Ray Casting against General Convex Objects with Application to Continuous Collision Detection

GINO VAN DEN BERGEN

Playlogic Game Factory
Breda, Netherlands
gino@acm.org

June 15, 2004

Abstract

This paper presents a new algorithm for computing the hit point and normal of a ray and a general convex object. The algorithm is loosely based on the the Gilbert-Johnson-Keerthi algorithm for computing the distance between convex objects in the sense that it is applicable to the same family of convex objects and uses the same subalgorithm for computing the closest point of a simplex. Since this family of convex objects includes objects constructed by Minkowski addition, this algorithm can be used for finding the earliest time two objects that move at a constant linear velocity come in contact of one another. In this way, this ray-casting algorithm is applicable to a simplified form of continuous collision detection.

1 Introduction

The Gilbert-Johnson-Keerthi distance algorithm (GJK) is an iterative method for computing the distance between convex objects [6]. The attractiveness of GJK lies in its simplicity, which makes it fairly easy to implement, and its applicability to a large family of convex objects. This family of convex objects includes common shape primitives, such as spheres, boxes, and cylinders, as well as convex hulls

and Minkowski sums of convex objects. GJK is used extensively in collision detection [11]; however, up until now, it has been used mostly for static collision tests at discretely sampled time steps.

A problem that occurs when performing collision detection in this way is that collisions of fast moving objects, such as bullets fired from a gun, may be detected too late or not at all. This temporal-aliasing effect can be reduced by taking smaller time steps, however, the added computational cost of running your simulations at a higher frequency is often too high to make such an approach feasible.

This suggests that for fast moving objects we need to solve the collision detection problem as an intersection test in continuous four-dimensional space (space-time). The object placements in between two sampled instances of time are not known but can be obtained by interpolating the sampled placements. The four-dimensional objects that we test for intersection are the result of extruding the three-dimensional objects along their interpolated placements over the time interval. Solutions to the four-dimensional intersection detection problem have been presented for a restricted class of objects and motions [1, 2, 3, 7, 9, 4, 8]. However, in many common cases, the four-dimensional object that is the extrusion of a three-dimensional object in motion is often geometrically too complex to make a four-dimensional intersection test at interactive rates computationally feasible. For instance, it is hard to test the space-time extrusion of a spinning object, such as a propellor or a fan, for intersection with the extrusion of another moving object.

Often, we can get away with the following simplified four-dimensional intersection test. When the angular velocity of the moving objects is low, that is, orientations do not change a lot in-between frames, then a good approximation of the trajectory of such objects can be obtained by changing the orientations instantaneous at fixed time steps. Thus we only interpolate the positions of the objects in-between frames. For such trajectories, the time of collision, the point of contact, and a contact normal can be computed by performing a ray cast on the *configuration space obstacle* (CSO) of the objects. The CSO of two objects A and B is the set of all vector differences of a point in A and a point in B. The contact point corresponds to the point where the ray enters the CSO, and the contact normal is the normal to the boundary of the CSO at this point. Figure 1 shows an example of the ray test for a box and a moving sphere.

For many pairs of shape types, the CSO's shape can not be represented explicitly in a simple way. However, the ray cast algorithm we propose in this paper does not require an explicit representation of the CSO. As GJK, it uses a support mapping to read the geometry of the object, and is therefore applicable to the same family of objects.

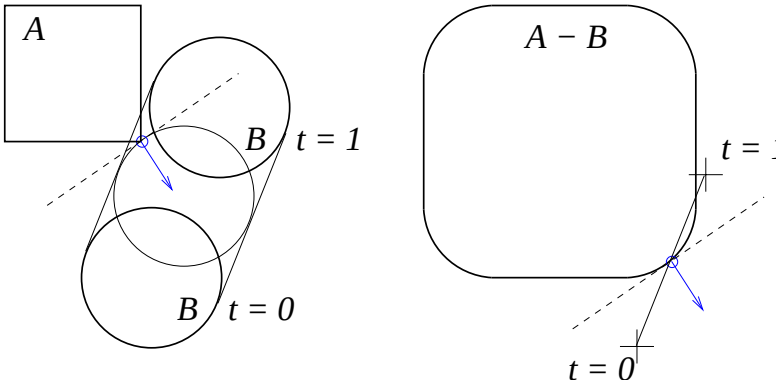

Figure 1: Computing a contact point (open dot) and contact normal (arrow) of box and a moving sphere by performing a ray test on the CSO of the objects.

The rest of this paper is organized as follows. In Section 2, we present a first iterative method for performing ray casts on general convex objects. This algorithm is an initial step towards the GJK-based ray cast presented in Section 4. The first method has little practical value. Its main purpose is to explain the concepts and help us prove certain properties of the GJK-based method. Since the latter method relies heavily on concepts used in GJK, we present an overview of the GJK algorithm in Section 3.

2 A First Solution

Let for a point $\mathbf{s}$ and vector $\mathbf{r}$, the ray R be given by

$$R = \{\mathbf{s} + \lambda\mathbf{r} : \lambda \geq 0\}.$$

The points $\mathbf{s}$ is the *source*, and the vector $\mathbf{r}$ is the *direction* of the ray. A ray cast of R onto an object C is a query that returns the smallest $\lambda \geq 0$ for which the point $\mathbf{x} = \mathbf{s} + \lambda\mathbf{r}$ is contained in C. This parameter is denoted by λ_{hit}, and the corresponding point $\mathbf{x}$ is called the *hit spot*. If the source is not contained in C, thus $\lambda_{\text{hit}} > 0$, then the hit spot must be a point on the boundary of C. In that case, we can define a normal at the hit point.

A non-zero vector $\mathbf{n}$ is a *normal* at a point $\mathbf{p}$ on the boundary of object C if

$$\mathbf{n} \cdot (\mathbf{x} - \mathbf{p}) \leq 0, \text{ for all } \mathbf{x} \in C.$$

A normal exists for all points on the boundary as long as C is convex. For a given point $\mathbf{p}$ on the boundary, a normal at $\mathbf{p}$ may not necessarily be unique. Normals that differ only in magnitude are considered equal, so here, uniqueness refers to the direction of the normal. A unique normal exists only for points where the boundary manifold is smooth (differentiable). Examples of non-smooth boundary points are vertices and edges of polytopes. The algorithm described here does not require the boundary manifold to be smooth, and returns an arbitrary normal for non-smooth boundary points.

Let us make a few observations. For $\mathbf{n}$ a normal and $\mathbf{p}$ a point on the boundary of C, we know that all points $\mathbf{x}$ for which $\mathbf{n} \cdot (\mathbf{x} - \mathbf{p}) > 0$ are not contained in C. Now, let $\mathbf{x} = \mathbf{s} + \lambda\mathbf{r}$ be a point of ray R. We find that the section of the ray corresponding to

$$\lambda\mathbf{n} \cdot \mathbf{r} > \mathbf{n} \cdot (\mathbf{p} - \mathbf{s}),$$

is not contained in C, and thus cannot contain the hit spot. More specifically:

1. If $\mathbf{n} \cdot \mathbf{r} > 0$, then the section of the ray for which $\lambda > \mathbf{n} \cdot (\mathbf{p} - \mathbf{s})/\mathbf{n} \cdot \mathbf{r}$ can be rejected. Thus, $\mathbf{n} \cdot (\mathbf{p} - \mathbf{s})/\mathbf{n} \cdot \mathbf{r}$ is an upper bound for the hit spot.

2. If $\mathbf{n} \cdot \mathbf{r} < 0$, then the section of the ray for which $\lambda < \mathbf{n} \cdot (\mathbf{p} - \mathbf{s})/\mathbf{n} \cdot \mathbf{r}$ can be rejected. Thus, $\mathbf{n} \cdot (\mathbf{p} - \mathbf{s})/\mathbf{n} \cdot \mathbf{r}$ is a lower bound for the hit spot.

3. If $\mathbf{n} \cdot \mathbf{r} = 0$ and $\mathbf{n} \cdot (\mathbf{p} - \mathbf{s}) < 0$ then the complete ray can be rejected.

Our strategy for computing λ_{hit} is to iteratively clip sections of R that do not contain the hit spot, until we arrive at the hit spot.

Let C be convex and let $\mathbf{x}_i = \mathbf{s} + \lambda_i\mathbf{r}$ be the current best lower bound for the hit spot. Thus, we already have rejected the section of the ray given by $0 \leq \lambda < \lambda_i$. Now, let $\mathbf{c}_i$ be the point of C closest to $\mathbf{x}_i$, then either

1. $\mathbf{c}_i = \mathbf{x}_i$, that is, $\mathbf{x}_i$ is contained in C, or

2. $\mathbf{c}_i$ is on the boundary and $\mathbf{n}_i = \mathbf{x}_i - \mathbf{c}_i$ is a normal at $\mathbf{c}_i$[1].

In the first case, $\mathbf{x}_i$ must be the hit spot and we are done. For the second case, we will show that either the ray does not hit the object, or there exists a better lower bound for the hit spot than the current one. Recall that we can reject the section of the ray defined by

$$\lambda\mathbf{n}_i \cdot \mathbf{r} > \mathbf{n}_i \cdot (\mathbf{c}_i - \mathbf{s}).$$

[1] Although intuitively this seems obvious, a formal proof for this theorem demands some consideration (See Lemma 4.1 in [11])

Since $\mathbf{n}_i = \mathbf{x}_i - \mathbf{c}_i$ and $\mathbf{s} = \mathbf{x}_i - \lambda_i \mathbf{r}$ this is equivalent to

$$\lambda \mathbf{n}_i \cdot \mathbf{r} > \lambda_i \mathbf{n}_i \cdot \mathbf{r} - \|\mathbf{n}_i\|^2.$$

First of all, if $\mathbf{n}_i \cdot \mathbf{r} = 0$, then the entire ray can be rejected since $-\|\mathbf{n}_i\|^2 < 0$. Assume $\mathbf{n}_i \cdot \mathbf{r} \neq 0$ and let $\lambda_{i+1} = \lambda_i - \|\mathbf{n}_i\|^2 / \mathbf{n}_i \cdot \mathbf{r}$. Then, for $\mathbf{n}_i \cdot \mathbf{r} > 0$, we find that λ_{i+1} is an upper bound. Since $\lambda_{i+1} < \lambda_i$, and we already have rejected the section up to λ_i, the entire ray can be rejected. For $\mathbf{n}_i \cdot \mathbf{r} < 0$, we find that λ_{i+1} is a lower bound. Since $\lambda_{i+1} > \lambda_i$, we have found a better lower bound for the hit spot. A summary of this iterative method is described in pseudo-code in Algorithm 1.

Algorithm 1 An iterative method for performing a ray cast of a ray $\mathbf{s} + \lambda \mathbf{r}$ against a convex object C. For positive results, this algorithm terminates with λ being the hit parameter, $\mathbf{x}$, the hit spot, and $\mathbf{n}$, the normal at $\mathbf{x}$.

```
λ ← 0;
x ← s;
n ← 0;
c ← "the point of C closest to x";
while not "x is close enough to c" do
begin
   n ← x − c;
   if n · r ≥ 0 then return false
   else
   begin
      λ ← λ − ‖n‖²/n · r;
      x ← s + λr;
      c ← "the point of C closest to x"
   end
end;
return true
```

The property $\lambda_i < \lambda_{i+1} \leq \lambda_{\text{hit}}$ is a necessary yet not sufficient condition for global convergence. In order to show that, in case of a hit, $\mathbf{x}_i$ indeed approaches the hit spot for $i \to \infty$, we need to show that the mapping from λ_i to λ_{i+1} is

continuous at all $\lambda < \lambda_{\text{hit}}$. Recall that $\lambda_{i+1} = \lambda_i - \|\mathbf{n}(\lambda_i)\|^2/\mathbf{n}(\lambda_i) \cdot \mathbf{r}$, where

$$\begin{aligned} \mathbf{n}(\lambda) &= \mathbf{x}(\lambda) - \mathbf{c}(\mathbf{x}(\lambda)), \\ \mathbf{x}(\lambda) &= \mathbf{s} + \lambda\mathbf{r}, \text{ and} \\ \mathbf{c}(\mathbf{x}) &= \text{“the point of } C \text{ closest to } \mathbf{x}\text{”}. \end{aligned}$$

For convex C, the point of C closest to $\mathbf{x}$ is unique for any point $\mathbf{x}$, so $\mathbf{c}(\mathbf{x})$ is a proper mapping. Since, in case of a hit, $\mathbf{n}(\lambda) \cdot \mathbf{r} < 0$ for all $\lambda < \lambda_{\text{hit}}$, it suffices to show that the mapping $\mathbf{n}(\lambda)$ is continuous on this domain.

First of all, it is clear that $\mathbf{x}(\lambda)$ is continuous. The mapping $\mathbf{c}(\mathbf{x}(\lambda))$ is continuous as well. The proof of this theorem may not interest the casual reader and is postponed to Appendix A. Since the difference of two continuous functions is continuous as well, we conclude that $\mathbf{n}(\lambda)$ is continuous.

If the ray misses C, then $\mathbf{n}(\lambda) \cdot \mathbf{r} = 0$ for some λ, and thus the mapping from λ_i to λ_{i+1} is discontinuous at this λ. However, since $\mathbf{n}(\lambda) \cdot \mathbf{r}$ is monotonically nondecreasing, as shown in Appendix A, we are assured of the fact that $\mathbf{n}(\lambda) \cdot \mathbf{r} < 0$ for all $\lambda \leq \lambda_i$, as long as $\mathbf{n}_i \cdot \mathbf{r} < 0$. So, a discontinuity is encountered only when $\mathbf{n}_i \cdot \mathbf{r} \geq 0$ in which case Algorithm 1 terminates.

We see that each iteration brings us closer to either the hit spot or a condition for rejection. In case of a hit, the sequence $\{\lambda_i\}$ converges to λ_{hit}. This means that, for any positive ε, it takes a finite number of iterations to find a point on the ray at a distance of less than ε from the hit point. Note that not necessarily $\lambda_i = \lambda_{\text{hit}}$ for some i, that is, for some objects we can get arbitrarily close to, but still may never reach the hit point. However, if C is a polytope, then λ_{hit} is always reached in a finite number of iterations.

Up until now, we have been avoiding the question of how to find the point of a general convex object closest to a given point. Our best bet for this operation would be the Gilbert-Johnson-Keerthi algorithm (GJK) [5]. GJK is an iterative method for computing the closest points pair of two convex objects. However, the idea of having an iterative method nested inside another does not sound too appealing from a performance point of view, let alone the difficulty of establishing a termination condition for the inner GJK loop. So, instead of having nested loops, we will combine the two iterative methods into a single iterative method in Section 4, but first, let us briefly discuss the GJK algorithm.

3 Overview of GJK

The Gilbert-Johnson-Keerthi algorithm (GJK) is essentially a descent method for approximating the distance between two general convex objects. The original paper discusses the use of GJK for polytopes only [6], however, with some minor adaptations of the termination condition, GJK is applicable to convex objects in general [5]. In this section we will briefly discuss the GJK algorithm in order to get us going for the next section. For an in-depth discussion of GJK, the reader is referred to [11].

The versatility of GJK lies in the fact that it uses support mappings for reading the geometry of convex objects. A *support mapping* for an object C is a function s_C that maps a vector $\mathbf{v}$ to a point of C, according to

$$s_C(\mathbf{v}) \in C \quad \text{such that} \quad \mathbf{v} \cdot s_C(\mathbf{v}) = \max\{\mathbf{v} \cdot \mathbf{x} : \mathbf{x} \in C\}.$$

The result of a support mapping for a given vector is called a *support point*. In order to use GJK for a given primitive shape type, we need to supply a support mapping for the type. Support mappings for objects obtained from primitive shape types by affine transformation, convex hulls, and Minkowski addition can be derived from the support mappings for the primitive types. A discussion of support mappings for commonly used shape types falls outside the scope of this article. The reader is referred to [11] for learning more about support mappings.

An important concept in GJK is the notion of *configuration space*. The distance query on two objects is transformed to a query on a single convex object called the *configuration space obstacle* (CSO). The CSO of objects A and B is the object

$$A - B = \{\mathbf{x} - \mathbf{y} : \mathbf{x} \in A, \mathbf{y} \in B\}.$$

It can be seen that the distance between A and B is equal to the distance between $A - B$ and $\mathbf{0}$, the origin of the configuration space. We denote the point closest to the origin of an object C as

$$v(C) \in C \quad \text{and} \quad \|v(C)\| = \min\{\|\mathbf{x}\| : \mathbf{x} \in C\}.$$

It follows that the distance between A and B can be expressed as

$$d(A, B) = \|v(A - B)\|.$$

Given support mappings for A and B, a support mapping s_{A-B} for $A - B$ can be found easily, since

$$s_{A-B}(\mathbf{v}) = s_A(\mathbf{v}) - s_B(-\mathbf{v}).$$

For now, let us assume we have support mappings for A and B.

GJK approximates the point $v(A - B)$ in the following way. In each iteration a simplex is constructed that is contained in $A - B$ and that lies closer to the origin than the simplex constructed in the previous iteration. A simplex is the convex hull of an affinely independent set of vertices. The simplices can have one to four vertices, so a simplex can be a single point, a line segment, a triangle, or a tetrahedron. We define W_k as the set of vertices of the simplex constructed in the k-th iteration, and $\mathbf{v}_k$ as $v(\mathrm{conv}(W_k))$, the point of the simplex closest to the origin. Initially, we take $W_0 = \emptyset$, and $\mathbf{v}_0$, an arbitrary point in $A-B$. Since $A-B$ is convex and $W_k \subseteq A-B$, we see that $\mathbf{v}_k \in A-B$, and thus $\|\mathbf{v}_k\| \geq \|v(A-B)\|$ for all $k \geq 0$. So, the length of $\mathbf{v}_k$ is an upper bound for the distance between A and B.

In each iteration we add a new support point $\mathbf{w}_k = s_{A-B}(-\mathbf{v}_k)$ as vertex to the current simplex W_k. Let $Y_k = W_k \cup \{\mathbf{w}_k\}$ be the new simplex. We take $\mathbf{v}_{k+1} = v(\mathrm{conv}(Y_k))$, the point closest to the origin of the new simplex. As W_{k+1}, we take the smallest set $X \subseteq Y_k$, such that $\mathbf{v}_{k+1}$ is contained in $\mathrm{conv}(X)$. It can be seen that exactly one such X exists, and that it must be affinely independent. So, while new vertices are being added to the simplex, earlier vertices, that are no longer necessary for supporting $\mathbf{v}_{k+1}$, are discarded.

GJK terminates as soon as $\mathbf{v}_k$ is close enough to $v(A-B)$. As upper bound for $\|\mathbf{v}_k - v(A-B)\|^2$ we use $\|\mathbf{v}_k\|^2 - \mathbf{v}_k \cdot \mathbf{w}_k$. It has been shown that this upper bound approaches zero when $\mathbf{v}_k \to v(A - B)$, and that the sequence $\{\mathbf{v}_k\}$ converges to $v(A - B)$ [6, 11]. Therefore, Algorithm 2, which describes the GJK distance algorithm in pseudo-code, must terminate in a finite number of iterations for any positive error tolerance ε.

For computing the point of a simplex $\mathrm{conv}(Y)$ closest to the origin, and for determining the smallest subsimplex that contains the closest point we use a sub-algorithm called Johnson's distance algorithm. Let $Y = \{\mathbf{y}_1, \ldots \mathbf{y}_n\}$ be the set of vertices of the simplex. Then, a point $\mathbf{v}$ of the simplex is described as a convex combination of Y in the following way.

$$\mathbf{v} = \sum_{i=1}^{n} \lambda_i \mathbf{y}_i \quad \text{where} \quad \sum_{i=1}^{n} \lambda_i = 1 \quad \text{and} \quad \lambda_i \geq 0.$$

The smallest $X \subseteq Y$ such that $\mathbf{v} \in \mathrm{conv}(X)$ is the set $X = \{\mathbf{y}_i : \lambda_i > 0\}$. In other words, the set X is found by discarding all the points $\mathbf{y}_i$ from Y for which $\lambda_i = 0$.

Now, let $\mathbf{v} = v(\mathrm{conv}(X))$ and X the corresponding minimum subsimplex.

Algorithm 2 The GJK distance algorithm.

$\mathbf{v} \leftarrow$ *"arbitrary point in* $A - B$*"*;
$W \leftarrow \emptyset$;
$\mathbf{w} \leftarrow s_{A-B}(-\mathbf{v})$;
while $\|\mathbf{v}\|^2 - \mathbf{v} \cdot \mathbf{w} > \varepsilon^2$ do
$\{\!\{$ $\mathbf{v}$ *is not close enough to* $v(A - B)$. $\}\!\}$
begin
  $Y \leftarrow W \cup \{\mathbf{w}\}$;
  $\mathbf{v} \leftarrow v(\mathrm{conv}(Y))$;
  $W \leftarrow$ *"smallest* $X \subseteq Y$ *such that* $\mathbf{v} \in \mathrm{conv}(X)$*"*;
  $\mathbf{w} \leftarrow s_{A-B}(-\mathbf{v})$;
end;
return $\|\mathbf{v}\|$

Then, also $\mathbf{v} = v(\mathrm{aff}(X))$, i.e., the point closest to the origin of the affine hull of X. The point $v(\mathrm{aff}(X))$ can be computed by solving a linear system of equations, since it is the unique point of the affine hull that is perpendicular to all vectors $\mathbf{y}_i - \mathbf{y}_j$, where $\mathbf{y}_i, \mathbf{y}_j \in X$. Johnson's algorithm uses a recursive formulation for the solution of the linear system. Let $X = \{\mathbf{y}_i : i \in I_X\}$, where $I_X \subseteq \{1, \ldots, n\}$. Then, we express $v(\mathrm{aff}(X))$ as

$$v(\mathrm{aff}(X)) = \sum_{i \in I_X} \lambda_i \mathbf{y}_i \quad \text{where} \quad \lambda_i = \frac{\Delta_i^X}{\Delta^X},$$

and Δ_i^X is defined recursively as

$$\begin{aligned} \Delta_i^{\{\mathbf{y}_i\}} &= 1 \\ \Delta_j^{X \cup \{\mathbf{y}_j\}} &= \sum_{i \in I_X} \Delta_i^X ((\mathbf{y}_k - \mathbf{y}_j) \cdot \mathbf{y}_i) \quad \text{for } j \notin I_X \text{ and any } k \in I_X, \end{aligned}$$

and finally, Δ^X is defined as

$$\Delta^X = \sum_{i \in I_X} \Delta_i^X.$$

The smallest $X \subseteq Y$ such that $\mathbf{v} \in \mathrm{conv}(X)$ can now be characterized as the subset X for which (i) $\Delta_i^X > 0$ for each $i \in I_X$, and (ii) $\Delta_j^{X \cup \{\mathbf{y}_j\}} \leq 0$, for all

$j \notin I_X$. Johnson's algorithm successively tests each nonempty subset X of Y until it finds one for which (i) and (ii) hold.

Since some or all vertices in Y_k reappear in Y_{k+1}, many vectors $\mathbf{y}_k - \mathbf{y}_j$ from the k-th iteration are also needed in the $k+1$-th iteration. So, a sensible implementation of GJK would cache these vectors for future iterations. Further performance gains can be obtained by caching also the values of the determinants Δ_i^X. For details on how to implement caching for these values we refer to [11].

4 GJK-Based Ray Cast

We are now ready to combine the two iterative methods described in Algorithm 1 and 2 into a single iterative method for performing a ray cast against general convex objects. Similar to GJK, the algorithm presented here uses only a support mapping for reading the geometry of the convex object, so it is applicable to the same family of convex objects as GJK.

In Algorithm 1, we use the normal $\mathbf{n}_i = \mathbf{x}_i - \mathbf{c}_i$ at the point $\mathbf{c}_i$ of C closest to $\mathbf{x}_i$ for finding λ_{i+1}. This normal can be expressed in terms of configuration space as $\mathbf{n}_i = v(\{\mathbf{x}_i\} - C)$. Generally, GJK will not return this normal in a finite number of iterations. However, it can generate a sequence $\{\mathbf{v}_k\}$ that converges to $\mathbf{n}_i$. For $k \geq 1$, the vector $\mathbf{v}_k$ represents the point closest to the origin of a simplex $\mathrm{conv}(\{\mathbf{x}_i\} - P_k)$, where P_k is the set of vertices of a simplex contained in object C. The points in P_k are generated by a support mapping of C. Let $\mathbf{p}_k = s_C(\mathbf{v}_k)$ be a support point of C for vector $\mathbf{v}_k$. Then, $\mathbf{v}_k$ is a normal at $\mathbf{p}_k$ by the definition of support mapping, and thus, the section of the ray corresponding to

$$\lambda \mathbf{v}_k \cdot \mathbf{r} > \mathbf{v}_k \cdot (\mathbf{p}_k - \mathbf{s})$$

can be rejected. So, although $\mathbf{v}_k$ is not necessarily equal to $\mathbf{n}_i$, it may help us in rejecting sections of the ray. Let $\mathbf{x}_i = \mathbf{s} + \lambda_i \mathbf{r}$ be the current best lower bound for the hit spot. If we substitute $\mathbf{s} = \mathbf{x}_i - \lambda_i \mathbf{r}$ in the above inequality, we find

$$\lambda \mathbf{v}_k \cdot \mathbf{r} > \lambda_i \mathbf{v}_k \cdot \mathbf{r} - \mathbf{v}_k \cdot \mathbf{w}_k, \quad \text{where} \quad \mathbf{w}_k = s_{\{\mathbf{x}_i\}-C}(-\mathbf{v}_k) = \mathbf{x}_i - \mathbf{p}_k.$$

Again, if $\mathbf{v}_k \cdot \mathbf{r} = 0$ and $\mathbf{v}_k \cdot \mathbf{w}_k > 0$, then the entire ray can be rejected. If $\mathbf{v}_k \cdot \mathbf{r} > 0$, then $\lambda_i - \mathbf{v}_k \cdot \mathbf{w}_k / \mathbf{v}_k \cdot \mathbf{r}$ is an upper bound for λ_{hit} In this case, we can also reject the entire ray if $\mathbf{v} \cdot \mathbf{w}_k > 0$. If $\mathbf{v}_k \cdot \mathbf{r} < 0$ then $\lambda_i - \mathbf{v}_k \cdot \mathbf{w}_k / \mathbf{v}_k \cdot \mathbf{r}$ is a lower bound. It is a better lower bound than λ_i if $\mathbf{v}_k \cdot \mathbf{w}_k > 0$. We see that in all three cases, progress is made if $\mathbf{v}_k \cdot \mathbf{w}_k > 0$.

Note that the sequence $\{\mathbf{v}_k\}$ generated by GJK can have a wild behavior, especially in the first few iterations, so it is not guaranteed that $\mathbf{v}_k \cdot \mathbf{w}_k$ is always positive. However, since $\|\mathbf{v}_k\|^2 - \mathbf{v}_k \cdot \mathbf{w}_k$ approaches zero, $\mathbf{v}_k \cdot \mathbf{w}_k$ must become positive in a finite number of iterations for any normal $\mathbf{n}_i \neq \mathbf{0}$. So, as long as $\mathbf{x}_i$ is not contained in C, we can either get closer to the hit spot or closer to a condition for rejection by performing a finite number of GJK iterations. Algorithm 3 describes the GJK-based ray cast in pseudo code.

Algorithm 3 GJK-based ray cast. For positive results, this algorithm terminates with λ being the hit parameter, $\mathbf{x}$, the hit spot, and $\mathbf{n}$, the normal at $\mathbf{x}$.

$\lambda \leftarrow 0$;
$\mathbf{x} \leftarrow \mathbf{s}$;
$\mathbf{n} \leftarrow \mathbf{0}$;
$\mathbf{v} \leftarrow \mathbf{x} -$ *"arbitrary point in* C*"*;
$P \leftarrow \emptyset$;
while $\|\mathbf{v}\|^2 > \varepsilon^2$ do
begin
  $\mathbf{p} \leftarrow s_C(\mathbf{v})$; {{ $\mathbf{v}$ *is a normal of* C *at* $\mathbf{p}$ }}
  $\mathbf{w} \leftarrow \mathbf{x} - \mathbf{p}$;
  if $\mathbf{v} \cdot \mathbf{w} > 0$ then
  begin
    if $\mathbf{v} \cdot \mathbf{r} \geq 0$ then return false
    else
    begin
      $\lambda \leftarrow \lambda - \mathbf{v} \cdot \mathbf{w} / \mathbf{v} \cdot \mathbf{r}$;
      $\mathbf{x} \leftarrow \mathbf{s} + \lambda \mathbf{r}$;
      $\mathbf{n} \leftarrow \mathbf{v}$
    end
  end;
  $Y \leftarrow P \cup \{\mathbf{p}\}$;
  $\mathbf{v} \leftarrow v(\mathrm{conv}(\{\mathbf{x}\} - Y))$;
  $P \leftarrow$ *"smallest* $X \subseteq Y$ *such that* $\mathbf{v} \in \mathrm{conv}(\{\mathbf{x}\} - X)$*"*
end;
return true

Note that in contrast to the GJK distance algorithm, the CSO $\{\mathbf{x}\} - C$ may change position during iterations. The CSO is translated along the vector $\mathbf{r}$ when-

ever $\mathbf{x}$ is updated. Updates of the lower bound for the hit spot result in a behavior that deviates from the normal behavior of GJK. First of all, $\|\mathbf{v}_k\|$ is no longer monotonically decreasing in k. Secondly, the same support point may be returned over multiple iterations.

With the original GJK algorithm, generating the same support point twice is theoretically impossible. Since it is a clear sign of numerical problems in a finite-precision implementation of GJK, this property can be exploited to exit gracefully should a support point ever reappear [10]. For the GJK ray cast, we can no longer rely on this property for signaling numerical problems. However, since the termination condition used in Algorithm 3 differs from the one in Algorithm 2, the support point check is no longer necessary, as we discovered in our single-precision (32-bit) floating-point implementation of the GJK ray cast.

Overall, the numerical behavior of the GJK ray cast is pretty decent. A known numerical issue with Johnson's algorithm is the fact that due to rounding the signs of the determinants Δ_i^X may be incorrect. This may result in failure to find a proper minimal subset X [6]. This problem occurs when Y is close to being affinely dependent, which usually happens when $\mathbf{v}_k$ is close to $v(\mathbf{x} - C)$. So, the best we can do in this situation is to simply exit the loop and return $\mathbf{x}$ as the current best lower bound for the hit spot.

Another issue that needs some attention in a finite-precision implementation is the choice of error tolerance ε in the termination condition. For vectors $\mathbf{v}_k$ close to zero the relative error due to rounding can be quite large. We have found that the error in the squared length of $\mathbf{v}_k$ is roughly proportional to the maximum squared distance between $\mathbf{x}$ and a point in P_k. So, for the numerical GJK ray cast we propose as termination condition,

$$\|\mathbf{v}_k\|^2 \leq \varepsilon_{\text{tol}} \max\{\|\mathbf{x} - \mathbf{p}\|^2 : \mathbf{p} \in P_k\},$$

where ε_{tol} is an order of magnitude larger that the machine epsilon of the used floating-point format. Choosing a tolerance ε_{tol} that is less than the machine epsilon will result in infinite looping, since the error in the computed $\|\mathbf{v}_k\|^2$ can become greater than $\varepsilon_{\text{tol}} \max\{\|\mathbf{x} - \mathbf{p}\|^2 : \mathbf{p} \in P_k\}$.

The convergence speed depends on the used shape types. We have found the worst convergence for smooth shapes such as spheres. For a sphere, the average number of iterations is roughly proportional to $-log(\varepsilon_{\text{tol}})$. For instance, an ε_{tol} of 10^{-6} results in eight iterations on average for random rays with a high hit probability. Polytopes usually take fewer iterations.

In the application of ray casting in continuous collision detection, the ray has a finite length. This feature can be exploited to improve the performance of the

ray cast. As soon as the lower bound λ becomes larger than the given maximum parameter $\lambda_{\max}$, Algorithm 3 may terminate returning a miss. By enabling an early out for finite rays, the average number of iterations drops dramatically in cases where the query rays are short with respect to the environment.

The possible translations of the CSO also impair the caching scheme for Johnson's algorithm we mentioned in Section 3. However, things are not as bad as they seem. A translation of the CSO does not affect the values of the cached vector differences $\mathbf{y}_k - \mathbf{y}_j$, since both $\mathbf{y}_k$ and $\mathbf{y}_j$ are translated over the same vector. The cached values of the determinants Δ_i^X *are* affected by a translation of the CSO. So, each time $\mathbf{x}$ is updated, we must recompute the determinants.

For collision detection, the performance of GJK can be boosted by allowing an early out as soon as $\mathbf{v}_k$ becomes a separating axis [10]. Frame coherence can be exploited, by caching this axis and using it as initial $\mathbf{v}_0$ in the next frame. A similar scheme can be used for the GJK ray cast. In case of a miss, $\mathbf{v}_k$ is a separating axis of the ray R and the object C. If the configuration of R and C does not change a lot over time, then this separating axis is likely to be a separating axis in the next frame as well. By initializing $\mathbf{v}_0$ with a previously returned separating axis, the number of iterations per frame can be reduced considerably. Note that since this $\mathbf{v}_0$ may no longer be contained in $\mathbf{x}_0 - C$, it is necessary to skip the termination test for the first iteration. Overall, the GJK ray cast implemented using this caching scheme is only slightly more expensive than the incremental separating axis variant of GJK.

5 Conclusion

We have presented an iterative method for performing ray casts against general convex objects, and have succeeded in attaining our main objective: to find an algorithm for performing a simplified form of continuous collision detection on convex objects in real-time. Moreover, due to its high performance and versatility, we expect this algorithm to have applications in other areas as well. An obvious application area is ray tracing. The use of support mappings for reading geometry offers new methods for representing shapes in ray-traced environments, and removes the need for representing compound convex shapes with polygonal surfaces.

Acknowledgments

Thanks Willem de Boer for being an excellent sounding board and proof reader.

A Theorems

Lemma 1. *Let C be a convex object, and $\mathbf{x}_1$ and $\mathbf{x}_2$ arbitrary points. Furthermore, let $\mathbf{c}_1$ and $\mathbf{c}_2$ be the points of C closest to respectively $\mathbf{x}_1$ and $\mathbf{x}_2$. Then,*

$$(\mathbf{x}_2 - \mathbf{x}_1) \cdot (\mathbf{c}_2 - \mathbf{c}_1) \geq \|\mathbf{c}_2 - \mathbf{c}_1\|^2.$$

Proof. We derive

$$\begin{aligned} & (\mathbf{x}_2 - \mathbf{x}_1) \cdot (\mathbf{c}_2 - \mathbf{c}_1) \\ = \; & (\mathbf{x}_2 - \mathbf{c}_2 + \mathbf{c}_2 - \mathbf{c}_1 + \mathbf{c}_1 - \mathbf{x}_1) \cdot (\mathbf{c}_2 - \mathbf{c}_1) \\ = \; & (\mathbf{x}_2 - \mathbf{c}_2) \cdot (\mathbf{c}_2 - \mathbf{c}_1) + \|\mathbf{c}_2 - \mathbf{c}_1\|^2 + (\mathbf{x}_1 - \mathbf{c}_1) \cdot (\mathbf{c}_1 - \mathbf{c}_2). \end{aligned}$$

Since C is convex and $\mathbf{x}_1 - \mathbf{c}_1$ is either a normal at $\mathbf{c}_1$ or equal to $\mathbf{0}$, we know that $(\mathbf{x}_1 - \mathbf{c}_1) \cdot (\mathbf{c}_1 - \mathbf{y}) \geq 0$ for any $\mathbf{y} \in C$. In particular, $(\mathbf{x}_1 - \mathbf{c}_1) \cdot (\mathbf{c}_1 - \mathbf{c}_2) \geq 0$. In the same way, we deduce that $(\mathbf{x}_2 - \mathbf{c}_2) \cdot (\mathbf{c}_2 - \mathbf{c}_1) \geq 0$, and thus, that

$$(\mathbf{x}_2 - \mathbf{x}_1) \cdot (\mathbf{c}_2 - \mathbf{c}_1) \geq \|\mathbf{c}_2 - \mathbf{c}_1\|^2.$$

□

Theorem 2. *Let C be a convex object, and $R = \{\mathbf{s} + \lambda\mathbf{r} : \lambda \geq 0\}$ a ray. Then, the mapping from λ to the point of C closest to $\mathbf{x}(\lambda) = \mathbf{s} + \lambda\mathbf{r}$ is continuous.*

Proof. We proof continuity of the closest point mapping in usual way by showing that for each $\varepsilon > 0$, we can find a $\delta > 0$ such that for all λ_1, λ_2 for which $|\lambda_2 - \lambda_1| < \delta$, the distance between the respective closest points of $\mathbf{x}(\lambda_1)$ and $\mathbf{x}(\lambda_2)$ is less than ε.

Let $\varepsilon > 0$, $\lambda_1, \lambda_2 \geq 0$, and $\mathbf{c}_1$ and $\mathbf{c}_2$ the points of C closest to respectively $\mathbf{x}_1 = \mathbf{x}(\lambda_1)$ and $\mathbf{x}_2 = \mathbf{x}(\lambda_2)$. From Lemma 1, we know that

$$(\mathbf{x}_2 - \mathbf{x}_1) \cdot (\mathbf{c}_2 - \mathbf{c}_1) \geq \|\mathbf{c}_2 - \mathbf{c}_1\|^2.$$

We substitute $\mathbf{x}_1 = \mathbf{s} + \lambda_1\mathbf{r}$ and $\mathbf{x}_2 = \mathbf{s} + \lambda_2\mathbf{r}$ and find

$$(\lambda_2 - \lambda_1)\mathbf{r} \cdot (\mathbf{c}_2 - \mathbf{c}_1) \geq \|\mathbf{c}_2 - \mathbf{c}_1\|^2.$$

Observe that $\mathbf{r} \cdot (\mathbf{c}_2 - \mathbf{c}_1)$ is zero only if $\mathbf{c}_1 = \mathbf{c}_2$, so this case can not result in a discontinuity. Assume $\mathbf{r} \cdot (\mathbf{c}_2 - \mathbf{c}_1) \neq 0$. Since the right-hand side of the inequality is non-negative, we can rewrite the equation above as as

$$|\lambda_2 - \lambda_1| \geq \frac{\|\mathbf{c}_2 - \mathbf{c}_1\|^2}{|\mathbf{r} \cdot (\mathbf{c}_2 - \mathbf{c}_1)|}.$$

Then, we see that for

$$|\lambda_2 - \lambda_1| < \frac{\varepsilon^2}{|\mathbf{r} \cdot (\mathbf{c}_2 - \mathbf{c}_1)|}$$

the distance between $\mathbf{c}_1$ and $\mathbf{c}_2$ must be less than ε. □

Theorem 3. *Let C be a convex object, and let $\mathbf{x}_1 = \mathbf{s} + \lambda_1 \mathbf{r}$ and $\mathbf{x}_2 = \mathbf{s} + \lambda_2 \mathbf{r}$ for $0 \leq \lambda_1 < \lambda_2$. Furthermore, let $\mathbf{c}_1$ and $\mathbf{c}_2$ be the points of C closest to respectively $\mathbf{x}_1$ and $\mathbf{x}_2$. Then,*

$$(\mathbf{x}_1 - \mathbf{c}_1) \cdot \mathbf{r} \leq (\mathbf{x}_2 - \mathbf{c}_2) \cdot \mathbf{r}.$$

Proof. We derive

$$\begin{aligned} & (\mathbf{x}_2 - \mathbf{x}_1) \cdot \mathbf{r} \\ = \; & (\mathbf{x}_2 - \mathbf{c}_2 + \mathbf{c}_2 - \mathbf{c}_1 + \mathbf{c}_1 - \mathbf{x}_1) \cdot \mathbf{r} \\ = \; & (\mathbf{x}_2 - \mathbf{c}_2) \cdot \mathbf{r} + (\mathbf{c}_2 - \mathbf{c}_1) \cdot \mathbf{r} - (\mathbf{x}_1 - \mathbf{c}_1) \cdot \mathbf{r}. \end{aligned}$$

Thus, if we can show that $(\mathbf{x}_2 - \mathbf{x}_1) \cdot \mathbf{r} - (\mathbf{c}_2 - \mathbf{c}_1) \cdot \mathbf{r} \geq 0$, then $(\mathbf{x}_1 - \mathbf{c}_1) \cdot \mathbf{r}$ cannot be greater than $(\mathbf{x}_2 - \mathbf{c}_2) \cdot \mathbf{r}$, and we are done. First of all, since $\lambda_1 < \lambda_2$ we see that

$$(\mathbf{x}_2 - \mathbf{x}_1) \cdot \mathbf{r} = (\lambda_2 - \lambda_1)\|\mathbf{r}\|^2 = \|\mathbf{x}_2 - \mathbf{x}_1\|\|\mathbf{r}\|.$$

According to Cauchy's inequality we have

$$(\mathbf{c}_2 - \mathbf{c}_1) \cdot \mathbf{r} \leq \|\mathbf{c}_2 - \mathbf{c}_1\|\|\mathbf{r}\|.$$

Thus,

$$(\mathbf{x}_2 - \mathbf{x}_1) \cdot \mathbf{r} - (\mathbf{c}_2 - \mathbf{c}_1) \cdot \mathbf{r} \geq \|\mathbf{x}_2 - \mathbf{x}_1\|\|\mathbf{r}\| - \|\mathbf{c}_2 - \mathbf{c}_1\|\|\mathbf{r}\|.$$

Remains to be shown that $\|\mathbf{x}_2 - \mathbf{x}_1\| \geq \|\mathbf{c}_2 - \mathbf{c}_1\|$. It follows from Lemma 1 that

$$(\mathbf{x}_2 - \mathbf{x}_1) \cdot (\mathbf{c}_2 - \mathbf{c}_1) \geq \|\mathbf{c}_2 - \mathbf{c}_1\|^2.$$

From Cauchy's inequality we know that $(\mathbf{x}_2-\mathbf{x}_1)\cdot(\mathbf{c}_2-\mathbf{c}_1) \leq \|\mathbf{x}_2-\mathbf{x}_1\|\|\mathbf{c}_2-\mathbf{c}_1\|$, thus

$$\|\mathbf{x}_2 - \mathbf{x}_1\|\|\mathbf{c}_2 - \mathbf{c}_1\| \geq \|\mathbf{c}_2 - \mathbf{c}_1\|^2.$$

This can be simplified to

$$\|\mathbf{x}_2 - \mathbf{x}_1\| \geq \|\mathbf{c}_2 - \mathbf{c}_1\|.$$

□

References

[1] S. Cameron. A study of the clash detection problem in robotics. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 488–493, 1985.

[2] S. Cameron. Collision detection by four-dimensional intersection testing. *IEEE Transactions on Robotics and Automation*, 6(3):291–302, 1990.

[3] J. Canny. Collision detection for moving polyhedra. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(2):200–209, 1986.

[4] J. Eckstein and E. Schömer. Dynamic collision detection in virtual reality applications. In *Proc. 7th International Conference in Central Europe on Computer Graphics and Visualization and Interactive Digital Media, WSCG '99*, pages 71–78, 1999.

[5] E. G. Gilbert and C.-P. Foo. Computing the distance between general convex objects in three-dimensional space. *IEEE Transactions on Robotics and Automation*, 6(1):53–61, 1990.

[6] 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 Journal of Robotics and Automation*, 4(2):193–203, 1988.

[7] P. M. Hubbard. Space-time bounds for collision detection. Technical Report CS-93-04, Dept. of Computer Science, Brown University, Feb. 1993.

[8] S. Redon, A. Kheddar, and S. Coquillart. Fast continuous collision detection between rigid bodies. In *Proc. EUROGRAPHICS 2002*, 2002.

[9] E. Schömer and C. Thiel. Efficient collision detection for moving polyhedra. In *Proc. 11th Annual ACM Symposium on Computational Geometry*, pages 51–60, 1995.

[10] G. van den Bergen. A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools*, 4(2):7–25, 1999.

[11] G. van den Bergen. *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann, San Francisco, CA, 2003.