# The Gilbert-Johnson-Keerthi Distance Algorithm

Patrick Lindemann

**Abstract**— This paper gives an overview of the Gilbert-Johnson-Keerthi (GJK) algorithm, which provides an iterative method for computing the euclidian distance between two convex sets in m-dimensional space with linear time complexity. The algorithm is very versatile and several enhancements have been published since it was first introduced. Apart from some historical information, this paper will provide the mathematical basics required to understand GJK, mainly support mappings and the Minkowski Difference of geometrical objects. The algorithm will be explained and examples of enhancements and applications are given.

**Index Terms**—GJK, Proximity Queries, Distance Computation, Collision Detection, Convex Hull, Minkowski Difference, Support Mapping

## 1 Introduction

This research paper shall provide an insight into a classic algorithm for distance computations: the Gilbert-Johnson-Keerthi distance algorithm or simply GJK algorithm. This particular distance algorithm computes the euclidian distance between two convex objects and is also able to return the objects' respective points closest to each other. The algorithm provides a linear time complexity, dependent on the number of vertices of which the pair of objects consists. Furthermore, it is not restricted to a specific number of dimensions and can therefore be used in any m-dimensional space. The algorithm's comparably low complexity is a consequence of its mathematical programming origin [5]. GJK uses support mapping functions to describe geometrical objects and reduces the problem of computing the distance between two objects to the simpler task of computing the distance of one object to the origin, which is achieved by using the Minkowski Difference. Both ideas are thoroughly explained later *(see chapter 3)*. Although the algorithm might be hard to grasp in the beginning, it is very popular, since the implementation is much easier [6]. To compute the actual distance, the GJK algorithm approaches the final result by using simplices *(see chapter 3.1)* contained in the distant object. Iteratively, simplices closer to the origin are constructed until the distance to the current simplex equals the distance to the whole object. GJK uses Johnson's so-called Distance Subalgorithm *(see chapter 4.2)* to calculate the distance from the origin to the current simplex [5].

This introduction gave a brief overview over the algorithm. The next section provides some information about the algorithm's history. Section 3 explains some basic mathematical knowledge required to grasp the functionality of GJK, before the actual algorithm will be presented in section 4. Section 5 gives examples for enhancements and applications of the algorithm and section 6 provides the paper's conclusion.

## 2 History

The original GJK algorithm was described by E.G. Gilbert, D.W. Johnson and S.S. Keerthi in 1988 [5] and was at that time restricted to compute the distance between two convex polyhedra, which are geometrical bodies defined by solely flat faces . In 1990, E.G. Gilbert and C.-P. Foo enhanced the original algorithm to handle all kinds of convex objects [4]. Due to the algorithm's age, the part of it called the Distance Subalgorithm *(see chapter 4.2)* is somewhat outdated, since more modern and mathematically intuitive procedures can be used today. As a result of this, when the Gilbert-Johnson-Keerthi distance algorithm is applied, part of it is often replaced by subalgorithms more adapted to the actual problem.


- *Patrick Lindemann is studying Media Informatics at the University of Munich, Germany, E-mail: patrick.lindemann@campus.lmu.de*

- *This research paper was written for the Media Informatics Proseminar on "Algorithms in Media Informatics", 2009*

## 3 Preliminaries

This section explains a number of mainly mathematical issues vital to the understanding of the GJK algorithm.

### 3.1 Simplex

To approach the actual distance to an object, GJK iteratively chooses a simplex *(see figure 1)* within the object and computes the distance to this simplex. A n-simplex is a polytope of n+1 vertices in n-dimensional space. For GJK, the two most interesting cases are naturally two-dimensional and three-dimensional space. Therefore, the algorithm uses a 2-simplex respectively a triangle for two-dimensional space and a 3-simplex respectively a tetrahedron for three-dimensional space [5].

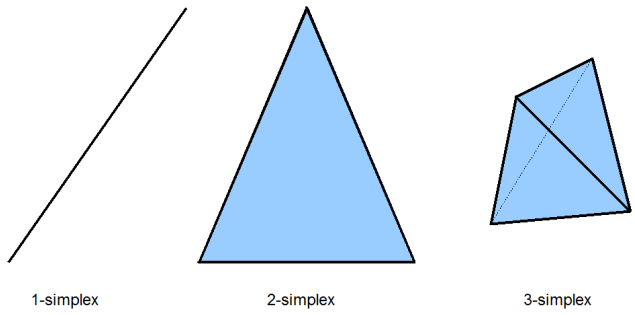


Figure 1. Example simplices for one-, two and three-dimensional space.

### 3.2 Convex Hull

The convex hull *(see figure 2)* of a finite set of points $A$ is the smallest convex set still entirely containing $A$. Thus, the cardinality of a convex hull is always smaller or equal to the cardinality of the original set, for potential inner points need not be considered for the convex hull. GJK works with convex objects only and approaches the distance of an actual object by computing distances to simplices within the object. Therefore, it might be necessary to use the convex hull of the current simplex for calculations, if it is not a convex shape itself [5, 6]. The convex hull of a set $A$ is denoted by $CH(A)$.

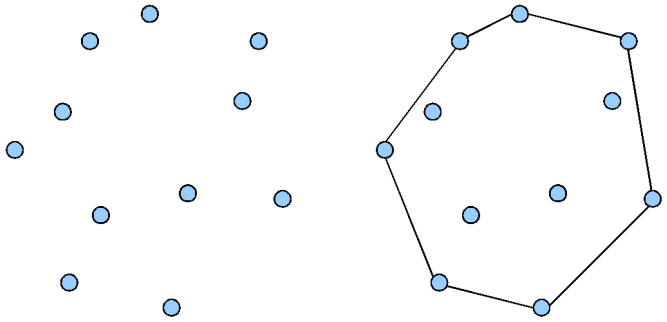

Figure 2. A set of points and its corresponding convex hull.

### 3.3 Support Mapping Functions

The use of so-called support mappings *(see figure 3)* is a major reason for the quickness of the GJK algorithm. Support mappings are an alternative way to completely describe geometrical objects, other than by storing the vertices of an object or an objects hull. A support mapping function $s_A(v)$ of a convex set $A$ maps a vector $v$ to a specific point of the same set, called the "support point". The support point is the one point within $A$, that is the most extreme in direction $v$. Therefore, for the general case of a convex set $A$, the respective support mapping function has to provide a result such that the following is fulfilled [5, 6]:
$v \cdot s_A(v) = \max\{v \cdot x : x \in A\}$.

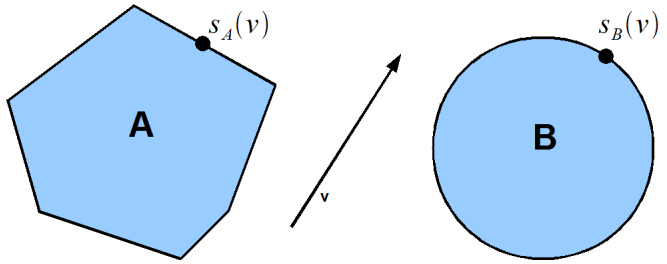


Figure 3. Support points of vector $v$ for a polygon $A$ and a circle $B$.

However, more precise support mapping functions can be given for most primitives, so that it is not required to compute all possible products $v \cdot x$. As an example, support points of circles or spheres can always be calculated according to the formula $c + r \cdot \frac{v}{||v||}$, with c being the point at the center and r being the radius of the respective circle or sphere [6]. Thus, support points for primitives can be calculated very easily, which makes the Gilbert-Johnson-Keerthi distance algorithm all the faster and very versatile. As complex objects can sometimes be comfortably disassembled into several primitives, it could still be possible to gain a time advantage by computing the distances to the primitives separately and taking the nearest as the result.

### 3.4 Minkowski Difference

The Minkowski Sum $C$ *(see figure 4)* of two convex sets $A$ and $B$ is another convex set defined as follows:
$C = A + B = \{x + y : x \in A, y \in B\}$.
In other words, every point of B is added to every point of A and the result of the operation forms the new object $C$. Since the result is a set as well, possible duplicate points evolved from the addition are eliminated. Thus, the cardinality of $C$ is always smaller or equal to the sum of the cardinalities of $A$ and $B$. [5] Although the term "Minkowski Difference" is sometimes used differently in other contexts [6], it will be treated analogically to the Minkowski Sum in the following. Thus, the Minkowski Difference $C$ of two sets $A$ and $B$ follows the definition:
$C = A - B = \{x - y : x \in A, y \in B\}$.
Instead of adding all points of $B$, they are now subtracted from all points of $A$. The Minkowski Difference is the key to the algorithm's idea of reducing the proximity problem of two objects to one object. The distance of two convex sets $A$ and $B$ is equal to the distance of their Minkowski Difference $A - B$ to the origin [5]. This fact is used by the GJK algorithm and the same procedure can be applied to $A - B$ with almost no change to the algorithm. Since GJK uses support mapping functions for representing the geometry of objects, only these have to be adapted to the new convex set. In case of the Minkowski Sum and Minkowski Difference, this can be achieved simply by adding or subtracting the support mappings of its single components in the following way:
$s_{A+B}(v) = s_A(v) + s_B(v)$
$s_{A-B}(v) = s_A(v) - s_B(-v)$
Furthermore, if two objects are intersecting, it can be easily determined since the origin is then contained in the Minkowski Difference of these objects.

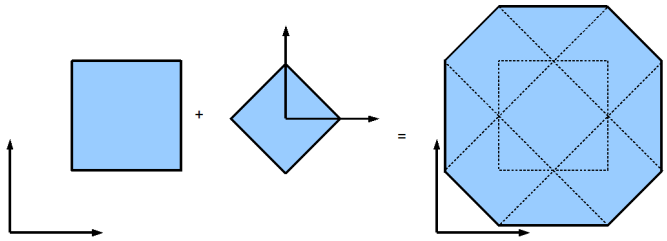

Figure 4. Example for the minkowski sum of two geometrical objects.

## 4 The Algorithm

This section provides an in-depth description of the enhanced Gilbert-Johnson-Keerthi distance algorithm by Gilbert and Foo [4], which can be applied to all convex objects instead of merely convex polytopes, as originally described by Gilbert, Johnson and Keerthi [5]. The algorithm computes the euclidian distance $d(A,B)$ between two convex objects $A$ and $B$, with the euclidian distance being defined by:
$d(A,B) = \min\{||x - y|| : x \in A, y \in B\}$.

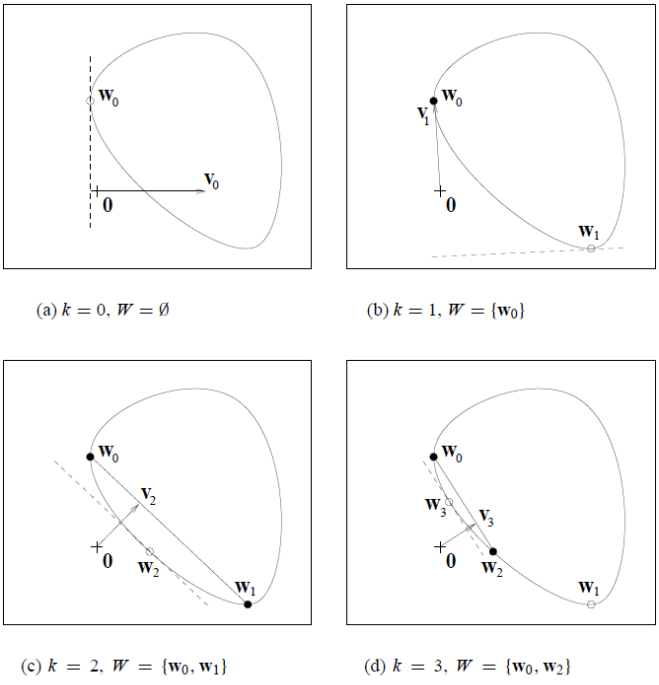


Figure 5. First four iterations of the GJK algorithm [6].

The algorithm can also provide the two points $a \in A$ respectively $b \in B$ closest to each other, meaning that $||a-b|| = d(A,B)$ [6]. Instead of actually computing the distance between objects $A$ and $B$, the GJK algorithm solves the problem in a reduced form, computing the distance between the origin and the Minkowski Difference $C = A - B$ of $A$ and $B$, which is equal to the desired result. Let $v(C) \in C$ be defined as the point in C nearest to the origin, such that $d(A,B) = ||v(A-B)|| = ||v(C)|| = \min\{||x|| : x \in C\}$.

### 4.1 The Main Algorithm

In each iteration, GJK constructs a simplex within $C$ that lies nearer to the origin than the simplex constructed in the previous iteration. In regard to these simplices, we take the following assumptions [6]:

- Let $W_k$ be the set of vertices of the simplex in the $k$-th iteration of the algorithm ($k >= 1$).
- Let $v_k$ be the point of the convex hull of the simplex, which is closest to the origin, i.e. $v_k = v(CH(W_k))$.

In detail, the Gilbert-Johnson-Keerthi distance algorithm proceeds in the following way [6] *(see figure 5)*:

1. Initialization step:
   - Set $k = 0$.
   - Set the simplex set $W_0 = \emptyset$.
   - Let $v_0$ be an arbitrary point within $C$.
2. Compute the support point $w_k = s_c(-v_k)$ in direction $-v_k$.
3. If $w_k$ is no more extreme than $v_k$ in direction $-v_k$: **return** $||v_k||$ .
4. Add $w_k$ to the current simplex set $W_k$.
5. Compute $v_{k+1} = v(CH(W_k \cup \{w_k\}))$, the point closest to the origin within the new simplex.
6. Make $W_{k+1}$ the smallest convex subset of $W_k \cup \{w_k\}$ still containing $v_{k+1}$.
7. Increment $k$, jump to step 2.

### 4.2 The Distance Subalgorithm

There is one part of the algorithm which needs further explanation: Johnson's Distance Subalgorithm [5]. It is responsible for computing the closest point to the origin of the current iteration's simplex *(see step 5)*. Basically, it searches all possible subsets of the current simplex, i.e. dependent on the number of dimensions, each vertice, each edge, each face and so on, including the entire simplex itself. For each of these subsets, a system of linear equations is solved in a recursive procedure. The subalgorithm is somewhat outdated, since it originates from a time at which math operations where expensive. It represents an algebraic, but hardly intuitive approach [7, 2]. The original subalgorithm will not be discussed in detail here, for it would go beyond the scope of this paper. However, an alternative by Christer Ericson [3] for the subalgorithm will be presented in the next section.

## 5 Enhancements and Applications

Basically, the Gilbert-Johnson-Keerthi distance algorithm can easily be enhanced, e.g. by replacing the Distance Subalgorithm by another procedure or by a specific way of constructing the simplices in each iteration.

Stephen Cameron presented an enhancement of GJK in 1997, which promises almost constant time complexity. It uses a method called "hill climbing" when computing the support points in each iteration, i.e. the support point of the last iteration is buffered and in the current iteration, the neighbours of this vertice are checked at first. For larger convex hulls, this approach supposedly brings dramatic improvements in computation time [1]. Furthermore, Cameron's enhancement provides the possibility to compute the penetration distance of two intersecting objects.

An alternative for Johnson's Distance Subalgorithm was presented by Ericson at SIGGRAPH 2004 [2]. It provides a geometrical rather than algebraic approach to the problem, making the subalgorithm more intuitive and easier to make robust, while remaining mathematically equivalent to the original subalgorithm. It uses straightforward functions for finding the closest point on a primitive. As an example, the operating method of a function *ClosestPointOnTriangleToPoint*() will be briefly explained. Based on the triangle of which the distance should be computed, it is determined in which Voronoi region *(see figure 6)* the distant point (usually the origin) lies. Based on the two types of Voronoi regions ($V_A$/$V_B$/$V_C$ or $E_{AB}$/$E_{AC}$/$E_{BC}$), a case differentiation is made and equations are solved, dependent on the case:

1. Distant point X lies within a vertice ($V$) region. Example equations for region $V_A$:
   - $AX \cdot AB \leq 0$
   - $AX \cdot AC \leq 0$
2. Distant point X lies within an edge ($E$) region. Example equations for region $E_{AB}$:
   - $(BC \times BA) \times BA \cdot BX \geq 0$
   - $AX \cdot AB \geq 0$
   - $BX \cdot BA \geq 0$

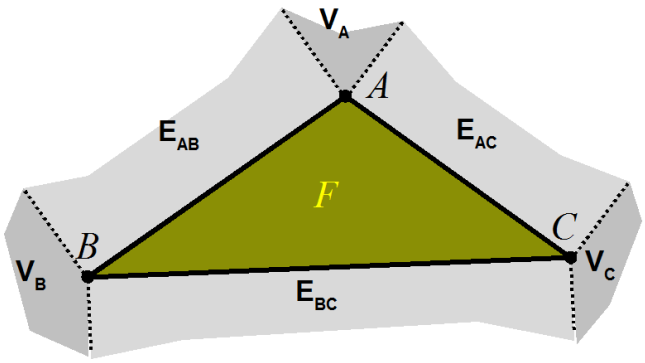


Figure 6. Using Voronoi regions for case differentiation [2].

A typical application of the Gilbert-Johnson-Keerthi distance algorithm is any kind of physics engine that uses real-time collision detection, which is commonly used in games or simulation systems. For instance, Ericson describes a way to effectively simplify a collision detection between two moving objects [2]. If two objects $A$ and $B$ both move along a specific vector $v_A$ respectively $v_B$, you can easily compress both movements into one by building the vector $v = v_A - v_B$. Assuming that object $A$ is now moving along this vector $v$, object $B$ can be regarded as fixed to its position, transforming the problem into a simpler one. Only one moving object has to be tested against a stationary object. If object $A$ is now extended by adding the vector $v$ to all of its vertices, you receive an object which contains all the area that A would have covered in its movement. Now, two stationary objects can be tested against each other in the familiar way with GJK. The procedure can be further sped up by starting an iteration with the simplex from the previous iteration, since it is likely to be close to the desired result in natural movements.

## 6 Conclusion

A fast and versatile procedure for computing the distance between convex objects was presented. Despite its age of 20 years, the Gilbert-Johnson-Keerthi distance algorithm is popular and widely used, since it is easy implementable, can handle many types of objects and is with linear time complexity a very fast algorithm in terms of collision detection procedures. It can be used for proximity queries, path planning or real-time collision detection. GJK's mathematical background is the reason why the algorithm takes some time to be mastered. However, it is the same reason that makes the algorithm so fast and versatile, especially the fact, that geometrical objects are solely described by support mappings. When applied, the GJK algorithm is usually adapted to the actual problem to be solved, and parts of it replaced by more suitable or intuitive procedures, providing improved time complexities up to constant time.

## References


[1] S. Cameron. Enhancing gjk: Computing minimum and penetration distances between convex polyhedra. In *Proceedings of International Conference on Robotics and Automation*, pages 3112–3117, 1997.

[2] C. Ericson. The gilbert-johnson-keerthi (gjk) algorithm. SIGGRAPH Presentation, 2004. Sony Computer Entertainment America.

[3] C. Ericson. *Real-Time Collision Detection*. Morgan Kaufmann, 2004.

[4] E. G. Gilbert and C.-P. Foo. Computing the distance between general convex objects in three-dimensional space. In *IEEE Transactions on Robotics and Automation*, volume 6, pages 53–61, February 1990.

[5] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. In *IEEE Journal of Robotics and Automation*, volume 4, pages 193–203, April 1988.

[6] G. van den Bergen. A fast and robust gjk implementation for collision detection of convex objects. Technical report, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1999.

[7] G. van den Bergen. *Collision Detection in Interaction 3D Environments*. Morgan Kaufmann, 2003.