Bachelor Seminar: The Gilbert - Johnson - Keerthi (GJK) Algorithm

Davor Jovanoski
University of Applied Sciences Burgenland

Department of Computer Science
University of Salzburg

Bachelor Seminar:

The Gilbert – Johnson – Keerthi (GJK) Algorithm

Davor Jovanoski
djovanos@cosy.sbg.ac.at
February 2008

Abstract

This paper represents the definition and implementation of the Gilbert – Johnson – Keerthi iterative algorithm for computing the minimum distance between two convex objects. GJK relies on a support function to generate simplexes closer to the correct solution, using the Minkowski sum of the two convex objects.

1 Introduction

The Gilbert-Johanson-Keerthi distance algorithm (GJK) is an iterative method for computing the distance between convex objects. The original idea was developed by E. G. Gilbert, D. W. Johnson, and S. S. Keerthi (1998) for convex polyhedra in 3D. The algorithm was later extended to handle convex objects in general (Gilbert and Foo 1990). An enhancement of the algorithm was also developed that computes penetration distances when the polyhedra are intersecting (Cameron 1997)[5].

GJK is extremely versatile, it can be applied to polytopes, quadrics[1], Minkowski sums of convex objects, and images of convex objects under affine transformation. Despite the fact that GJK can be difficult to grasp, since the algorithm requires quite a lot of nonintuitive mathematics to describe, implementing of GJK in not so hard, and the algorithm hardly needs to handle a special case.

From the viewpoint of mathematics, the distance between two points can be measured as the difference between the two vectors, generated from the origin and the two points. If these were point sets *A* and *B*, from the calculation of all the differences between the points of *A* and *B*, we will be able to find the minimal distance between the two point sets.

From the viewpoint of computer science, computation of all the differences costs time and resources. GJK exceeds this disadvantage.

The basic idea of the GJK is that instead of computing the minimum distance between *A* and *B*, the minimum distance between *A*-*B* and the origin is computed[7].

1 A surface defined by an algebraic equation of degree two is called a quadric. Spheres, circular cylinders, and circular cones are quadrics.

2 Overview

Minkowski sum, also known as dilation of two sets *A* and *B*, in Euclidean space is the result of adding every element of *A* to every element of *B*.

$$A+B=\{x+y : x\in A, y\in B\}$$

- If *A* is convex polygon with *n* vertices and *B* is convex polygon with *m* vertices, in the worst case the sum *A+B* has *n+m* vertices.

GJK is essentially a descent method for approximating the point closest to the origin of *A-B,* where *A* and *B* are general convex objects. We denote this point as *v(A-B),* where

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

We express the distance between *A* and *B* in terms of their Minkowski sum[2] *A-B* as

$$d(A,B)=min\{\|x-y\| : x\in A, y\in B\}$$

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

GJK approximates the point *v(A-B)* in the following way. In each iteration a simplex[3] is constructed that is contained in *A-B* and lies closer to the origin than the simplex constructed in the previous iteration. Simplex is the convex hull[4] of a set of *(n+1)* affine independent points in some Euclidean space of dimension *n* or higher. So a simplex can be a single point, a line segment, a triangle, a tetrahedron, ... We define W_k as the set of vertices of the simplex constructed in the *k-th* iteration $(k\geq 1)$, and $\boldsymbol{v_k}$ as *v(conv(Wk)),* the point of the simplex closest to the origin. Initially, we take $W_0=\emptyset$ and $\boldsymbol{v_0}$, an arbitrary point in *A-B*. Since *A-B* is convex and $W_k\subseteq A-B$, we see that $\boldsymbol{v_k}\in A-B$, and $\|\boldsymbol{v_k}\|\geq\|v(A-B)\|$ for all $(k\geq 0)$. So, the length of $\boldsymbol{v_k}$ is an upper bound for the distance between *A* and *B[2,3].*

GJK constructs each new simplex using a support mapping of *A-B. A support mapping* of a convex object *A* is a function s_A that maps a vector $\boldsymbol{v}$ to a point of *A*, according to

$$s_A(\boldsymbol{v})\in A \text{ such that } v\cdot s_A(\boldsymbol{v})=max\{\boldsymbol{v}\cdot x : x\in A\}$$

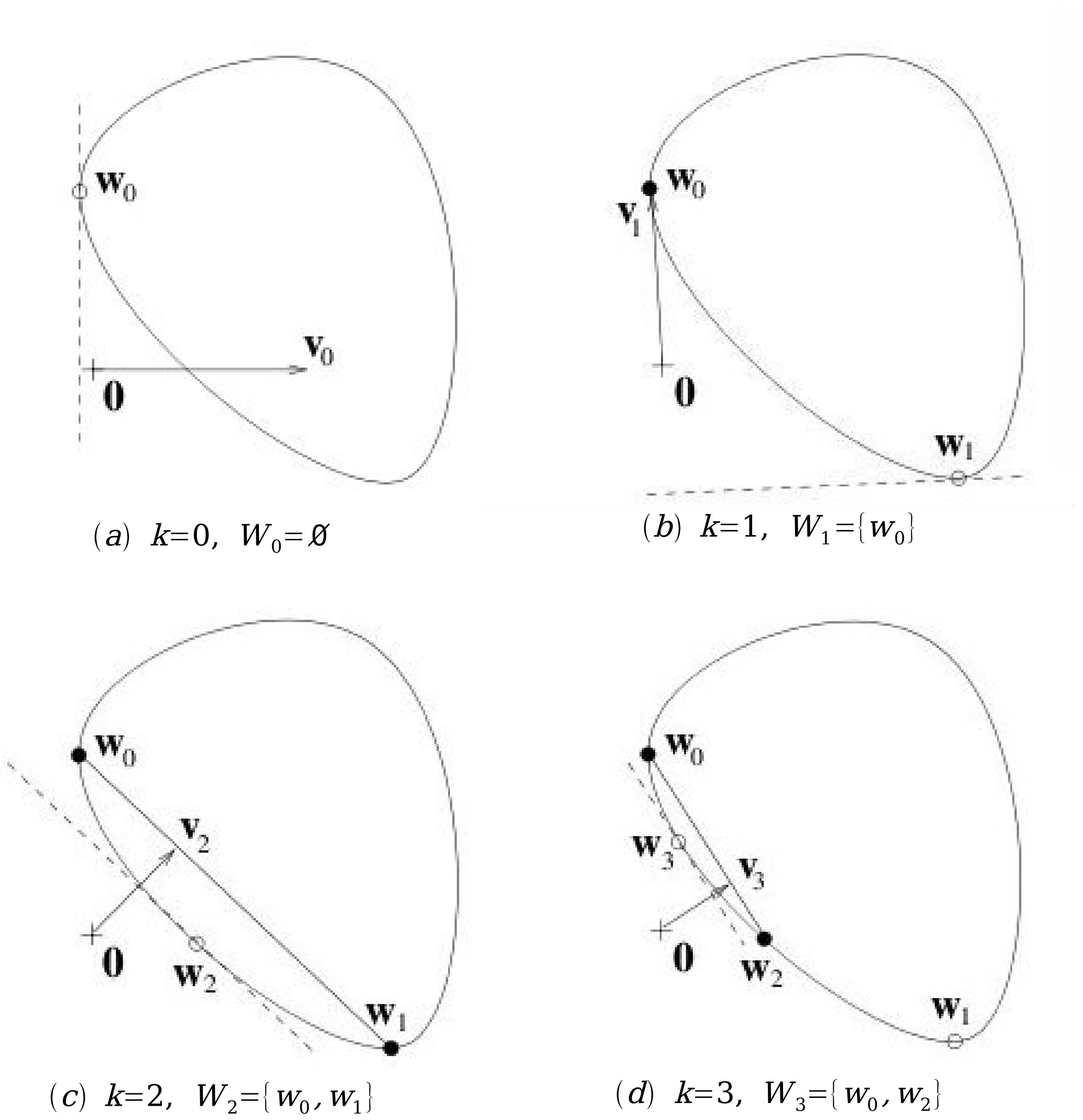
The result of a support mapping for a given vector is called a support point[2]. The computation of a support mapping function for different types of convex shapes is discussed later in this paper. For now lets assume that we have a support mapping s_{A-B} of *A-B.*

2 Minkowski difference seems more appropriate, but we avoid using this term since it is defined differently in many geometry texts.

3 Simplex or n-simplex is an n-dimensional analogue of a triangle.

4 Convex Hull - conv(), for a set of points C in a real vector space V is the minimal convex set containing C.

In each iteration we add a new support point $w_k = s_{(A-B)}(-\boldsymbol{v_k})$ as a vertex to the current simplex W_k. We take $\boldsymbol{v_{k+1}} = v(conv(W_k \cup \{w_k\}))$, the point closest to the origin of the new simplex. As W_{k+1}, we take the smallest set $X \subseteq W_k \cup \{w_k\}$, such that $\boldsymbol{v_{k+1}}$ is contained in *conv(X)*. We can see that exactly one such *X* exists and that it must be affine independent. So, what happens is that while we are adding vertices to the simplex, earlier vertices that are no longer necessary for supporting $\boldsymbol{v_{k+1}}$ are discarded[2].



Picture 2-1: Four iterations of the GJK algorithm. The dashed lines represent the support planes $H(-\boldsymbol{v_k}, \boldsymbol{v_k} \cdot w_k)$

Picture 2-1 illustrates a sequence of iterations of the GJK algorithm in two-dimensional space.

GJK-Algorithm, pseudo code

```
v:= “arbitrary point in A-B”;
W:= $\emptyset$;
u:=0;
close_enough:=false;
while not closeEnough and $\boldsymbol{v} \neq 0$ do begin
        $w := s_{A-B}(-\boldsymbol{v})$;
        $d := \boldsymbol{v} \cdot w / \|\boldsymbol{v}\|$;
        $u := max\{u, d\}$;
        $closeEnough := \|\boldsymbol{v}\| - u \leqslant e$;
       if not closeEnough then begin
              $\boldsymbol{v} := v(conv(W \cup \{w\}))$;
              $W :=$ smallest $X \subseteq W \cup \{w\}$ such that $\boldsymbol{v} \in conv(X)$;
      end
end
return $\|\boldsymbol{v}\|$
```

3 Convergence

The following theorem proves that the sequence $\{\boldsymbol{v_k}\}$ converges to $v(A-B)$. Theorem 1 shows that in each iteration step the new $\boldsymbol{v_{k+1}}$ must be closer to the origin than the previous one, except if the previous $\boldsymbol{v_k}$ was already the closest point[2] . The proof of this theorem uses *Lemma 1* and *Lemma 2.*

- **Theorem 1:** $\|\boldsymbol{v_{k+1}}\| \leqslant \|\boldsymbol{v_k}\|$, with equality only if $\boldsymbol{v_k} = v(A-B)$

  - **Proof:**
    First $\|\boldsymbol{v_{k+1}}\| = min\{\|x\| : x \in conv(W_k \cup \{w_k\})\} \leqslant \|\boldsymbol{v_k}\|$, since $\boldsymbol{v_k} \in conv(W_k)$ and $conv(W_k) \subseteq conv(W_k \cup \{w_k\})$. Suppose $\|\boldsymbol{v_{k+1}}\| = \|\boldsymbol{v_k}\|$. Then, $\|\boldsymbol{v_k}\| \leqslant \|x\|$, for any $x \in conv(W_k \cup \{w_k\})$. Since $\|\boldsymbol{v_k}\| \in conv(W_k)$, and the line segment $\overline{\boldsymbol{v_k} w_k}$ is contained in $conv(W_k \cup \{w_k\})$, we have $\|u\| \geq \|\boldsymbol{v_k}\|$ for any point *u* on the line segment $\overline{\boldsymbol{v_k} w_k}$. According to *Lemma 1,* it follows that $\|\boldsymbol{v}\|^2 - \boldsymbol{v_k} \cdot w_k \leqslant 0$. From *Lemma 2* we know that $\|\boldsymbol{v}\|^2 - \boldsymbol{v_k} \cdot w_k \geq 0$ and $\|\boldsymbol{v}\|^2 - \boldsymbol{v_k} \cdot w_k = 0$, which can only be the case if $\boldsymbol{v_k} = v(A-B)$.

- *Lemma 1:* Let ***v*** and *w* be vectors. The line segment connecting ***v*** and *w* contains a vector *u* for which $\|u\| < \|\boldsymbol{v}\|$ only if $\|\boldsymbol{v}\|^2 - \boldsymbol{v} \cdot w > 0$.

- **Proof:**
  Let $u = \boldsymbol{v} + \lambda(w - \boldsymbol{v})$. Then, for $0 \le \lambda \le 1$, u is contained by the line segment $\overline{\boldsymbol{v}w}$. We see that $\|u\|^2 - \|\boldsymbol{v}\|^2 = 2\lambda\, \boldsymbol{v} \cdot (w - \boldsymbol{v}) + \lambda^2 \|w - \boldsymbol{v}\|^2$. For $\|u\|^2 - \|\boldsymbol{v}\|^2 = 0$, we find the roots $\lambda_1 = 0$ and $\lambda_2 = -2\, \boldsymbol{v} \cdot (w - \boldsymbol{v}) / \|w - \boldsymbol{v}\|^2$. Since $\|w - \boldsymbol{v}\|^2 > 0$, we find that $\|u\|^2 - \|\boldsymbol{v}\|^2$ is positive for $\lambda \to \infty$. It follows that $\lambda_2 > 0$ only if $\|\boldsymbol{v}\|^2 - \boldsymbol{v} \cdot w > 0$. If $\|\boldsymbol{v}\|^2 - \boldsymbol{v} \cdot w \le 0$, then $\lambda_2 \le 0$, and $\|u\|^2 - \|\boldsymbol{v}\|^2$ is positive for all $0 \le \lambda \le 1$. Otherwise, $\|u\|^2 - \|\boldsymbol{v}\|^2 < 0$ for $\lambda_1 < \lambda < \lambda_2$. Since $[\lambda_1, \lambda_2] \cap [0,1] \ne \emptyset$, there must be a point u on the line segment $\overline{\boldsymbol{v}w}$. for which $\|u\| < \|\boldsymbol{v}\|$.

- *Lemma 2:* $\|\boldsymbol{v}\|^2 - \boldsymbol{v_k} \cdot w_k \ge 0$, with equality only if $\boldsymbol{v_k} = v(A - B)$.

  - **Proof:**
    Since $-\boldsymbol{v_k} \cdot w_k = -\boldsymbol{v_k} \cdot s_{A-B}(-\boldsymbol{v_k}) \ge -\boldsymbol{v_k} \cdot x$ for all $x \in A - B$, we have $\boldsymbol{v_k} \cdot x - \boldsymbol{v_k} \cdot w_k \ge 0$ for $x \in A - B$. In particular, $\|\boldsymbol{v_k}\|^2 - \boldsymbol{v_k} \cdot w_k \ge 0$. Suppose $\|\boldsymbol{v_k}\|^2 - \boldsymbol{v_k} \cdot w_k = 0$. Then, for any $x \in A - B$,

$$\begin{aligned}\|\boldsymbol{v_k}\|^2 &\le \|\boldsymbol{v_k}\|^2 + \|x - \boldsymbol{v_k}\|^2 \\ &\to \|x\|^2 - 2(\boldsymbol{v_k} \cdot x - \|\boldsymbol{v_k}\|^2) \\ &\to \|x\|^2 - 2(\boldsymbol{v_k} \cdot x - \boldsymbol{v_k} \cdot w_k) \\ &\le \|x\|^2,\end{aligned}$$

and thus, $\boldsymbol{v_k} = v(A - B)$.

4 Support Mapping

The versatility of GJK is a result of the fact that it relies solely on support mappings for reading the geometry of an object. A support mapping fully describes the geometry of a convex object and can be viewed as an implicit representation of the object.

The class of objects we consider is recursively constructed from

- Convex primitives
  - Polytopes (line segments, triangles, boxes and other convex polyhedra)
  - Quadrics (spheres, cones and cylinders)
- Images of convex objects under affine transformation
- Minkowski sums of two convex objects
- Convex hulls of a collection of convex objects

In this section of the paper we supply a support mapping for each primitive. The support mappings for affine transformations, Minkowski sums, and convex hulls are derived from the support mappings of their child objects.

4.1 Convex Primitives

Polytope

For a polytope *P*, we may take $s_P(v) = s_{vert(P)}(v)$ that is,

$$s_P(v) \in vert(P), \quad where \quad v \cdot s_p(v) = max\{v \cdot p \ : \ p \in vert(P)\}$$

Apparently, worst case for computation of a support point of a polytope can be determined in linear time in the number of the vertices of the polytope. Simply search the list of vertices for a vertex *p* for which $v \cdot p$ is maximum. For simple polytopes, such as simplexes this is the fastest way to determine a support point. However, for more complex polytopes this isn't the case.

Support point can be found in *O(log n)* time for two- and three-dimensional polytopes represented by the Dobkin-Kirkpatrick hierarchical representation[2]. But it has been mentioned in [2,3], by exploiting frame coherence, the cost of computing a support point of a convex polyhedron can be reduced to almost constant time. For the computation of the supporting point *p,* the *polytope's vertex adjacency graph* is used, and each edge on the polytope is represented as an edge in the graph, thus in this way a supporting point that lies close to the previously returned support point can be found using the *hill climbing* technique. *Hill climbing* is an optimization technique which belongs to the family of local search. The algorithm starts with a random solution to the problem. It sequentially makes small changes to the solution, each time improving. At some point the algorithm arrives at a point where it cannot see any further improvements, and the algorithm terminates.

Computing a support point *p = s*$p = s_p(v)$ using ***hill climbing*** on the polytope's vertex adjacency graph.

```
p:="cached support vertex"
repeat
    optimal:=true;
    for $q \in adj(p)$ do
    begin
        if $v \cdot q > v \cdot p$ then
        begin
            p:=q;
            optimal:=false;
        end
    end
until optimal
```

Box

A Box primitive is a rectangular parallelepiped centered at the origin and aligned with the coordinate axes. Let *A* be a Box with extents $2n_x$, $2n_y$, and $2n_z$. Then we take as support mapping for *A*,

$$S_a((x,y,z)^T)=(sgn(x)n_x, sgn(y)n_y, sgn(z)n_z)^T$$

where *sgn(x)=-1*, if $x < 0$, and *1*, otherwise.

Sphere

A Sphere primitive is a ball centered at the origin[3]. The support mapping of a Sphere *A* with radius r is

$$S_A(v)=\left\{\begin{matrix} \frac{r}{\|v\|}v \;\; if\, v\neq 0 \\ 0 \;\; otherwise \end{matrix}\right\}$$

Cone

A Cone primitive is a capped cone that is centered at the origin with central axis aligned with the *y-axis*. Let *A* be a Cone with a radius *r* at its base *y =-n*, and with its apex[5] at *y = n*. Then for the top angle *α* we have

$$\sin(\alpha)=\frac{r}{\sqrt{r^2+(2n)^2}} \quad \text{Let} \quad \sigma=\sqrt{x^2+z^2}$$

σ - the distance from $(x,y,z)^T$ to the *y-axis*. We choose as support mapping for *A,* the mapping

$$s_A((x,y,z)^T)=\left\{\begin{matrix} (0,n,0)^T & if\, y>\|(x,y,z)^T\|\sin(\alpha) \\ (\frac{r}{\sigma}x,-n,\frac{r}{\sigma}z)^T & else, if\, \sigma>0 \\ (0,-n,0)^T & otherwise \end{matrix}\right\}$$

5 Apex is a descriptive label for a visual singular highest or most distant point or vertex, usually contrasting with the opposite side which is called the base.

Cylinder

A Cylinder primitive is a capped cylinder that is centered at the origin with central axis aligned with the *y-axis*. Let *A* be a Cylinder with a radius *r,* top at *y=n,* and bottom at *y=-n.* We find as support mapping for *A* the mapping

$$S_A((x,y,z)^T)=\begin{Bmatrix}(\frac{r}{\sigma}x, sgn(y)n, \frac{r}{\sigma}z)^T & if\,\sigma>0\\ (0, sgn(y)n, 0)^T & otherwise\end{Bmatrix}$$

where *sgn(x)=-1*, if *x < 0*, and *1*, otherwise.

4.2 Affine Transformation

An affine transformation $T(x)=Bx+c$ is a linear transformation followed by a translation. An animation of an object is due changing of his placement (position, orientation and scaling) of its local coordinate system. The computation of the support point for image under affine transformation is defined by the following Theorem[2,3].

- **Theorem 2**: *Given* s_A, *a support mapping of object A, and* $T(x)=Bx+c$, *an affine transformation, a support mapping for T(A), the image of A under T, is*

$$s_{T(A)}(v)=T(S_A(B^T v))$$

- **Proof:**
  A support mapping $s_{T(A)}$ is characterized by

$$v\cdot s_{T(A)}(v)=max\{v\cdot T(x):x\in A\}$$

We rewrite the right member of this equation using the following deduction.

$$v\cdot T(x)=v\cdot Bx+v\cdot c=v^T Bx+v\cdot c=(B^T v)^T x+v\cdot c=(B^T v)\cdot x+v\cdot c$$

This equation is used in the steps marked by ***def*** in the following deduction.

$$\begin{aligned}max\{v\cdot T(x):x\in A\} &\overset{def}{=} max\{(B^T v)\cdot x+v\cdot c:x\in A\}\\ &\rightarrow max\{(B^T v)\cdot x:x\in A\}+v\cdot c\\ &\rightarrow (B^T v)\cdot s_A(B^T v)+v\cdot c\\ &\overset{def}{=} v\cdot T(s_A(B^T v))\end{aligned}$$

Hence, $s_{T(A)}(v)=T(s_A(B^T v))$ is a support mapping of $T(A)$.

4.3 Minkowski Sum

Given two convex objects *A* and *B*, and the support mappings s_A and s_B,

$$s_{A+B}(v) = s_A(v) + s_B(v)$$

is a proper support mapping for *A+B*, the Minkowski sum of A and B, therefore an explicit representation of A+B isn't needed. For -B, the mirror image of the convex object B, we find the support mapping

$$s_{-B}(v) = -s_B(-v)$$

So therefore the proper mapping for *A-B* is,

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

The versatility of GJK lies in the fact that it uses a support mapping for reading the geometry of the CSO[6]. Since it is easy to compute support points for CSOs of different types of convex objects, we can combine any two types of convex primitives in the GJK algorithm.

4.4 Convex Hull

Earlier we saw that for polytopes we may choose a vertex *p*, for which the dot product $v \cdot p$ is maximum, as the support point for *v*. This idea can be generalized to collections of arbitrary convex objects, which leads for a support mapping for *conv(X),* the convex hull of *X*, is

$$s_{conv(X)}(v) = s_A(V),\ where\ A \in X\ and\ v \cdot s_A(v) = max\{v \cdot s_B(v) : B \in X\}.$$

In other words, simply compute support points $s_A(v)$ for all $A \in X,$ and select the point *p* where $v \cdot p$ is maximum.

5 Penetration Depth

First, we should define what a penetration depth is.

Def: Let *A* and *B* be two intersecting convex objects. The *penetration depth* of *A* and *B* is the minimum distance by which *A* has to be translated so that *A* and *B* do not intersect.

In this part of the paper we will represent an iterative method which is very similar to GJK for computing the penetration depth. Like GJK, it uses only support mapping for reading the geometry of the objects, and therefore it is acceptable to the same class of objects as GJK. The only requirement as initial state is that the Minkowski sum *A-B* should contain the origin.

6 CSO - configuration space obstacle, the Minkowski sum (difference) A-B.

For intersecting objects, GJK terminates when a simplex is generated that contains the origin. In most cases in 3D, the simplex is a tetrahedron. As mentioned before all support mappings return support points on the boundary of the object. So, the tetrahedron generated by GJK is a proper initial polytope for the penetration depth method.

The method presented here is also presented in [2] and, is similar to an earlier method by Cameron for estimating the penetration depth [5], while in this case, it uses the simplex returned by GJK, to determine the penetration depth. However, unlike Cameron's method, this method returns the exact penetration depth of an intersecting pair of polytopes.

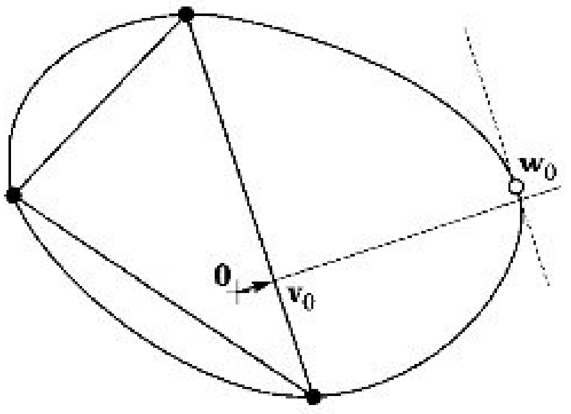
As mentioned earlier, the penetration depth of a pair of intersecting objects *A* and *B* is a point on the boundary of *A-B* closest to the origin. This algorithm for finding such a point starts with a polytope that contains the origin and “expands” it by adding vertices that lie on the boundary. The added vertices are generated by the support mapping of *A-B*. The basic strategy is to iteratively pick the facet[7] of the polytope closest to the origin and subdivide it using support points as additional vertices.

For easier comprehension we will explain the algorithm in 2D. At fist we blow up a convex polygon by splitting the edges. We start of with a simple polygon such as triangle that contains the origin, in our case the simplex result from the GJK. For each edge *X* of the polygon, we compute *v(aff(X))*, the point on the affine hull[8] of the edge (i.e. the line through the edge's vertices) that lies closest to the origin. Since the polygon is convex, the point *v* must be an internal point of edge *X*.

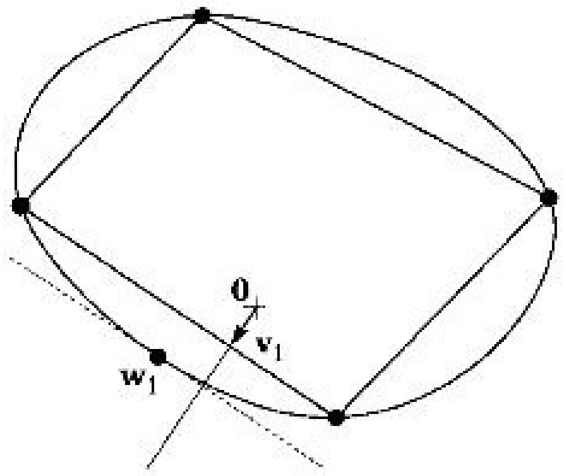
The length of the vector v is lower bound for the penetration depth, since the polygon is contained in the CSO. In each iteration step, the closest edge is split by intersecting the support point $s_{A-B}(v)$ as new vertex. For the two new edges we recompute the points closest to the origin on the affine hulls of the edges and repeat this procedure until $\boldsymbol{v}$ lies sufficiently close to the penetration depth. Picture 4-1 shows a sequence of iterations of this algorithm, which in [2] is referred as the *expanding – polytope algorithm* (EPA).

7 Facets are flat faces on geometric shapes.

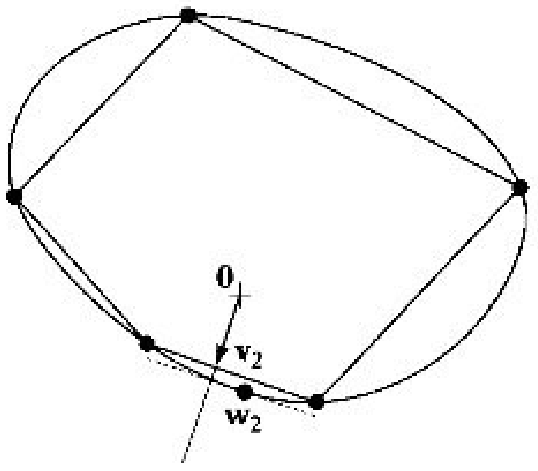
8 Affine hull- aff(), of a set *S* in Euclidean Space is the smallest affine set containing *S*.



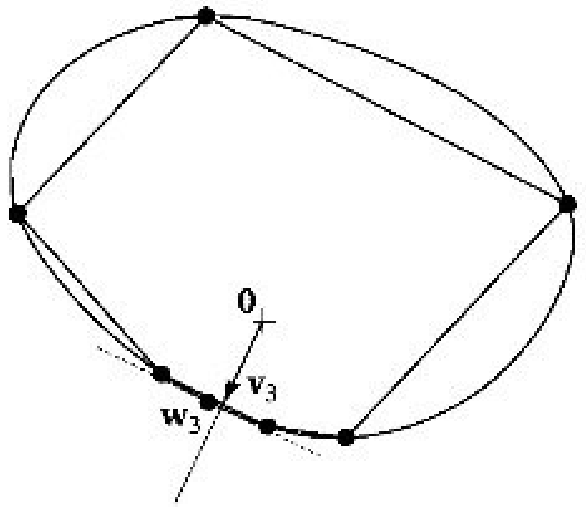
(a) k=0



(b) k=1



(c) k=2



(d) k=3

Picture 4-1: A sequence of iterations of the expanding polytope algorithm. An arrow denotes a point v on the polygon's boundary closest to the origin. The dashed lines represent the support planes $H(v_k, -v_k \cdot w_k)$

6 Conclusion

We have presented a practical algorithm for measuring the minimal distance between two convex objects; also we have presented usage of a similar algorithm if an intersection took place allowing us the measuring of the penetration depth, from which we can conclude the versatility of the algorithm.

Unlike many other distance algorithms, it does not require the geometry data to be stored in any specific format, but instead relies solely on a support mapping function and using the result in its next iteration. Relying on the support mapping function to read the geometry of the objects gives this algorithm great advantage, because every enhancement on the support mapping function leads to enhancement of the GJK algorithm.

The algorithm's stability, speed, and small storage footprint make it popular for Real-time collision detection, especially in physics engines for video games.

As a final conclusion, we can say that GJK is one of the finest algorithms for measuring distances due to its versatility, easy implementation, possibility for enhancements and, most important, its speed.

7 References

[1] Ericson Christer. *Real-time collision detection*, from Morgan Kaufmann Publishers Inc, 2005

[2] Gino Van Den Bergen, *Collision Detection in Interactive 3D Environments,* from Morgan Kaufmann Publishers Inc, 2004

[3] Gino Van Den Bergen, *A Fast and Robust GJK Implementation for Collision Detection of Convex Objects*, 1999

[4] Philip J. Schneider & David H. Eberly, *Geometric Tools for Computer Graphics,* from Morgan Kaufmann Publishers Inc, 2003

[5] Stephen A. Cameron, *Enhancing GJK: Computing Minimum and Penetration Distances between Convex Polyhedra*, 1997

[6] Stephen A. Cameron & R. K. Culley, *Determining the minimum Translational Distance between Two Convex Polyhedra*, 2002

[7] Tomas A. Möller & Eric Heines, *Real-Time Rendering*, (2nd edition) from A.K. Peters Ltd. 2002