0, 1]$ there is a collisions during the translation along v . Furthermore, the vertex only collides with the polygon if the plane is hit between the polygon edges. To verify if the latter is true the position of the vertex in combination with the translation vector is represented by a Plücker coordinate as described in section 2.1. The polygon edges are also represented by Plücker coordinates which describe lines in a counter clockwise fashion. If the permuted inner product of the Plücker coordinate for the vertex translation with each Plücker coordinate for a polygon edge, results in a positive scalar, then the vertex collides with the front of the polygon between the edges. Using the inner product between Plücker coordinates to verify if a vertex collides with a polygon plane between the polygon edges works better than for instance commonly used inside triangle tests. For two adjacent polygons that share an edge the sidedness relation is exactly the same and there is no degenerate case at the shared edge due to numerical error.

2.3 Translational edge-edge collisions

During a translation an edge of one polyhedron can collide with an edge of another polyhedron. The two supporting lines of these edges lie in a plane when the permuted inner product of their

Plücker coordinates is zero. The collision with a stationary edge can be calculated by substituting a translation vector into the Plücker coordinate of a translating edge. The Plücker coordinate of the stationary edge is $q = (q_0, q_1, q_2, q_3, q_4, q_5)$ and the translation is given by the vector $v = (v_x, v_y, v_z)$. If $p = (p_0, p_1, p_2, p_3, p_4, p_5)$ is the Plücker coordinate for the translating edge through $a = (a_x, a_y, a_z)$ and $b = (b_x, b_y, b_z)$ then the translation p^t is defined by:

$$\begin{aligned} p_0^t &= (a_x + v_x f)(b_y + v_y f) - (b_x + v_x f)(a_y + v_y f) \\ p_1^t &= (a_x + v_x f)(b_z + v_z f) - (b_x + v_x f)(a_z + v_z f) \\ p_2^t &= a_x - b_x \\ p_3^t &= (a_y + v_y f)(b_z + v_z f) - (b_y + v_y f)(a_z + v_z f) \\ p_4^t &= a_z - b_z \\ p_5^t &= b_y - a_y \end{aligned}$$

Here f is the fraction of the translation completed. Substituting this Plücker coordinate into the permuted inner product with the Plücker coordinate of the stationary edge results in:

$$\begin{aligned} \alpha &= p_0 q_4 + p_1 q_5 + p_2 q_3 + p_4 q_0 + p_5 q_1 + p_3 q_2 \\ \beta &= q_4 (-p_2 v_y - p_5 v_x) + q_5 (-p_2 v_z + p_4 v_x) + q_2 (p_5 v_z + p_4 v_y) \end{aligned}$$

$$f = \alpha / \beta$$

α is the permuted inner product of the Plücker coordinates for the lines. If $\beta = 0$ the lines are parallel and any collisions will be found as vertex-polygon collisions. Only if $f \in [0, 1]$ there is a collisions during the translation along the vector v . Edges are line segments and a collision between edges can only occur when the supporting lines of the edges collide, and the collision is between the bounds of the edges. To verify if two edges collide between the vertices that define their bounds these vertices in combination with the translation vector are represented by Plücker coordinates as described in section 2.1. If the permuted inner products between each of the two Plücker coordinates for the vertices of one edge and the Plücker coordinate of a second edge have a different sign, then the first edge collides with the supporting line of the second edge between its bounds. The same calculation can be used to verify if the supporting line of the first edge collides between the vertices of the second edge. Notice that these are the same sidedness calculations as used to verify if a vertex collides with a polygon plane between the polygon edges.

2.4 Rotational vertex-polygon collisions

During a rotation a vertex of one polyhedron can collide with a polygon of another polyhedron. Instead of rotating a vertex about an arbitrary axis, a vertex is rotated about the z-axis. A situation with a vertex rotating about an arbitrary axis can always easily be transformed such that the rotation axis becomes the z-axis. The polygon plane is $p = (p_a, p_b, p_c, p_d)$ and the angle of rotation about the z-axis is β . If $a = (a_x, a_y, a_z)$ is the position of a vertex before rotation then the rotation of this point a^r is given by:

$$\begin{aligned} a_x^r &= a_x \cos \alpha + a_y \sin \alpha \\ a_y^r &= a_y \cos \alpha - a_x \sin \alpha \\ a_z &= a_z \end{aligned}$$

Here α is the angle of rotation. Substituting this point in the plane equation results in:

$$\begin{aligned} v_0 &= p_c a_z + p_d \\ v_1 &= p_a a_y - p_b a_x \\ v_2 &= p_a a_x + p_b a_y \\ v_2 \cos \alpha + v_1 \sin \alpha + v_0 &= 0 \end{aligned}$$

By substituting the following parametric formulation:

$$\sin \alpha = 2 r / (1 + r^2), \quad \cos \alpha = (1 - r^2) / (1 + r^2), \quad r = \tan(\alpha / 2) \text{ for } 0 < \alpha < \pi$$

the equation becomes a quadratic equation:

$$a r^2 + 2 b r + c = 0$$

$$a = v_0 - v_2, \quad b = v_1, \quad c = v_0 + v_2$$

with roots:

$$r_1 = (-b + \sqrt{(b^2 - a c)}) / a$$

$$r_2 = (-b - \sqrt{(b^2 - a c)}) / a$$

The two fractions of rotation completed before a collision occurs are given by:

$$f_1 = 2 \arctan(r_1) / \beta$$

$$f_2 = 2 \arctan(r_2) / \beta$$

As expected there are in general two angles at which the vertex collides with the polygon plane. If $a = 0$ and $b \neq 0$ there is one collision at time $-c / b$ and another at ∞ , which corresponds to a rotation angle $\alpha = \pi$. The degenerate case $a = b = c = 0$ occurs if the rotation axis is orthogonal to the plane. The smallest fraction in the range $[0-1]$ gives the first collision with the polygon plane. To verify if a collision occurs between the polygon edges a similar procedure is used as for the translational case. The vertex is first rotated up to the point of collision. At that point the position of the vertex in combination with the direction of motion of the vertex is used to verify if the collision occurs between the polygon edges.

2.5 Rotational edge-edge collisions

During a rotation an edge of one polyhedron can collide with an edge of another polyhedron. Just like with the vertex-polygon collision calculations a stationary edge and a rotating edge are transformed such that the rotation axis becomes the z-axis. The collision with a stationary edge can be calculated by substituting a rotation into the Plücker coordinate of a rotating edge. The Plücker coordinate of the stationary edge is $q = (q_0, q_1, q_2, q_3, q_4, q_5)$ and the angle of rotation about the z-axis is β . If $p = (p_0, p_1, p_2, p_3, p_4, p_5)$ is the Plücker coordinate for the rotating edge through $a = (a_x, a_y, a_z)$ and $b = (b_x, b_y, b_z)$, then the rotation p^r is defined by:

$$p_0^r = (a_x \cos \alpha + a_y \sin \alpha) (b_y \cos \alpha - b_x \sin \alpha) - (b_x \cos \alpha + b_y \sin \alpha) (a_y \cos \alpha - a_x \sin \alpha)$$

$$p_1^r = (a_x \cos \alpha + a_y \sin \alpha) b_z - (b_x \cos \alpha + b_y \sin \alpha) a_z$$

$$p_2^r = (a_x \cos \alpha + a_y \sin \alpha) - (b_x \cos \alpha + b_y \sin \alpha)$$

$$p_3^r = (a_y \cos \alpha - a_x \sin \alpha) b_z - (b_y \cos \alpha - b_x \sin \alpha) a_z$$

$$p_4^r = a_z - b_z$$

$$p_5^r = (b_y \cos \alpha - b_x \sin \alpha) - (a_y \cos \alpha - a_x \sin \alpha)$$

Here α is the angle of rotation. Substituting this Plücker coordinate into the permuted inner product with the Plücker coordinate of the stationary edge results in:

$$v_0 = q_0 p_4 + q_4 p_0$$

$$v_1 = q_1 p_2 - q_2 p_1 + q_5 p_3 - q_3 p_5$$

$$v_2 = q_1 p_5 + q_2 p_3 + q_5 p_1 + q_3 p_2$$

$$v_2 \cos \alpha + v_1 \sin \alpha + v_0 = 0$$

By substituting the following parametric formulation:

$$\sin \alpha = 2 r / (1 + r^2), \quad \cos \alpha = (1 - r^2) / (1 + r^2), \quad r = \tan(\alpha / 2) \text{ for } 0 < \alpha < \pi$$

the equation becomes a quadratic equation:

$$a r^2 + 2 b r + c = 0$$
$$a = v_0 - v_2, \quad b = v_1, \quad c = v_0 + v_2$$

with roots:

$$r_1 = (-b + \sqrt{b^2 - a c}) / a$$
$$r_2 = (-b - \sqrt{b^2 - a c}) / a$$

The two fractions of rotation completed before a collision occurs are given by:

$$f_1 = 2 \arctan(r_1) / \beta$$
$$f_2 = 2 \arctan(r_2) / \beta$$

There are in general two angles for which the supporting lines of the edges intersect. If $a = 0$ and $b \neq 0$ there is one collision at time $-c / b$ and another at ∞ , which corresponds to a rotation angle $\alpha = \pi$. The degenerate case $a = b = c = 0$ occurs if both supporting lines intersect the rotation axis in the same point or if both are parallel to it or if both lie in the same plane perpendicular to the axis of rotation. In these cases there will be either no collision or the collision will be found as a vertex-polygon collision. The smallest fraction in the range $[0-1]$ gives the first collision between the supporting lines of the edges. Edges are line segments and a collision between edges can only occur when the supporting lines of the edges collide, and the collision is between the bounds of the edges. To verify if a collision occurs between the edge bounds a similar procedure is used as for the translational case. However, the rotating line is first rotated up to the point of collision. At that point the direction of motion at the collision point is used in combination with the edge vertices to verify if the collision occurs between the edge bounds.

2.6 Retrieving contact points

If two triangles are considered for collision detection there will be 6 vertex-polygon and 9 edge-edge translational and rotational collision detection calculations. Using the translational and rotational collision detection described above the first collision can be found by sorting the fractions or angles for which collisions occur between edges, vertices and polygons. For physics simulations it is often important to retrieve all the contact points between two objects in a colliding state. Retrieving these contact points is relatively easy. Instead of searching for the first collision, all the collisions between edges, vertices and polygons within a small distance from the moving object are listed. This small distance is the numerical tolerance within which objects are considered to be touching each other. The contacts can be gathered in an omni-directional fashion by translating the features of a polyhedron outwards along the polygon plane normals and edge normals. The contact point for vertex-polygon collisions is the position of the vertex at the moment of impact. For edge-edge collisions the contact point is the intersection of the supporting lines at the moment of impact. This intersection point can easily be calculated by constructing a plane through one of the lines and calculating the intersection of this plane with the other line.

3. Expel zone

An issue often neglected in exact collision detection algorithms is dealing with rounding in calculations on computers. Scalars are typically stored as floating-point numbers which are represented by a limited number of bits, and as such are limited in precision and subject to rounding. As more calculations are performed with these numbers the rounding errors accumulate. The accumulated error is often relatively small, but with an algebraic approach to collision detection as described here, the accumulated error can pose an interesting problem. Once a collision has been detected a polyhedron is usually moved up to the point of collision. At this point the colliding polyhedra should be touching each other exactly. However, this might not be true because of rounding in the collision detection calculations. The polyhedra might be slightly separated or they may interpenetrate. Once the polyhedra intersect, no collisions will be found during a next query if the polyhedra still move towards each other because collisions are only calculated between the boundary representations of the polyhedra. When polyhedra are in a colliding state constraints can be introduced which allow the polyhedra to only move away from each other. However, the results are often poor because of algorithmic complexity.

Instead, an expel zone is introduced which tightly fits around a polyhedron. The thickness of this expel zone is just larger than the maximum accumulated error which can be built up during collision detection calculations. A polyhedron will not collide with the exact representation of another polyhedron, but with the outer hull of the expel zone. Due to rounding a polyhedron can slightly poke into the expel zone of another polyhedron. However, the polyhedron will never intersect with the other polyhedron because the expel zone is larger than the maximum accumulated error in the collision detection calculations. Any polyhedral fragment which ends up in the expel zone of another polyhedron, may only move through the expel zone in a direction which leads the fragment away from the other polyhedron. Fortunately such an expel zone can be implemented with just a small amount of logic and few calculations. In general the expel zone does not slow down collision detection queries.

The expel zone is assumed to have a thickness e . The error built up during collision detection calculations is assumed to be smaller than e at all times. The checks to verify if a collision occurs between polygon edges or between edge bounds are not affected by the expel zone and stay the same.

3.1 Translational vertex-polygon

When α and β are defined as in section 2.2, the expel zone for the translational vertex-polygon collision detection is implemented in pseudo code as follows.

```
IF  $\beta \geq 0$  THEN
   $f = 1$ 
ELSE
   $f = \min(\max((\alpha - e) / \beta, 0), 1)$ 
ENDIF
```

The above pseudo code implements the expel zone where a point is only allowed to move away from a stationary polyhedron once it ends up inside the expel zone.

3.2 Translational edge-edge

Implementing the expel zone for the translational edge-edge collision detection is no more complicated than for translational vertex-polygon collisions. When the planes of two polygons that share a stationary edge are expanded with e they intersect at a line. This line is used for the expel zone for translational edge-edge collision detection. First the fraction f_1 of the translation completed before the translating edge collides with the stationary edges is calculated. If this fraction is negative the edges move away from each other and there will be no collision. Next the

fraction f_2 of the translation completed before the translating edge collides with the line created by the expanded polygon planes is calculated. If this fraction f_2 is larger than 1 or f_1 is smaller than f_2 there is no collision. The expel zone for the translational edge-edge collision detection as described in section 2.3 is implemented in pseudo code as follows:

```

IF  $f_1 < 0$  THEN
   $f = 1$ 
ELSE
  IF  $f_2 \geq 1 \vee f_1 < f_2$  THEN
     $f = 1$ 
  ELSE
     $f = \max(f_2, 0)$ 
  ENDIF
ENDIF

```

As can be easily verified, the above pseudo code implements the expel zone for the translational edge-edge collision detection where the translating edge is only allowed to move away from the stationary edge once it is in the expel zone.

3.3 Rotational vertex-polygon

Implementing the expel zone for rotational vertex-polygon collision detection is a little bit more complicated. During rotation a point in the expel zone can first move deeper into the expel zone and later move out of the expel zone, or the other way around. Three different rotations of a point in the expel zone are shown in Fig. 2-4. The hatched lines at the bottom represent a stationary polyhedron. The dashed line represents the outer hull of the expel zone.

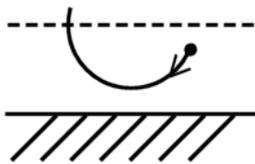


Fig. 2: Stop immediately.

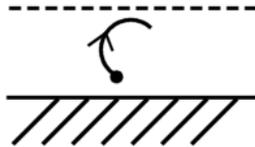


Fig. 3: Move up to top.

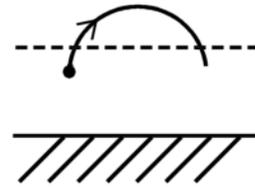


Fig. 4: Move out and collide.

The expel zone is implemented such that any rotating point in the expel zone can only move away from the stationary polyhedron at any time. In the situation shown in Fig. 2 the point is not allowed to rotate at all because the initial direction of motion leads the point deeper into the expel zone. The point in Fig. 3 is only allowed to rotate up to the point where it is furthest away from the stationary polyhedron. In the situation shown in Fig. 4 the point first moves out of the expel zone and then collides with the expel zone. In this case only the rotation up to the point of collision with the outer hull of the expel zone is allowed.

To implement this expel zone the direction of motion at the initial position, and the fraction at which the point is furthest away from the polyhedron are required. The derivative of the parametric formulation for the rotation can be used to acquire both. The derivative of the parametric formulation given in section 2.4 is:

$$v_1 \cos \alpha - v_2 \sin \alpha$$

The derivative at $\alpha = 0$ (called s) gives the direction of motion at the initial position. The equation for the derivative can be set to zero to find the fraction of rotation for which the point is furthest away from the polygon plane. The same strategy as described in section 2.4 is used for solving this equation. If the point moves away from the polygon in the initial position then the smallest fraction larger than zero, which is calculated by setting the derivative to zero, is called d and is the position where the point is furthest away from the polygon. f_1 and f_2 are defined as in section 2.4,

however, they are calculated with the polygon plane expanded with e . The expel zone for the rotational vertex-polygon collision detection is implemented in pseudo code as follows:

```

IF the vertex is in the expel zone THEN
  IF  $s < 0$  THEN
     $f = 0$ 
  ELSE
    IF  $f_1$  and  $f_2$  are valid THEN
       $f = \min(\max(f_1, f_2, 0), 1)$ 
    ELSE
       $f = \min(\max(d, 0), 1)$ 
    ENDIF
  ELSE
    IF  $f_1$  and  $f_2$  are valid THEN
       $f = \min(\max(f_1, \text{sign}(-f_1)), \max(f_2, \text{sign}(-f_2)), 1)$ 
    ELSE
       $f = 1$ 
    ENDIF
  ENDIF
ENDIF

```

To verify if the vertex starts inside the expel zone the distance of the initial vertex position to the polygon plane is used which is trivially calculated. The fractions f_1 and f_2 do not have to be valid at all times because the vertex might not collide with the expanded polygon at all during the rotation. The above pseudo code handles this situation checking if both fractions are valid. In case there is only a single collision with the expanded polygon plane both fractions are assumed to be the same.

3.4 Rotational edge-edge

The expel zone for rotational edge-edge collision detection works very similar to the expel zone for rotational vertex-polygon collisions. Just like with the translational edge-edge collision detection an additional line is used which is the intersection of two expanded polygons that share a stationary edge. For the rotating line to start in the expel zone it has to start between the stationary edge and intersection line of the expanded polygons. This can be verified by looking at the signs of the permuted inner products of the Plücker coordinates of the rotating edge with the stationary edge, and with the intersection line of the expanded polygons. Just like with the rotational vertex-polygon collision detection the derivative of the parametric formulation for the rotation of the edge is used to find the fraction of rotation for which the edges are furthest apart. Furthermore the pseudo code which implements the expel zone for the rotational edge-edge collision detection is the same as for the rotational vertex-polygon collisions.

3.5 Expel zone in practice

Whether or not a vertex or edge starts in the expel zone of another polyhedron needs to be calculated for the rotational collision detection. These calculations have numerical error themselves. However, these calculations are minor and the maximum numerical error of these calculations can be determined. This maximum error can be used as a tolerance to favor a vertex or edge to be in the expel zone. Assuming a vertex or edge is in the expel zone while it is still a tiny distance away, does not change the general behavior of the expel zone.

For the expel zone for edge-edge collision detection the intersection line of two expanded planes is used. Such a line cannot be created for dangling edges because these edges are used by only a single polygon. However, in practice the dangling edge can be expanded in the polygon plane away from the polygon center. The supporting line of this expanded edge can be used instead. The expel zone for vertex-polygon collision detection covers for the discontinuity in the expel zone caused by the expansion of the edge in the polygon plane. Furthermore dangling edges can usually be avoided and in most applications objects only use continuous surfaces that completely enclose a volume.

When two polygons sharing a stationary edge meet at a very sharp angle the intersection line of the expanded polygon planes can be pushed out quite far from the stationary edge. As a result the expel zone can reach out relatively far at such sharp edges. The intersection line can however, be moved back towards the stationary edge such that the in between distance becomes smaller. There are no problems for the behavior of the expel zone due to the discontinuity in the expel zone caused by moving the intersection line.

One might argue that objects can never “exactly” touch each other due to the expel zone. However, the expel zone is tiny because it only needs to be large enough to contain the error due to rounding in the calculations. Furthermore every object can be decreased in size with half the size of the expel zone before being used for collision detection, while keeping the visual representation of the objects the same. Objects can then “exactly” touch each other as far as the accumulated error of the collision detection calculations allows the positions of the objects to be accurate enough.

4. Narrow phase optimizations

In collision detection algorithms there is usually a distinction between a broad phase and a narrow phase. The collision detection calculations as described above are typically considered part of the narrow phase. In this phase the exact collisions are calculated. Although these calculations are straightforward and not very expensive in isolation, it can be quite expensive to test each pair of features of two polyhedra. These calculations can be minimized by first using a broad phase where possible collision regions are localized using bounding volume and spatial subdivision techniques. Before looking into this broad phase some optimizations are suggested for the narrow phase.

The polyhedra used for collision detection are often continuous surfaces. As such a lot of edges and vertices are shared between multiple polygons. Using an appropriate data structure the collision detection calculations for shared edges and shared vertices only need to be performed once.

Sidedness relations between three dimensional lines are used to verify if a vertex collides between the edges of a polygon, and to verify if two edges collide between their bounds. As mentioned earlier, a lot of the same sidedness calculations are used both to verify the boundaries for vertex-polygon and edge-edge collisions. When the memory is available to cache the results of these sidedness relations, there should be no reason to perform such computations more than once. Only the sign of the sidedness relation is used to verify the boundaries for collisions. Therefore the results of these relations can be efficiently stored in a bit cache, which keeps the memory requirements to a minimum.

An arbitrary polyhedron which is not convex can have so called internal edges. Three such internal edges are shown as fat lines in Fig. 5.

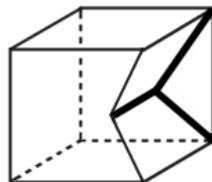


Fig. 5: Internal edges.

The two polygons that share an internal edge face towards each other. These internal edges can never collide with edges of another polyhedron and do not have to be considered for collision detection. If a vertex of a polyhedron is a boundary of only internal edges, the vertex can also never collide with polygons of another polyhedron. In Fig 5. the three internal edges meet at a vertex that does not have to be considered for collision detection. If the shape of a polyhedron used for collision detection does not change, the polyhedron can be preprocessed and the internal edges can be identified prior to any collision detection calculations. Although the internal edges do not have to be considered for collision detection, they are still required to verify if a vertex collides with a polygon plane between the polygon edges.

Collisions are assumed to only occur with the front side of polygons. Therefore polygons that are facing away from the direction of motion do not have to be considered for collision detection. For the translational collision detection the direction of motion is constant and culling back facing polygons is trivial. For the rotational collision detection back face culling is somewhat more complicated.

The computations for the rotational collision detection involve calculating the arc tangent of half the rotation angle. This is typically an expensive operation on a computer. Fortunately the tangent is a monotonic increasing function on the interval $[-\pi/2, \pi/2]$. Furthermore only collisions for rotations in the range $[-\pi, \pi]$ are considered. Therefore the first collision can be found by sorting the collisions based on the tangent of half the rotation angle, instead of sorting on the rotation angle itself. As a result there is no need to ever calculate the angle of rotation using an arc tangent calculation, because the sine and cosine for a specific rotation angle can be trivially derived from the tangent of half the rotation angle. The rotational collision detection also requires the calculation of square roots. There are very fast and relatively accurate algorithms available for square root calculations. Calculating the square root, represented by a 32 bit floating-point number, requires no more than two Newton-Rapson iterations [52].

5. Broad phase

The exact collision detection calculations can be minimized by first localizing possible collision regions using bounding volume and spatial subdivision techniques. Bounding volumes can be used to easily determine if objects can possibly collide. Only when the bounding volumes of two objects or their motions overlap, there is a need to perform narrow phase collision detection. Commonly used bounding volumes are spheres, oriented bounding boxes (OBB), axis aligned bounding boxes (AABB), and discrete orientation polytopes (*k*-dops). Spheres are very easy to construct and overlap tests only require the computation of the distance between the center of the spheres. Overlaps between OBBs are rapidly determined by performing 15 simple axis projection tests. The intersection tests between AABBs are not orientation dependent and even faster. *K*-dops, can also be used which are bounding volumes that are convex polytopes whose facets are determined by halfspaces with outward normals coming from a small fixed set of *k* orientations.

Hierarchies of bounding volumes can be used to easily and rapidly localize possible collision regions. Hierarchies of spheres or bounding boxes are commonly used in collision detection algorithms. General spatial subdivisions like regular grids, octrees and BSP-trees can also be used to easily determine which objects or parts of objects can possibly collide.

Several approaches have been implemented and tested for the environments in the computer game DOOM III. A combination of axis aligned bounding boxes and an axial BSP-tree turned out to be the most efficient solution. The motion of an object between two positions along its path is contained in an AABB. Only when two such AABBs of two objects intersect, the objects can possibly collide. Furthermore for each polygon the smallest AABB is computed which contains the whole polygon. A large polyhedron also uses an axial BSP-tree which is created from the bounding planes of the polygon bounding boxes. This BSP-tree is used to quickly determine for which polygons narrow phase collision detection needs to be calculated.

6. Results

In DOOM III the expel zone has a thickness of $e = 0.25$ units where 32 units is approximately one meter. This e is much larger than the maximum accumulated error due to rounding in collision detection calculations. However, this large e was chosen to be absolutely sure objects will never interpenetrate even when the initial positioning is somewhat sloppy. To get objects to exactly touch each other visually, some objects are decreased in size for collision detection while their visual representations stay the same.

In DOOM III on the average more than 30% of all edges turned out to be internal in both indoor and outdoor environments. These internal edges do not have to be considered for collision detection as described in section 4. Some speed is gained by preprocessing the polyhedra to identify and flag the internal edges.

Several physics simulations have been implemented in DOOM III among which one using an impulse based approach [53]. Rigid bodies never interpenetrate and the collision detection system can easily be used for an impulse based physics simulation of a brick wall being blown apart. Independent from the update frequency the bricks do not sink into each other and the simulation does not become unstable or "blow up". Fig. 6 shows a brick wall after being shot at several times.

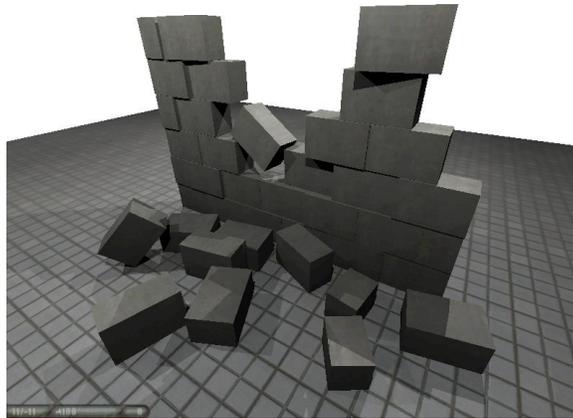


Fig. 6: Brick wall simulated with impulse based physics.

A constraint force based physics simulation has been implemented as well, which allows for the simulation of complex articulated figures. A Lagrangian multiplier approach is used to calculate constraint forces in combination with the collision detection algorithm described in this paper to simulate complex articulated figures without interpenetration.

In DOOM III the same collision detection is also used for player movement. Inaccuracies in the collision detection would be noticeable immediately, because a player receives very direct and accurate feedback from the game. The collision detection presented in this paper is very suitable for the simulation of player movement.

Degenerate polyhedra like a single polygon and a single point are trivially handled by the presented algorithm. Single polygons can be used to simulate simple objects like, for instance, the debris from explosions or particles with minimal computational cost. The translational collision detection for a single point is very fast and can easily be used for general visibility tests that are very common in computer games.

7. Conclusion

In a computer game like DOOM III an exact collision detection algorithm as described in this paper is very much preferred over, for instance, the more commonly used algorithms based on multiple interference detection. The physically simulated objects never interpenetrate and objects in contact are not visibly separated. Objects cannot pass through each other no matter how fast they move relative to each other. The algorithm also handles arbitrary polyhedra which allows the visual geometry to be used for collision detection. Furthermore, the algorithm uniformly handles degenerate polyhedra like single polygons and points which is useful for particle collisions and general visibility tests. The same collision detection algorithm can also be used to retrieve contact points between polyhedra in contact, and the algorithm can be used for robust and accurate simulation of the player movement through the game environment.

8. Literature

Surveys

- [0] Collision detection between geometric models: a survey
Ming C. Lin, Stefan Gottschalk
University of North Carolina, 1998
Proceedings of IMA Conference on Mathematics of Surfaces, 1998
- [1] Collision Detection: Algorithms and Applications
Ming C. Lin, Dinesh Manocha, Jon Cohen, Stefan Gottschalk
U.S. Army Research Office and University of North Carolina, 1998
- [2] 3D Collision Detection: A Survey
P. Jimenez, F. Thomas, C. Torras
Institut de Robotica i Informatica Industrial, Barcelona, Spain, 2000

Space-time Intersection Tests

- [3] Bintree, CSG Trees, and Time
Hanan Samet, Tamminen Markku
Proceedings of SIGGRAPH Computer Graphics, Vol. 19, No. 3, pp. 121-130, July 1985
- [4] Collision Detection by Four-Dimensional Intersection Testing
S. Cameron
IEEE Transaction on Robotics and Automation, Vol. 6, No. 3, pp. 291-302, June 1990
- [5] Space-time bounds for collision detection
Philip M. Hubbard
Technical Report CS-93-04, Dept. Computer Science, Brown University, 1993
- [6] Interactive collision detection
Philip M. Hubbard
Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality, pp. 24-31, 1993
- [7] Real-time four-dimensional collision detection for an industrial robot manipulator
Harry H. Cheng
Journal of Applied Mechanisms & Robotics, Vol. 2, No. 2, pp. 20-33, April 1995

Swept volume interference detection

- [8] A Safe Swept Volume Method for Collision Detection
Jean-Claude Latombe
Sixth International Symposium, Robotics Research, Cambridge USA, 1995
- [9] Dynamic Plane Shifting BSP Traversal
Stan Melax
Game Developer Conference, 2001

Multiple Interference detection

- [10] A Fast Procedure for Computing the Distance Between Complex Objects in Three-dimensional Space
E. G. Gilbert, D. W. Johnson, S. S. Keerthi
IEEE Journal of Robotics and Automation, Vol. 4, No. 2, pp. 193-203, 1988
- [11] Efficient Collision Detection for Animation and Robotics
Ming C. Lin
Thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1993.
- [12] Efficient Contact Determination Between Geometric Models
Ming C. Lin, Dinesh Manocha
TR94-024, August 1993
- [13] Interactive and Exact Collision Detection for Large-Scaled Environments
Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, Madhav K. Ponamgi
Technical report, University of North Carolina at Chapel Hill, March 1994.
ACM SIGGRAPH, 14-11, 1994
- [14] I-Collide: An Interactive and Exact Collision Detection System for Large-Scale Environments
Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, Madhav K. Ponamgi
Proceedings ACM Interactive 3D Graphics Conference, pp. 189-196, 1995
- [15] Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs
M. Held, J.T. Klosowski, J.S.B. Mitchell
7th Canadian Conference on Computational Geometry, Québec City, Québec Canada, pp. 205-210, August 10-13, 1995
- [16] Efficient Collision Detection for Real-time Simulated Environments
P.J. Dworkin
Thesis, Massachusetts Institute of Technology, June 1994
- [17] OBBTree: A Hierarchical Structure for Rapid Interference Detection
S. Gottshalk, Ming C. Lin, D. Manocha
Proceedings of ACM Siggraph, May 1996
- [18] Enhancing GJK: Computing Minimum and Penetration Distances between Convex Polyhedra
S. Cameron
International Conference Robotics & Automation, April 1997
- [19] V-Clip: fast and robust polyhedral collision detection
Brian Mirtich
TR-97-05, Mitsubishi Electric Research Lab, Cambridge, MA, July 1997
ACM Transactions on Graphics, Vol. 17, No. 3, pp. 177-208, July 1998
- [20] Rigid Body Contact: Collision Detection to Force Computation
Brian Mirtich
TR-98-01, Mitsubishi Electric Research Laboratory, March 1998
- [21] Efficient Algorithms for Two-Phase Collision Detection
Brian Mirtich
TR-97-23, Mitsubishi Electric Research Lab, Cambridge, MA, December 1997
- [22] Rigid Body Contact: Collision Detection to Force Computation
Brian Mirtich
TR-98-01, Mitsubishi Electric Research Lab, March 1998
- [23] Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs
J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, K. Zikan
IEEE Transactions on Visualization and Computer Graphics, Vol. 4, No. 1, March 1998
- [24] Efficient collision detection for interactive 3D graphics and virtual environments
J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, K. Zikan
IEEE Transactions on Visualization and Computer Graphics, Vol. 4, No. 1, pp. 21-36, May 1998
- [25] A System for Interactive Proximity Queries On Massive Models
A. Wilson, E. Larsen, D. Manocha, Ming C. Lin
Technical report, TR98-031, Computer Science Department, University of North Carolina at Chapel Hill, 1998
- [26] Accelerated Collision Detection for VRML
Thomas C. Hudson, Ming C. Lin, Jonathan Cohen, Stefan Gottschalk, Dinesh Manocha
Proceedings 2nd Annual Symposium on the Virtual Reality Modeling Language, Monterey, CA, USA,

BIBLIOGRAPHY 176, 1997.

- [26] Kinetic Collision Detection Between Two Simple Polygons
Julien Basch, Je Erickson, Leonidas J. Guibas, John Hershberger, Li Zhang
ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms), 1998
- [27] Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions
David Eppstein, Jeff Erickson
Discrete & Computational Geometry, March 1998
Symposium on Computational Geometry, pp. 58-67, 1998
- [28] A Framework for Fast and Accurate Collision Detection for Haptic Interaction
Arthur D. Gregory, Ming C. Lin, Stefan Gottschalk, Russell Taylor
IEEE Virtual Reality Conference, pages 38--45, 1999
- [29] Contact Determination for Real-time Haptic Interaction in 3D Modeling, Editing and Painting
Ming C. Lin, Arthur Gregory, Stephen Ehmann, Stephan Gottschalk, Russ Taylor
Technical report, Computer Science Department, University of North Carolina at Chapel Hill, 1999
- [30] A Fast and Robust GJK Implementation for Collision Detection of Convex Objects
Gino van den Bergen
Journal of Graphics Tools (JGT), Vol. 4, No. 2, pp. 7-25, 1999
- [31] Fast and Accurate Collision Detection for Haptic Interaction Using a Three Degree-of-Freedom Force-Feedback device
Arthur D. Gregory, Ming C. Lin, Stefan Gottschalk, Russell Taylor
Journal of Computational Geometry, Vol. 15, No. 1-3, pp. 69-89, 2000
- [32] Accelerated Proximity Queries Between Convex Polyhedra By Multi-Level Voronoi Marching
S.A. Ehmann, Ming C. Lin
Technical report, Computer Science Department, University of North Carolina at Chapel Hill, 2000
- [33] Accelerated Proximity Queries Between Convex Polyhedra By Multi-Level Voronoi Marching
S.A. Ehmann, Ming C. Lin
Technical report, Computer Science Department, University of North Carolina at Chapel Hill, 2000
- [34] Fast Distance Queries with Rectangular Swept Sphere Volumes
Eric Larsen, Stefan Gottschalk, Ming C. Lin, Dinesh Manocha
Technical report, Computer Science Department, University of North Carolina at Chapel Hill, 2000
- [35] Accurate and Fast Proximity Queries Between Polyhedra Using Convex Surface Decomposition
S.A. Ehmann, Ming C. Lin
TR-01-012, Department of Computer Science, University of North Carolina at Chapel Hill, 2001
Proceedings EG 2001, Vol. 20, No. 3, pp. 500-510, 2001
- [36] Optimizing the collision detection pipeline
Gabriel Zachmann
Proceedings of the First International Game Technology Conference (GTEC), January 2001
- [37] Accurate and Fast Proximity Queries between Polyhedra Using Surface Decomposition
Stephen A. Ehmann and Ming C. Lin
Proceedings of Eurographics (Computer Graphics Forum), 2001
- [38] DEEP: Dual-space Expansion for Estimating Penetration Depth between convex polytopes
Young J. Kim, Ming C. Lin, Dinesh Manocha
IEEE International Conference on Robotics and Automation, May 11-15 2002

Trajectory Parameterization

- [39] Interference Detection among Solids and Surfaces
John W. Boyse
Communications of the ACM, Vol. 22, No. 1, pp. 3-9, January 1979
- [40] On finding collisions between polyhedra
John Canny
Proceedings of European Conference on Artificial Intelligence (ECAI), 1984
- [41] Collision detection for moving polyhedra
John Canny
IEEE Transactions Pattern Analysis and Machine Intelligence, Vol. 8, No. 2, pp. 200-209, 1986

- [42] Polynomial Time Collision Detection for Manipulator Paths specified by Joint Motions
Achim Schweikard
IEEE Transactions on Robotics and Automation, Vol. 7, No. 6, pp. 865-870, 1991
- [43] Efficient collision detection for moving polyhedra
Elmar Schoemer, Christian Thiel
Universitat des Saarlandes, April 1995
ACM 11th Annual Symposium on Computational Geometry, pages 51-60, June 1995
- [44] Exact Geometric Collision Detection
Elmar Schoemer, Juergen Sellen, Markus Welsch
Universitat des Saarlandes, 1995
7th Canadian Conference on Computational Geometry, pages 211-216, August 1995.
- [45] Subquadratic algorithms for the general collision detection problem
Elmar Schoemer, Christian Thiel
Universitat des Saarlandes, December 1995
12th European Workshop on Computational Geometry, pages 95-101, March 1996
- [46] An Algebraic Solution to the Problem of Collision Detection for Rigid Polyhedral Objects
S. Redon, A. Kheddar, S. Coquillart
Proceedings of IEEE International Conference on Robotics and Automation, pp 3733-3738, April 2000
- [47] CONTACT: Arbitrary in-between motions for collision detection
S. Redon, A. Kheddar, S. Coquillart
Proceedings of IEEE ROMAN'2001, Sep. 2001

Misc.

- [48] Analytical Geometry of Three Dimensions
D.M.Y. Sommerville
Cambridge University Press, 1959
- [49] Primitives for computational geometry
Jorge Stolfi
Technical Report 36, DEC SRC, 1989
- [50] Oriented Projective Geometry: A Framework for Geometric Computations
Jorge Stolfi
Academic Press, 1991.
- [51] Methods of Algebraic Geometry, Volume 1
W.V.D. Hodge, D. Pedoe
Cambridge, 1994.
ISBN 0-521-469007-4 Paperback
- [52] Computing the Inverse Square Root
Ken Turkowski
Graphics Gems V
Morgan Kaufmann Publishers, 1st edition, January 15 1995
ISBN: 0125434553
- [53] Impulse-based Dynamic Simulation of Rigid Body Systems
Brian Mirtich
PhD Thesis, University of California, Berkeley, 1996