The Gilbert-Johnson-Keerthi algorithm

Christer Ericson
Sony Computer Entertainment America

1 Introduction

One of the most effective methods for determining intersection between two polyhedra is an iterative algorithm due to Gilbert, Johnson, and Keerthi, commonly referred to as the *GJK algorithm* [Gilbert88]. As originally described, GJK is a simplex-based descent algorithm that, given two sets of vertices as inputs, finds the Euclidean distance (and closest points) between the convex hulls of these sets. In a generalized form, the GJK algorithm can also be applied to arbitrary convex point sets, not just polyhedra [Gilbert90]. While the examples presented here are in terms of polytopes, the described algorithm is the generalized version and it applies to non-polygonal convex sets in a straightforward way. For more detail on the GJK algorithm than presented here, please consult [Ericson] or [Bergen03].

2 Preliminaries

While the original presentation of the GJK algorithm is quite technical, neither the algorithm itself nor its implementation are very complicated in practice. However, understanding of the algorithm does require the introduction of some concepts from convex analysis and the theory of convex sets on which the algorithm relies.

An important point is that the GJK algorithm does not actually operate on the two input objects per se, but on the *Minkowski difference* between the objects. This transformation of the problem reduces the problem from finding the distance between two convex sets to that of finding the distance between the origin and a single convex set. The GJK algorithm searches the Minkowski difference object iteratively, a subvolume at a time, each such volume being a *simplex*. The Minkowski difference is not explicitly computed but sampled through a *support mapping* function on demand. The concepts of Minkowski

difference, simplices, and support mapping are described in more detail in the following three sections.

2.1 Minkowski sums and differences

Two operations on convex sets important for the understanding of the GJK algorithm are the *Minkowski sum* and the *Minkowski difference* of point sets. Let A and B be two point sets and let $\mathbf{a}$ and $\mathbf{b}$ be the position vectors corresponding to pairs of points in A and B. The Minkowski sum, $A \oplus B$ is then defined as the set:

$$A \oplus B = \{\mathbf{a} + \mathbf{b} : \mathbf{a} \in A, \mathbf{b} \in B\}$$

where $\mathbf{a} + \mathbf{b}$ is the vector sum of the position vectors $\mathbf{a}$ and $\mathbf{b}$. Visually the Minkowski sum can be seen as the region swept by A translated to every point in B (or vice versa). An illustration of the Minkowski sum is given in Figure 1.

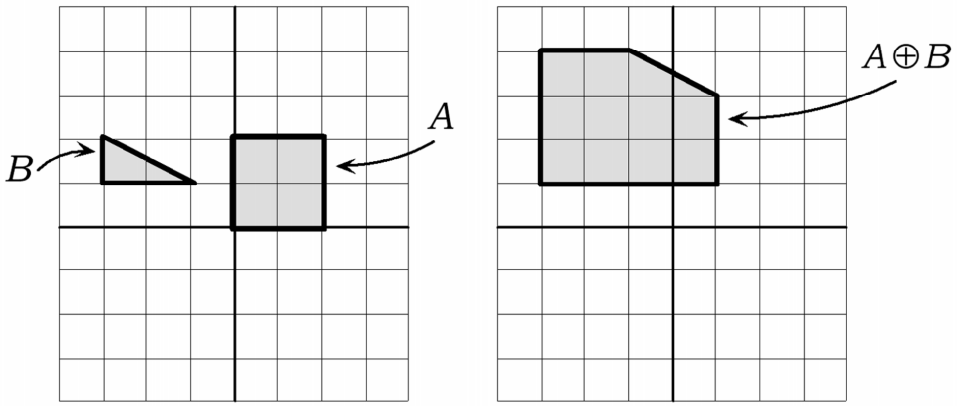

Figure 1: The Minkowski sum of a square A and a triangle B.

The Minkowski difference of two point sets A and B is defined analogously to the Minkowski sum:

$$A \ominus B = \{\mathbf{a} - \mathbf{b} : \mathbf{a} \in A, \mathbf{b} \in B\}$$

Geometrically, the Minkowski difference is obtained by adding A to the reflection of B about the origin, that is, $A \ominus B = A \oplus (-B)$ (Figure 2). For this reason, both terms are often simply referred to as the Minkowski sum. For two convex polygons, P and Q, the Minkowski sum $R = P \oplus Q$ has the properties that R is a convex polygon and the vertices of R are sums of vertices of P and Q. The Minkowski sum of two convex polyhedra is a convex polyhedron, with corresponding properties.

The Minkowski difference is important from a collision detection perspective as two point sets A and B collide (that is, have one or more points in common)

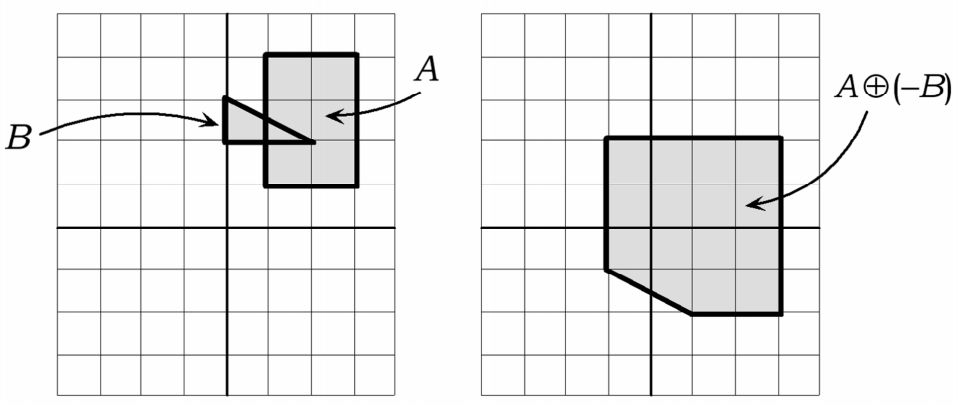

Figure 2: Since rectangle A and triangle B intersect, the origin must be contained in their Minkowski difference.

if and only if their Minkowski difference C, $C = A \ominus B$, contains the origin (c.f. Figure 2). In fact, it is possible to establish an even stronger result: computing the minimum distance between A and B is equivalent to computing the minimum distance between C and the origin. This follows since:

$$\begin{aligned} distance(A, B) &= \min\{\|\mathbf{a} - \mathbf{b}\| : \mathbf{a} \in A, \mathbf{b} \in B\} \\ &= \min\{\|\mathbf{c}\| : \mathbf{c} \in A \ominus B\} \end{aligned}$$

Note that the Minkowski difference of two convex sets is also a convex set, so its point of minimum norm is unique.

2.2 Simplices

A *d-simplex* is the convex hull of $d+1$ affinely independent points in d-dimensional space. A *simplex* (plural *simplices*) is a d-simplex, for some given d. For example, the 0-simplex is a point, the 1-simplex is a line segment, the 2-simplex is a triangle and the 3-simplex is a tetrahedron (Figure 3). A simplex has the property that removing a point from its defining set reduces the dimensionality of the simplex by one.

2.3 Supporting points and support mappings

For a general convex set C (thus not necessarily a polytope) a point from the set most distant along a given direction is called a *supporting point* of C. More specifically, P is a supporting point of C if, for a given direction $\mathbf{d}$ it holds that $\mathbf{d} \cdot P = \max\{\mathbf{d} \cdot V : V \in C\}$; that is, P is a point for which $\mathbf{d} \cdot P$ is maximal. Figure 4 illustrates the supporting points for two different convex

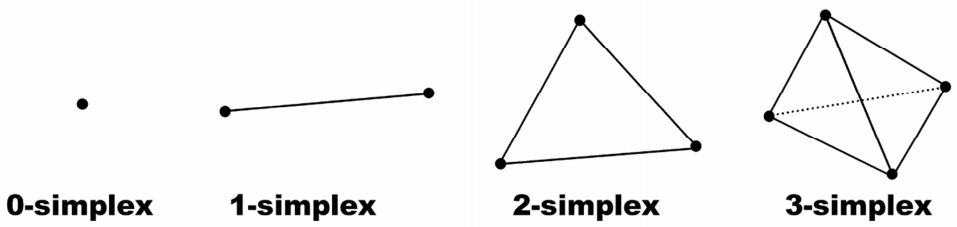

Figure 3: Simplices of dimension 0 through 3: a point, a line segment, a triangle, and a tetrahedron.

sets. Supporting points are sometimes called *extreme points*. They are not necessarily unique. For a polytope, one of its vertices can always be selected as a supporting point for a given direction. When a supporting point is a vertex, the point is commonly called a *supporting vertex*.

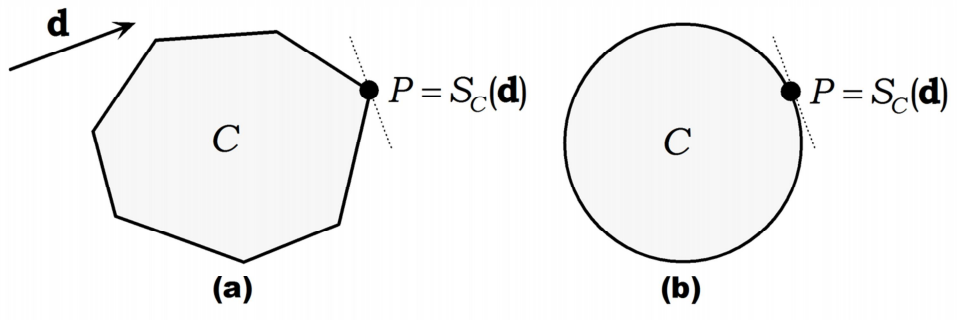

Figure 4: (a) A supporting vertex P of polygon C with respect to the direction $\mathbf{d}$. (b) A supporting point P of circle C with respect to the direction $\mathbf{d}$. In both cases, P is given by the support mapping function $S_C(\mathbf{d})$.

A *support mapping* is a function, $S_C(\mathbf{d})$, associated with a convex set C that maps the direction $\mathbf{d}$ into a supporting point of C. For simple convex shapes, such as spheres, boxes, cones, and cylinders, support mappings can be given in closed form. For example, for a sphere C centered at O and with a radius of r, the support mapping is given by $S_C(\mathbf{d}) = O + r\,\mathbf{d}/\,\|\mathbf{d}\|$ (c.f. Figure 4(b)). Convex shapes of higher complexity require the support mapping function to determine a supporting point using numerical methods.

For a polytope of n vertices, a supporting vertex is trivially found in $O(n)$ time by searching over all vertices. Assuming a data structure listing all adjacent vertex neighbors for each vertex, an extreme vertex can be found through a simple hill-climbing algorithm, greedily visiting more and more extreme vertices until no vertex more extreme can be found. This approach is very efficient as it only explores a small corridor of vertices as it moves towards the extreme vertex. For larger polyhedra, the hill-climbing can be sped up by adding one or more

artificial neighbors to the adjacency list for a vertex. Through precomputation of a hierarchical representation of the vertices, it is possible to locate a supporting point in $O(\log n)$ time. These acceleration schemes are described in more detail in [Ericson].

3 The Gilbert-Johnson-Keerthi algorithm

As mentioned earlier, the GJK algorithm effectively determines intersection between polyhedra by computing the Euclidean distance between them. The algorithm is based on the fact that the separation distance between two polyhedra A and B is equivalent to the shortest distance between their Minkowski difference C, $C = A \ominus B$, and the origin (Figure 5). Thus, the problem is reduced to that of finding the point on C closest to the origin. At the outset, this does not seem like much of an improvement as the Minkowski difference is non-trivial to compute explicitly. However, a key point of the GJK algorithm is that it does not explicitly compute the Minkowski difference C. It only samples the Minkowski difference point set using a support mapping of $C = A \ominus B$. Since the support mapping function is the maximum over a linear function, the support mapping for C, $s_{A\ominus B}(\mathbf{d})$, can be expressed in terms of the support mappings for A and B as $s_{A\ominus B}(\mathbf{d}) = s_A(\mathbf{d}) - s_B(-\mathbf{d})$. Thus, points from the Minkowski difference can be computed, on demand, from supporting points of the individual polyhedra A and B.

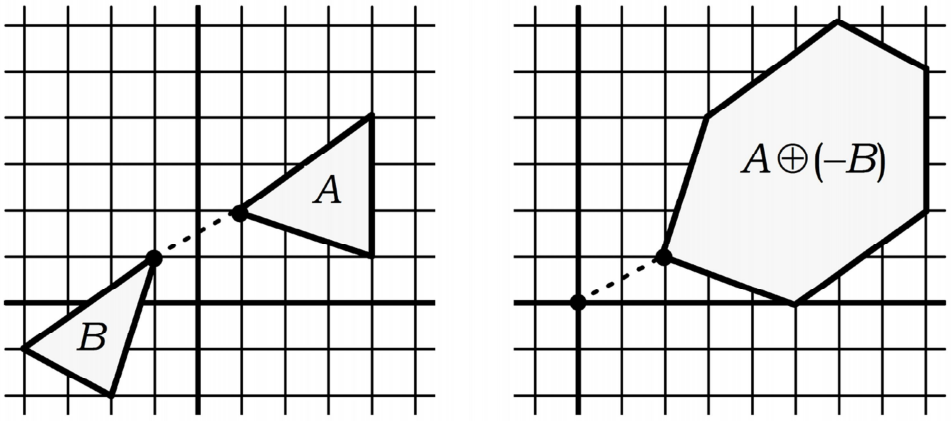

Figure 5: The distance between A and B is equivalent to the distance between their Minkowski difference and the origin.

To search the Minkowski difference C for the point closest the origin, the GJK algorithm utilizes a result known as *Carathéodory's theorem* [Rockafellar96]. The theorem says that for a convex body H of $\mathbf{R}^d$, each point of H can be expressed as the convex combination of no more than $d+1$ points from H. This allows the GJK algorithm to search the volume of the Minkowski difference C

by maintaining a set Q of up to $d+1$ points from C at each iteration. The convex hull of Q forms a simplex inside C. If the origin is contained in the current simplex, A and B are intersecting, and the algorithm stops. Otherwise, the set Q is updated to form a new simplex, guaranteed to contain points closer to the origin than the current simplex. Eventually this process must terminate with a Q containing the closest point to the origin. In the non-intersecting case, the smallest distance between A and B is realized by the point of minimum norm in $CH(Q)$ (the convex hull of Q). The following step-by-step description of the GJK algorithm specifies in more detail how the set Q is updated:

1. Initialize the simplex set Q to one or more points (up to $d+1$ points, where d is the dimension) from the Minkowski difference of A and B.
2. Compute the point P of minimum norm in $CH(Q)$.
3. If P is the origin itself, the origin is clearly contained in the Minkowski difference of A and B. Stop and return A and B as intersecting.
4. Reduce Q to the smallest subset Q' of Q such that $P \in CH(Q')$. That is, remove any points from Q not determining the subsimplex of Q in which P lies.
5. Let $V = s_{A\ominus B}(-P) = s_A(-P) - s_B(P)$ be a supporting point in direction $-P$.
6. If V is no more extremal in direction $-P$ than P itself, stop and return A and B as not intersecting. The length of the vector from the origin to P is the separation distance of A and B.
7. Add V to Q and go to 2.

Figure 6 illustrates how GJK iteratively finds the point on a polygon closest to the origin O for a two-dimensional problem. The algorithm arbitrarily starts with the vertex A as the initial simplex set Q, $Q = \{A\}$. For a single-vertex simplex, the vertex itself is the closest point to the origin. Searching for the supporting vertex in direction $-A$ results in B. B is added to the simplex set, giving $Q = \{A, B\}$. The point on $CH(Q)$ closest to the origin is C. Since both A and B are needed to express C as a convex combination, both are kept in Q. D is the supporting vertex in direction $-C$ and it is added to Q, giving $Q = \{A, B, D\}$. The closest point on $CH(Q)$ to the origin is now E. Since only B and D are needed to express E as a convex combination of vertices in Q, Q is updated to $Q = \{B, D\}$. The supporting vertex in direction $-E$ is F, which is added to Q. The point on $CH(Q)$ closest to the origin is now G. D and F is the smallest set of vertices in Q needed to express G as a convex combination, so Q is updated to $Q = \{D, F\}$. At this point, since no vertex is closer to the origin in direction $-G$ than G itself, G must be the closest point to the origin, and the algorithm terminates.

Note that the GJK algorithm trivially deals with spherically extended polyhedra, simply by comparing the computed distance between the inner polyhedral

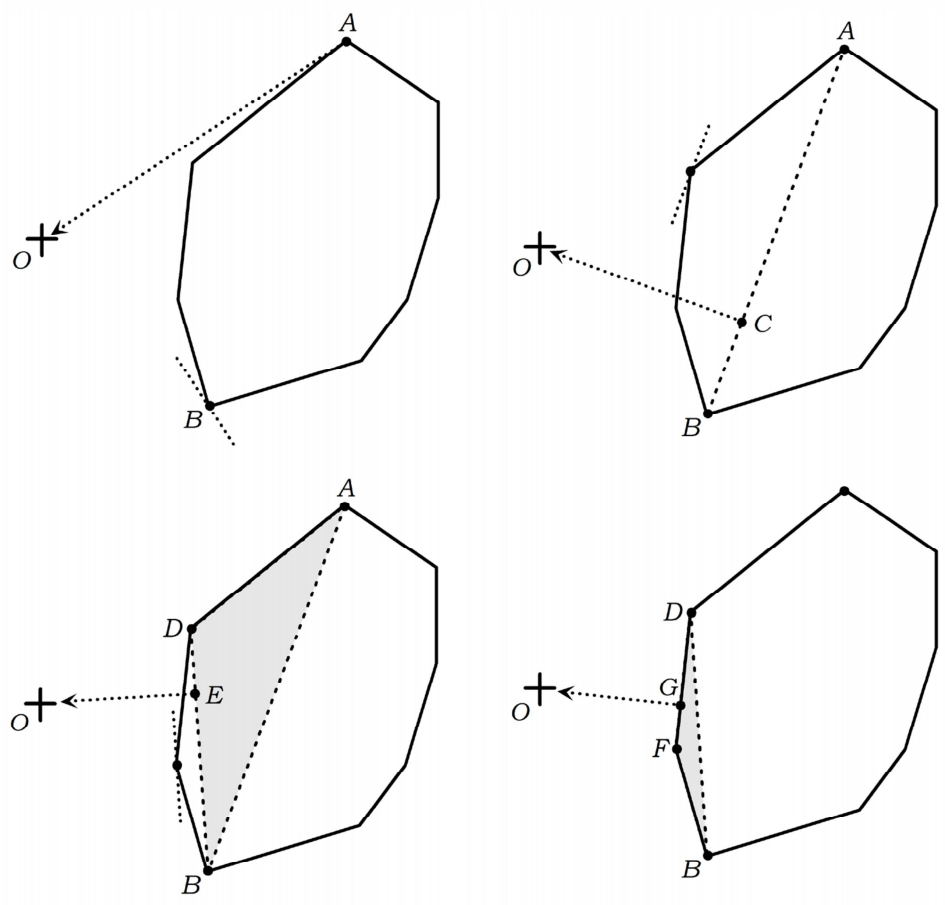

Figure 6: GJK finding the point on a polygon closest to the origin.

structures with the sum of the radii of the spherical extensions. Note also that if the GJK algorithm is applied to the vertex sets of nonconvex polyhedra, it will compute the smallest distance between the convex hulls of these nonconvex polyhedra.

While presented here as a method operating on polyhedra only, the GJK algorithm can, in fact, be applied to arbitrary convex bodies. Since the input bodies are only ever sampled through their support mappings, all that is required to allow the GJK algorithm to work with general convex objects is to supply appropriate support mappings.

The GJK algorithm terminates with the separation distance in a finite number of steps for polyhedra. However, it only asymptotically converges to the separation distance for arbitrary convex bodies. Therefore, a suitable tolerance should be added to the termination condition to allow the algorithm to terminate correctly when operating on non-polyhedral objects.

On termination with non-intersection, in addition to returning the separation distance, it is possible to have GJK compute the closest points between the input objects, computed from the points of objects A and B that formed the points in the last copy of simplex set Q. The steps of this computation are beyond the scope of these notes, see [Ericson] for full details.

3.1 Finding the point of minimum norm in a simplex

It remains to describe how to determine the point P of minimum norm in $CH(Q)$ for a simplex set $Q = \{Q_1, Q_2, \ldots, Q_k\}$, $1 \leq k \leq 4$. In the original presentation of GJK, this is done through a single procedure called the distance subalgorithm. The distance subalgorithm reduces the problem to considering all subsets of Q separately. For example, for $k = 4$ there are 15 subsets corresponding to 4 vertices (Q_1, Q_2, Q_3, Q_4), 6 edges (Q_1Q_2, Q_1Q_3, Q_1Q_4, Q_2Q_3, Q_2Q_4, Q_3Q_4), 4 faces ($Q_1Q_2Q_3$, $Q_1Q_2Q_4$, $Q_1Q_3Q_4$, $Q_2Q_3Q_4$), and the interior of the simplex ($Q_1Q_2Q_3Q_4$). The subsets are searched one by one, in order of increasing size. The search stops when the origin is contained in the *Voronoi region* of the feature (vertex, edge, or face) specified by the examined subset (or, for the subset corresponding to the interior of the simplex, when the origin is inside the simplex). Once a feature has been located, the point of minimum norm on this feature is given by the orthogonal projection of the origin onto the feature.

The Voronoi region for a feature F is the region of space containing points that lie closer to (or as close to) F than to any other feature. Figure 7 illustrates the Voronoi regions determined by the features of a triangle.

Consider again the case of $Q = \{Q_1, Q_2, Q_3, Q_4\}$. An arbitrary point P (specifically the origin) lies in the Voronoi region for, say, vertex Q_1 if and only if the following inequalities are satisfied:

$$\begin{aligned}(P - Q_1) \cdot (Q_2 - Q_1) &\leq 0 \\ (P - Q_1) \cdot (Q_3 - Q_1) &\leq 0 \\ (P - Q_1) \cdot (Q_4 - Q_1) &\leq 0\end{aligned}$$

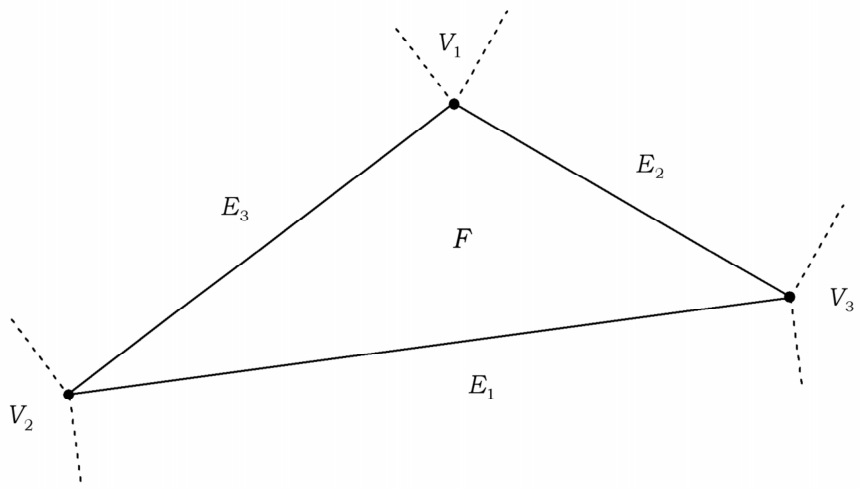

Figure 7: A triangle divides its supporting plane into 7 Voronoi feature regions: 1 face region (F), 3 edge regions (E_1, E_2, E_3), and 3 vertex regions (V_1, V_2, V_3).

P lies in the Voronoi region associated with edge Q_1Q_2 if and only if the following inequalities are satisfied:

$$\begin{gathered}(P - Q_1) \cdot (Q_2 - Q_1) \geq 0 \\ (P - Q_2) \cdot (Q_1 - Q_2) \geq 0 \\ (P - Q_1) \cdot ((Q_2 - Q_1) \times \mathbf{n}_{123}) \geq 0 \\ (P - Q_1) \cdot (\mathbf{n}_{142} \times (Q_2 - Q_1)) \geq 0\end{gathered}$$

where:

$$\begin{aligned}\mathbf{n}_{123} &= (Q_2 - Q_1) \times (Q_3 - Q_1) \\ \mathbf{n}_{142} &= (Q_4 - Q_1) \times (Q_2 - Q_1)\end{aligned}$$

If P does not lie in a vertex or edge Voronoi region, face regions are tested by checking if P and the remaining point from Q set lie on opposite sides of the plane through the three chosen points from Q to form the face. For example, P lies in the Voronoi region of $Q_1Q_2Q_3$ if and only if the following inequality holds:

$$((P - Q_1) \cdot \mathbf{n}_{123})((Q_4 - Q_1) \cdot \mathbf{n}_{123}) < 0$$

where again:

$$\mathbf{n}_{123} = (Q_2 - Q_1) \times (Q_3 - Q_1)$$

Analogous sets of inequalities can be defined for testing containment in the remaining vertex, edge, and face Voronoi regions. Note that most of the computed quantities are shared between different Voronoi region tests and need not be recomputed, resulting in an efficient test overall. Simplex sets of fewer than four points are handled in a corresponding way.

4 GJK for moving objects

While the GJK algorithm is usually presented and thought of as operating on two convex polyhedra, it is more general than so. Given two point sets, it computes the minimum distance vector between the convex hulls of the point sets (an easy way of seeing this is to note that the support mapping function never returns a point interior to the hull). This is an important distinction as it allows GJK to be used to determine collisions between convex objects under linear translational motion in a straightforward manner.

One approach to dealing with moving polyhedra is presented in [Xavier97]. Consider two polyhedra P and Q, with movements given by the vectors $\mathbf{t}_1$ and $\mathbf{t}_2$, respectively. To simplify the collision test, the problem is recast so Q is stationary. The relative movement of P (with respect to Q) is now given by $\mathbf{t} = \mathbf{t}_1 - \mathbf{t}_2$. Let V_i be the vertices of P in its initial position. $V_i + \mathbf{t}$ describes the location of the vertices of P at the end of its translational motion.

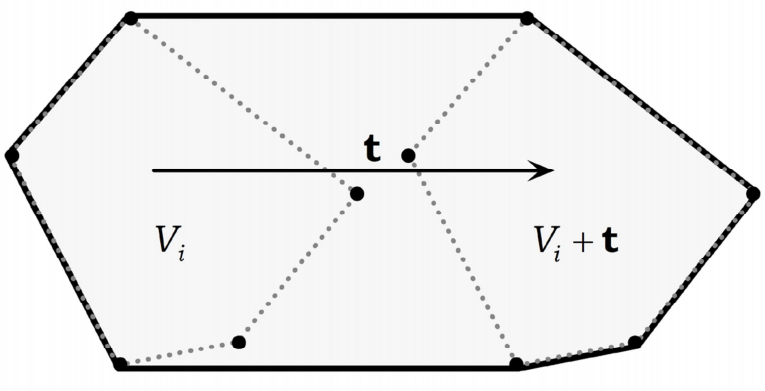

Figure 8: For a convex polyhedron under a translational movement $\mathbf{t}$, the convex hull of the vertices V_i at the start and the vertices $V_i + \mathbf{t}$ at the end of motion corresponds to the swept hull for the polyhedron.

It is not hard to see that as P moves from start to end over its range of motion, the volume swept out by P corresponds to the convex hull of its vertices at their initial and final positions (Figure 8). Determining if P collides with Q during its translational motion is therefore as simple as passing GJK the vertices of P at both start and end of P's motion (since this convex hull is what GJK effectively computes, given these two point sets). A drawback with this solution is that doubling the number of vertices for P increases the time to find a supporting vertex. A better approach, which does not suffer from this problem, is to consider the movement vector $\mathbf{t}$ of P with respect to $\mathbf{d}$, the vector for which an extreme vertex is sought. From the definition of the extreme vertex it is easy to see that when $\mathbf{t}$ is pointing away from $\mathbf{d}$, none of the vertices $V_i + \mathbf{t}$ can be more extreme than the vertices V_i. Thus, when $\mathbf{d} \cdot \mathbf{t} \leq 0$, an extreme vertex is guaranteed to be found amongst the vertices V_i (Figure 9(a)). Similarly, when

$\mathbf{d} \cdot \mathbf{t} > 0$, only the vertices $V_i + \mathbf{t}$ need to be considered for locating an extreme vertex (Figure 9(b)).

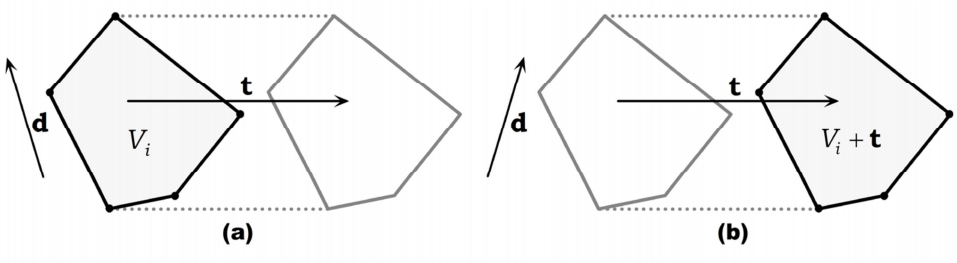

Figure 9: (a) When $\mathbf{d} \cdot \mathbf{t} \leq 0$, the supporting vertex is found amongst the original vertices V_i and the vertices $V_i + \mathbf{t}$ do not have to be tested. (b) Similarly, when $\mathbf{d} \cdot \mathbf{t} > 0$, only the vertices $V_i + \mathbf{t}$ have to be considered.

The second of two presented approaches is effectively implemented by changing the support mapping function such that the motion vector is added to the vertices during hill-climbing.

A drawback is that both presented methods only provide interference detection. However, interval halving can be effectively used to get the time of collision. Using an interval halving approach, the simplices from the previous iteration are ideal candidates for starting the next iteration. Alternatively, the time of collision can be obtained iteratively using a root finder such as Brent's method, as described in [Vlack01].

References

[Bergen03] van den Bergen, Gino. *Collision detection in interactive 3d environments.* Morgan Kaufmann Publishers, 2003.

[Ericson] Ericson, Christer. *Real-Time collision detection.* Morgan Kaufmann Publishers. Forthcoming.

[Gilbert88] Gilbert, Elmer. Daniel Johnson, S. Sathiya Keerthi. "A fast procedure for computing the distance between complex objects in three dimensional space." *IEEE Journal of Robotics and Automation*, vol.4, no. 2, pp. 193-203, 1988.

[Gilbert90] Gilbert, Elmer. Chek-Peng Foo. "Computing the Distance Between General Convex Objects in Three-Dimensional Space." *IEEE Transactions on Robotics and Automation*, vol. 6, no. 1, pp. 53-61, 1990.

[Rockafellar96] Rockafellar, R. Tyrell. *Convex Analysis.* Princeton University Press, 1996.

[Vlack01] Vlack, Kevin. Susumu Tachi. "Fast and accurate spacio-temporal intersection detection with the GJK algorithm." *Proceedings of the International Conference on Artificial Reality and Telexistence (ICAT2001)*, pp. 79-84, 2001. `http://vrsj.t.u-tokyo.ac.jp/ic-at/papers/01079.pdf`

[Xavier97] Xavier, Patrick. "Fast swept-volume distance for robust collision detection." *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, Albuquerque, NM, April 1997.