UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUÍS FERNANDO GARCIA JALES

ALGORITMO GILBERT-JOHNSON-KEERTHI, SUAS MELHORIAS E MODIFICAÇÕES

RIO DE JANEIRO
2024

LUÍS FERNANDO GARCIA JALES

ALGORITMO GILBERT-JOHNSON-KEERTHI, SUAS MELHORIAS E MODIFICAÇÕES

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Professor Vinícius Gusmão Pereira de Sá

RIO DE JANEIRO

2024

CIP - Catalogação na Publicação

Jales, Luís Fernando Garcia
J26a Algoritmo Gilbert-Johnson-Keerthi: suas melhorias e modificações / Luís Fernando Garcia Jales. -- Rio de Janeiro, 2024.
50 f.

Orientador: Vinícius Gusmão Pereira de Sá.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Computação, Bacharel em Ciência da Computação, 2024.

1. Detecção de colisão. 2. Algoritmo. I. Sá, Vinícius Gusmão Pereira de, orient. II. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

LUÍS FERNANDO GARCIA JALES

ALGORITMO GILBERT-JOHNSON-KEERTHI, SUAS MELHORIAS E MODIFICAÇÕES

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado no dia 23 de agosto de 2024

BANCA EXAMINADORA:

gov.br Documento assinado digitalmente
VINICIUS GUSMAO PEREIRA DE SA
Data: 27/08/2024 18:22:06-0300
Verifique em https://validar.iti.gov.br

Prof. Vinícius Gusmão Pereira de Sá
D.Sc (UFRJ)

gov.br Documento assinado digitalmente
JOAO ANTONIO RECIO DA PAIXAO
Data: 27/08/2024 18:57:37-0300
Verifique em https://validar.iti.gov.br

Prof. João Antônio Recio da Paixão
D.Sc (UFRJ)

gov.br Documento assinado digitalmente
MARCELLO GOULART TEIXEIRA
Data: 28/08/2024 06:26:50-0300
Verifique em https://validar.iti.gov.br

Prof. Marcello Goulart Teixeira
D.Sc (UFRJ)

I dedicate this paper to my mother Simone Garcia, to my stepfather Mauro Leão Gomes, and to my three pets: Gabriela, Yasmin, Clara.

RESUMO

Algoritmos de detecção de colisão são muito importantes em várias áreas, tais como robótica, física computacional, computação gráfica, e outras. Ao longo do tempo, o número de vértices de modelos tridimensionais vem aumentando cada vez mais. Isso acaba demandando algoritmos cada vez mais rápidos em detecção de colisão de fase estreita. Um dos algoritmos mais versáteis e rápidos para isso é o algoritmo Gilbert-Johnson-Keerthi, ou simplesmente GJK, que é um algoritmo famoso para computar a distância entre dois politopos convexos. O objetivo principal desse trabalho é revisar o artigo original do algoritmo GJK, apresentando uma descrição completa e a prova de sua corretude (e que o algoritmo sempre termina). Como esse algoritmo depende de um subalgoritmo de distância, uma descrição completa do subalgoritmo de distância de Johnson e uma revisão da prova de sua corretude e que o algoritmo sempre termina também estão inclusos aqui. Ademais, esse trabalho também apresenta uma descrição completa de uma melhoria que usa *hill climbing* para alcançar uma complexidade de tempo quase constante e uma modificação para computar um eixo de separação, além de mencionar brevemente outras modificações: a primeira serve para resolver detecção de colisão contínua; a outra é o Algoritmo da Expansão de Politopo, ou simplesmente EPA, para computar a profundidade de penetração.

Palavras-chave: detecção de colisão; algoritmo.

ABSTRACT

Collision detection algorithms are very important in many fields such as robotics, computational physics, computer graphics, and more. Over time, the number of vertices in three-dimensional models has been growing more and more, demanding faster algorithms in narrow-phase collision detection. One of the most versatile and fast algorithms for that is the Gilbert-Johnson-Keerthi algorithm, GJK for short, which is a famous algorithm for computing the distance between two convex polytopes. The main objective of this work is to review the original paper of the GJK algorithm by presenting a full description of the algorithm, along with a proof of its correctness and termination. Since the algorithm relies on a distance sub-algorithm, a complete description of Johnson's distance sub-algorithm and a review of its correctness and termination proof are covered here as well. In addition, this work also presents a full description of an improvement that uses *hill climbing* to achieve an almost constant time complexity and a modification for computing a separating axis, and it briefly mentions two other possible modifications: the first one addresses continuous collision detection; the other one is the Expanding Polytope Algorithm, EPA for short, for computing penetration depth.

Keywords: collision detection; algorithm.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

iff	If and only if.
i.e.	Id est. Same as "that is".
GJK	Gilbert-Johnson-Keerthi
SAT	Separating axis theorem
ISA	Incremental Separating Axis
EPA	Expanding Polytope Algorithm

LISTA DE SÍMBOLOS

$\mathbf{0}$	Zero vector.
$\|\cdot\|$	Euclidean norm.
$\top$	Inner product.
$\emptyset$	Empty set.
$\setminus$	Relative complement
$\oplus$	Minkowski sum.
$\ominus$	Minkowski difference.
$\#$	Cardinality (number of elements in a set).
$\subseteq$	Subset.
$\subset$	Strict subset.
$\{\cdot \mid \cdot\}$	Set-builder notation.
$\mathcal{P}$	Power set.
$\sum$	Summation.
∂	Partial derivative.
$\wedge$	Logical conjunction.
$\vee$	Logical disjunction.
$\Longrightarrow$	Logical implication.
$\Longleftarrow$	Reversed logical implication.
$\Longleftrightarrow$	Logical equivalence.
$\forall$	Universal quantification.
$\exists$	Existential quantification.
O	Big O notation.
Ω	Big Omega notation.
Θ	Big Theta notation.

SUMÁRIO

1 INTRODUCTION

Collision detection is a term that encompasses a collection of problems arising from the apparently simple classical problem of detecting whether two geometrical objects intersect. From there, new problems came to light, and they bifurcated into two classes of collision detection: broad-phase and narrow-phase ones. Broad-phase collision detection is the class of problems related to reducing the number of collision tests between two objects from a large number of objects in space. Narrow-phase collision detection is the class of problems aimed at detecting collision between two objects, or even computing further information like the distance between the objects or the contact points in case of intersection. These and other concepts about collision detection are well covered in (ERICSON, 2004). Naturally, collision detection is one of the main problems in fields like robotics, computational physics, computer graphics, etc, as mentioned in the introductory sessions of (GILBERT; JOHNSON; KEERTHI, 1988; CAMERON, 1997; MONTANARI; PETRINIC, 2016).

The Gilbert-Johnson-Keerthi algorithm, or GJK for short, is an algorithm for computing the distance between two convex polytopes (GILBERT; JOHNSON; KEERTHI, 1988). Empirical experiments have shown this algorithm runs in approximate linear time (GILBERT; JOHNSON; KEERTHI, 1988). Furthermore, the GJK can be enhanced in order to perform in almost constant time complexity (CAMERON, 1997; BERGEN, 1999), while it is also possible to modify it for computing more information in narrow-phase collision detection, among these: a pair of nearest points or a point of intersection (GILBERT; JOHNSON; KEERTHI, 1988), penetration depth (BERGEN, 2001), separating axis (BERGEN, 1999), time of impact (BERGEN, 2004), and perhaps other features. This algorithm is not restricted only to polytopes because it can be adapted to work with quadrics (BERGEN, 1999) and spherical extensions (GILBERT; JOHNSON; KEERTHI, 1988). Ellipses and capsules are some examples. Also, the GJK algorithm can solve continuous collision detection for linear trajectories (BERGEN, 2004) since the algorithm can be adapted to deal with affine transformations and Minkowski sums (GILBERT; JOHNSON; KEERTHI, 1988). All these features lead the GJK algorithm to be considered one of the most versatile algorithms, if not the most versatile, in collision detection. Although the GJK algorithm was developed almost 40 years ago (GILBERT; JOHNSON; KEERTHI, 1988), there are still contributions being made to improve the speed of GJK, such as (LIDECN; SIVIC; CARPENTIER, 2022) , where they compare the GJK to a similar, Gradient Descent, method called Frank-Wolfe and use the Nesterov acceleration to improve the convergence speed. A relatively recent paper (MONTANARI; PETRINIC, 2016) shows the current most numerically stable distance sub-algorithm.

In Chapter 2, we cover some crucial concepts to fully understand the GJK algorithm,

which are some definitions such as Minkowski difference, distance vector, support, and simplex, as well as some important theorems. More importantly, we prove (17, 21) from (GILBERT; JOHNSON; KEERTHI, 1988), which has not been proven, and we prove theorem 2.2, which is crucial to understand the distance sub-algorithm. In Chapter 3, we describe the GJK algorithm, we show a different yet detailed proof of theorem 1 from (GILBERT; JOHNSON; KEERTHI, 1988) for its correctness, and we revisit theorem 2 from (GILBERT; JOHNSON; KEERTHI, 1988) for its termination. In Chapter 4, we cover the distance sub-algorithm and fully describe in detail the whole theory behind Johnson's distance sub-algorithm, which is the one used in the original paper (GILBERT; JOHNSON; KEERTHI, 1988). We also show a step-by-step proof of theorem 3 from (GILBERT; JOHNSON; KEERTHI, 1988). In Chapter 5, we discuss the time complexity of the GJK algorithm and present an improvement that uses *hill climbing* (CAMERON, 1997) to achieve almost constant time complexity. We construct a counterexample to prove the *hill climbing* method is actually incorrect, yet we show a procedure that corrects it. Also, we show a modification called Incremental Separating Axis GJK or ISA-GJK for short (BERGEN, 1999), and cover briefly two other modifications, which are ray casting with GJK (BERGEN, 2004) and the Expanding Polytope Algorithm or EPA for short (BERGEN, 2001).

2 MAIN CONCEPTS

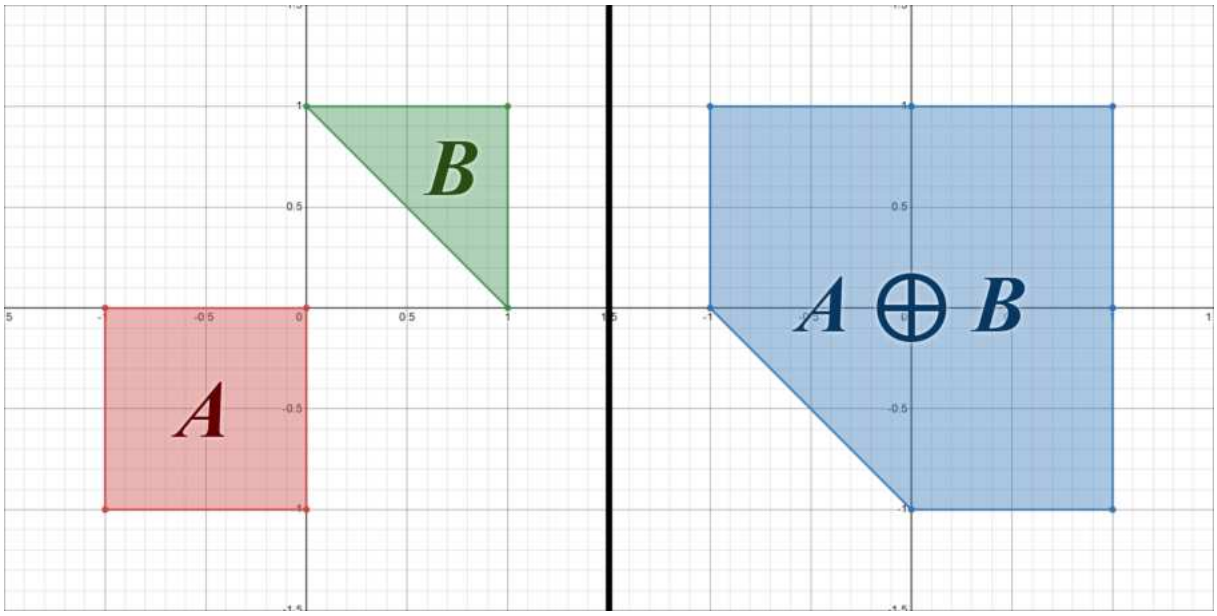
In order to fully understand the Gilbert-Johnson-Keerthi algorithm and its enhanced version, some important concepts and their properties that will be used throughout this paper must be defined beforehand. Those concepts are, in order: Minkowski sum and difference, convexity, distance, support, and simplex.

All those concepts can be generalized to some more abstract algebraic structures. However, for the sake of simplicity, the term region will henceforth mean a non-empty compact subset of the Euclidean affine space, often represented by a set $X \subset \mathbb{R}^n$ so we can have strong properties from real analysis. For example, using that definition of a region, we could use minimum and maximum without concerns about infimum or supremum.

Definition 2.1 (Minkowski Sum and Minkowski Difference)**.** Given two sets A and B, the Minkowski sum $A \oplus B$ is the set of all sums between an element from A and an element from B. More formally:

$$A \oplus B = \{a + b \mid a \in A, b \in B\}.$$

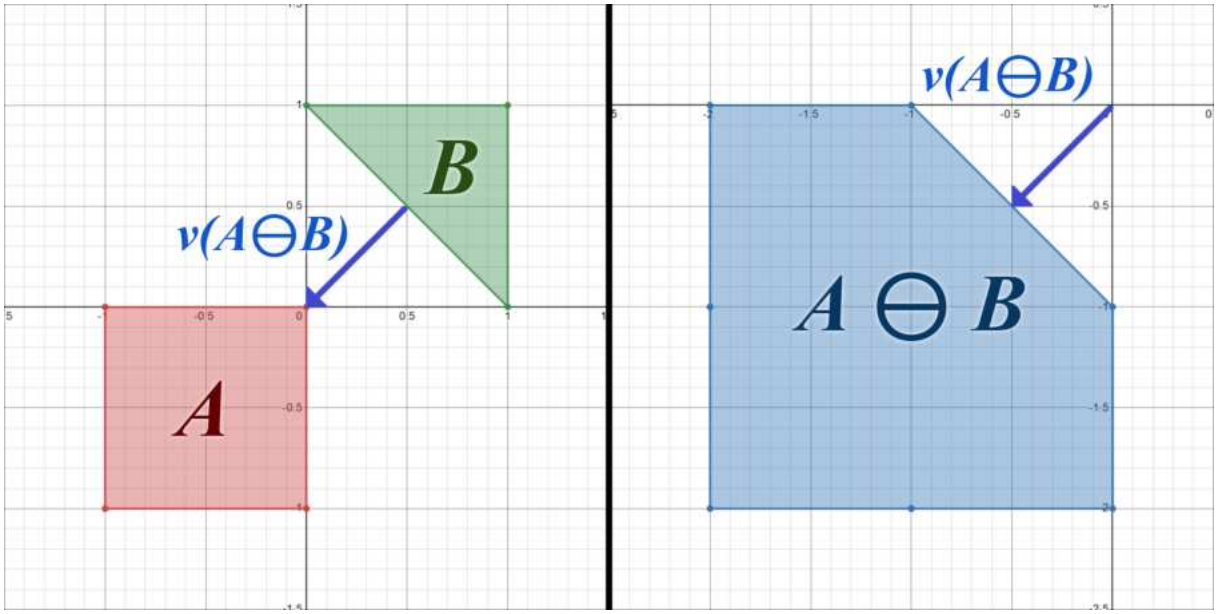
Figure 1 – Visualization of Minkowski sum



Analogously and more importantly, the Minkowski difference $A \ominus B$ is the set of all subtractions between an element from A and an element from B in that order, meaning:

$$A \ominus B = \{a - b \mid a \in A, b \in B\}.$$

Figure 2 – Visualization of Minkowski difference and distance vector



$v(A \ominus B)$ is the distance vector to A from B.

Definition 2.2 (Affine Hull and Convex hull)**.** For a set of points $P \subset \mathbb{R}^n$, the affine hull of P, denoted by $\text{aff}(P)$, is a set of all affine combinations of points in P. More formally:

$$\text{aff } P = \left\{ \sum_{p \in P} \lambda_p p \mid p \in P, \lambda_p \in \mathbb{R}, \sum_{p \in P} \lambda_p = 1 \right\}$$

that is, the linear combination of point $p \in P$ with coordinates λ_p is a point in $\text{aff}(P)$ iff the sum of all coordinates λ_p is equal to 1.

Similarly, the convex hull of P, denoted by $\text{conv}(P)$, is a subset of the affine hull where all the coordinates λ_p must be non-negative. More formally:

$$\text{conv } P = \left\{ \sum_{p \in P} \lambda_p p \mid p \in P, 0 \leq \lambda_p, \sum_{p \in P} \lambda_p = 1 \right\}$$

Definition 2.3 (Distance)**.** The distance between two regions A, B is defined as the minimum distance between two points $a \in A, b \in B$. More formally:

$$d(A, B) = \min\{\|a - b\| \mid a \in A, b \in B\}$$

where $d(A, B)$ is the distance between A and B. Equivalently, using the Minkowski difference:

$$d(A, B) = \min\{\|x\| \mid x \in A \ominus B\}.$$

The immediate conclusion is that the distance between two regions can be viewed as the norm of some vector of the Minkowski difference that has the smallest norm. From

this perspective, similarly to (GILBERT; JOHNSON; KEERTHI, 1988), a minimum norm vector $v(X)$ can be defined as:

$$v(X) \in X, \|v(X)\| = \min\{\|x\| \mid x \in X\}$$

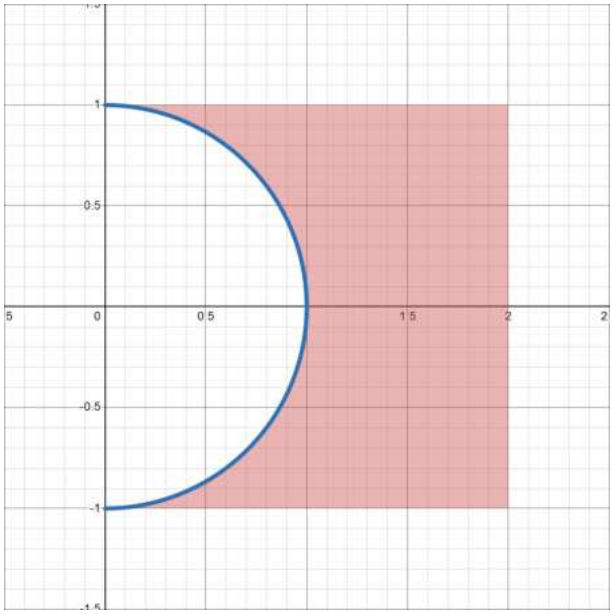
and then we conclude that:

$$\|v(A \ominus B)\| = \min\{\|x\| \mid x \in A \ominus B\} = d(A, B).$$

From this result, $v(A \ominus B)$ is called a distance vector to A from B (see figure 2).

When considering vector spaces as affine spaces, the distance between two regions may be viewed as the distance of a nearest point to the origin in the Minkowski difference with the origin being the zero vector, as considered by some important authors (MONTANARI; PETRINIC, 2016; GILBERT; JOHNSON; KEERTHI, 1988; CAMERON, 1997; BERGEN, 1999) to name but a few. Depending on the shape of the region, there might be more than one vector with the same minimum norm, so the distance vector is not always unique (see figure 3).

Figure 3 – A region where the minimum norm vector is not unique



In this non-convex region, all points in the blue arc are minimum norm vectors.

Theorem 2.1. *Given a convex region $X \subset \mathbb{R}^n$, if X is convex, then $v(X)$ is unique.*

Proof. Let us suppose by contradiction that there are two distinct minimum norm vectors $v_1, v_2 \in X$, that is, $v_1 \neq v_2$ and $\|v_1\| = \|v_2\| = \ell$. Because X is convex, the line segment $\text{conv}\{v_1, v_2\}$ is a subset of X. Also, $\text{conv}\{v_1, v_2\}$ is a chord in the sphere of radius ℓ centered at $\mathbf{0}$, hence there is a vector in the chord whose norm is less than ℓ, which is a contradiction.

□

From that result, as long X is convex, an equivalently way to redefine $v(X)$ using destructuring is:

$$\{v(X)\} = \underset{x \in X}{\operatorname{argmin}} \|x\|.$$

Corollary 2.1.1. *If two regions are convex, then there is a unique distance vector.*

Proof. Assume two convex regions $A, B \subset \mathbb{R}^n$. $A \ominus B$ can be easily proven convex by theorem 1.1.2 in (SCHNEIDER, 1993). Consequently, the distance vector $v(A \ominus B)$ is proven to be unique by 2.1.

□

Theorem 2.2. *Let $P \subset \mathbb{R}^n$ be any set of points and let three statements be:*

1. *There are all positive coordinates for $v(\text{conv}(P))$,*
2. *There are all positive coordinates for $v(\text{aff}(P))$,*
3. *$v(\text{aff}(P)) = v(\text{conv}(P))$.*

Therefore:

$$(1.) \iff (2.)$$

$$(1.) \vee (2.) \implies (3.)$$

Proof. Let $\mathbf{v_c} = v(\text{conv}(P))$ and $\mathbf{v_a} = v(\text{aff}(P))$.

To prove (1.) $\implies$ (3.), suppose there are all positive coordinates for $\mathbf{v_c}$. Since $\text{conv}(P) \subset \text{aff}(P)$, $\|\mathbf{v_a}\| \leq \|\mathbf{v_c}\|$. Now, suppose by contradiction that $\|\mathbf{v_a}\| < \|\mathbf{v_c}\|$. Because all coordinates are positive, $\mathbf{v_c}$ is an interior point of $\text{conv}(P)$, therefore, there is a point $p \in \text{conv}(P)$ in the line segment between $\text{conv}\{\mathbf{v_c}, \mathbf{v_a}\}$ where $\|p\| < \|\mathbf{v_c}\|$, which is a contradiction. Consequently, $\|\mathbf{v_a}\| = \|\mathbf{v_c}\|$. By theorem 2.1, $\mathbf{v_a} = \mathbf{v_c}$.

To prove (2.) $\implies$ (3.), (2.) implies that $\mathbf{v_a} \in \text{conv}(P)$, therefore, $\|\mathbf{v_c}\| \leq \|\mathbf{v_a}\|$. Since $\text{conv}(P) \subset \text{aff}(P)$, $\|\mathbf{v_a}\| \leq \|\mathbf{v_c}\|$. Therefore, $\mathbf{v_a} = \mathbf{v_c}$.

To prove (1.) $\implies$ (2.), assume (1.), therefore, (3.). A convex combination is a special case of an affine combination, therefore, since all coordinates for $\mathbf{v_c} = \mathbf{v_a}$ are positive, we conclude (2.).

To prove (2.) $\implies$ (1.), assume (2.), therefore, (3.). Since all coordinates for $\mathbf{v_a} = \mathbf{v_c}$ are positive, we conclude (1.).

□

Definition 2.4 (Support)**.** Let $X \subset \mathbb{R}^n$ be a region, $x \in X$ a point, and $H \subset \mathbb{R}^n$ a hyperplane. It is said that H supports X at x iff: x is a point of both X and H, and X is entirely contained in one of the half-spaces of H (since a hyperplane divides the space in two).

To construct hyperplanes, let the hyperplane $H(a, b)$ be defined as:

$$H(a, b) = \{p \mid p \in \mathbb{R}^n, a^\top p = b\},$$

where the non zero vector $a \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ is orthogonal to the hyperplane and $b \in \mathbb{R}$ is the offset to the origin in the a direction. Analogously, $H^+(a, b)$ and $H^-(a, b)$ are the positive and negative half-spaces respectively, defined as:

$$H^+(a, b) = \{p \mid p \in \mathbb{R}^n, a^\top p \geq b\}$$

$$H^-(a, b) = \{p \mid p \in \mathbb{R}^n, a^\top p \leq b\}.$$

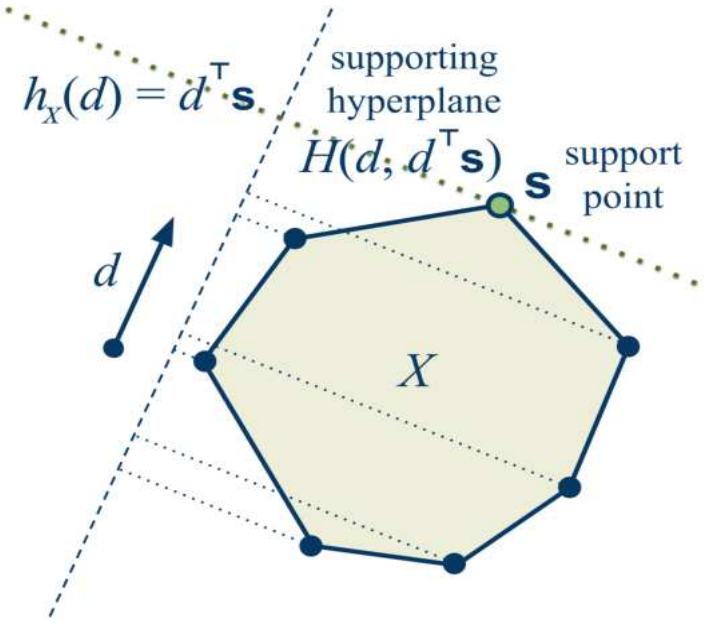
A trivial, but important, property is that $H^+(-a, -b) = H^-(a, b)$. To define more formally the concept of a supporting hyperplane, $H(a, b)$ supports X at x iff:

$$\Big(x \in X \cap H\Big) \wedge \Bigg(\Big(X \subseteq H^+(a, b)\Big) \vee \Big(X \subseteq H^-(a, b)\Big)\Bigg).$$

Given a vector $d \in \mathbb{R}^n$, a support point of X in the direction d is a point $\mathbf{s} \in X$ furthest in the direction d. More formally:

$$d^\top \mathbf{s} = \max\{d^\top x \mid x \in X\}.$$

Figure 4 – Visualization of the concepts of support



Another way of thinking about a support point is by projecting all points of X in the axis that is passing through the vector d and taking a point of X that is furthest in that axis. The greater the inner product is, the further the projection is in the d's axis. A support point $\mathbf{s}$ in the direction d means that all points of $x \in X$ are not further than $\mathbf{s}$ in the direction d, which means that, for all $x \in X$:

$$d^\top x \leq d^\top \mathbf{s}.$$

The name support point comes from the fact that $H(d, d^\top \mathbf{s})$ is a supporting hyperplane since $\mathbf{s} \in X \cap H(d, d^\top \mathbf{s})$ and $X \subset H^-(d, d^\top \mathbf{s})$, which is an immediate result from the inequality above.

On top of that, we define the support function $h_X(d) : \mathbb{R}^n \to \mathbb{R}$ by:

$$h_X(d) = \max\{d^\top x \mid x \in X\}.$$

Equivalently, we can define the support point $\mathbf{s}$ as a point that satisfies:

$$d^\top \mathbf{s} = h_X(d).$$

As a consequence, $H(d, h_X(d))$ is a supporting hyperplane as well as $X \subset H^-(d, h_X(d))$. An intuitive interpretation of the support function is that $\frac{h_X(d)}{\|d\|}$ can be viewed as the signed Euclidean distance from its supporting hyperplane $H(d, h_X(d))$ to the origin (see lemma 2.6).

With all those concepts at hand, we can define the support mapping $s_X(d)$ to return a support point of X in the direction d. Although there might be more than one support point given a region and a direction, one can define the support mapping so it picks a pre-selected random support point to guarantee the univalent property. From this definition, we can roughly define the support mapping as:

$$s_X(d) \in \operatorname*{argmax}_{x \in X} d^\top x.$$

Theorem 2.3. *(Support Computability) For all sets of points $P \subset \mathbb{R}^n$ and directions $d \in \mathbb{R}^n$:*

$$h_P(d) = h_{\text{conv}(P)}(d).$$

Proof. There are only three possibilities:

1. $h_P(d) > h_{\text{conv}(P)}(d)$,
2. $h_P(d) = h_{\text{conv}(P)}(d)$,
3. $h_P(d) < h_{\text{conv}(P)}(d)$.

The first one is impossible because $P \subset \text{conv}(P)$, so, if true, there would be a point $p \in P$, where $d^\top p > h_{\text{conv}(P)}(d)$, but also $p \in \text{conv}(P)$ since $P \subset \text{conv}(P)$, which would be a contradiction. The last one is also impossible. To prove it, assume $\mathbf{s}$ to be a support point of $\text{conv}(P)$ in the direction d, i.e.:

$$d^\top \mathbf{s} = h_{\text{conv}(P)}(d).$$

Since $\mathbf{s}$ is a point of $\text{conv}(P)$, $\mathbf{s}$ must be a convex combination of points in P, i.e.:

$$\mathbf{s} = \sum_{p \in P} \lambda_p p$$

$$\sum_{p \in P} \lambda_p = 1$$
$$\forall p \in P \, (0 \leq \lambda_p) \, .$$

Expanding $\mathbf{s}$ in $d^\top \mathbf{s}$, we get:

$$d^\top \mathbf{s} = \sum_{p \in P} \lambda_p d^\top p.$$

Since $d^\top \mathbf{s} = h_{\text{conv}(P)}(d)$ and we assumed that $h_P(d) < h_{\text{conv}(P)}(d)$, for all points $p \in P$:

$$d^\top p < d^\top \mathbf{s},$$

therefore:

$$\sum_{p \in P} \lambda_p d^\top p < \sum_{p \in P} \lambda_p d^\top \mathbf{s}.$$

Because there is a $\lambda_p \neq 0$, we can keep the strict inequality "$<$" above. Expanding the expression above:

$$d^\top \mathbf{s} = \sum_{p \in P} \lambda_p d^\top p < \sum_{p \in P} \lambda_p d^\top \mathbf{s} = d^\top \mathbf{s} \left(\sum_{p \in P} \lambda_p \right) = d^\top \mathbf{s},$$

in conclusion:

$$d^\top \mathbf{s} < d^\top \mathbf{s},$$

which is a contradiction, therefore, the only left possibility is:

$$h_P(d) = h_{\text{conv}(P)}(d).$$

□

The property above is the (17) from (GILBERT; JOHNSON; KEERTHI, 1988) whose prove has been omitted. This allows us to compute a support point of the convex hull of P in the direction d by searching for a point $p \in P$ that gives the maximum $d^\top p$, which shows an explicit construction of a support mapping:

$$s_P(d) = s_{\text{conv}(P)}(d).$$

Lemma 2.4. *(Maximum of sum)*

$$\max A \oplus B = \max A + \max B.$$

Proof. Assume $a + b = \max A \oplus B$, thus:

$$\max A + \max B \leq a + b.$$

By definition, $a \leq \max A$ and $b \leq \max B$, hence:

$$a + b \leq \max A + b \leq \max A + \max B$$

by transitivity, we have:

$$\max A + \max B \leq a + b \leq \max A + \max B.$$

Therefore:

$$\max A \oplus B = a + b = \max A + \max B.$$

□

Theorem 2.5. *(Support Mapping of the Minkowski Difference)*

$$s_{A \ominus B}(d) = s_A(d) - s_B(-d).$$

Proof. Let $\mathbf{s}_m = s_{A \ominus B}(d)$. By the definition of support function:

$$d^\top \mathbf{s}_m = h_{A \ominus B}(d) = \max\{d^\top m \mid m \in A \ominus B\},$$

which is the same as:

$$d^\top \mathbf{s}_m = \max\{d^\top (a - b) \mid a \in A, b \in B\}$$

$$d^\top \mathbf{s}_m = \max\{d^\top a - d^\top b \mid a \in A, b \in B\}$$

$$d^\top \mathbf{s}_m = \max\left(\{d^\top a \mid a \in A\} \oplus \{-d^\top b \mid b \in B\}\right).$$

Using lemma 2.4, we conclude:

$$d^\top \mathbf{s}_m = \max\{d^\top a \mid a \in A\} + \max\{-d^\top b \mid b \in B\}.$$

From those two maxima, let $\mathbf{s}_a = s_A(d)$ and $\mathbf{s}_b = s_B(-d)$. By the definition of support points:

$$d^\top \mathbf{s}_a = \max\{d^\top a \mid a \in A\}$$

$$-d^\top \mathbf{s}_b = \max\{-d^\top b \mid b \in B\}.$$

Applying substitution on the last $d^\top \mathbf{s}_m$ equality, we get:

$$d^\top \mathbf{s}_m = d^\top \mathbf{s}_a - d^\top \mathbf{s}_b$$

$$d^\top \mathbf{s}_m = d^\top (\mathbf{s}_a - \mathbf{s}_b).$$

As result, we conclude that $\mathbf{s}_a - \mathbf{s}_b$ is a support point since, by substitution:

$$d^\top (\mathbf{s}_a - \mathbf{s}_b) = d^\top \mathbf{s}_m = h_{A \ominus B}(d),$$

therefore, we can compute the support function of the Minkowski difference by doing:

$$s_{A \ominus B}(d) = s_A(d) - s_B(-d).$$

□

The property above is (21) from (GILBERT; JOHNSON; KEERTHI, 1988) whose prove has been omitted as well. This is one of the most important results in the GJK algorithm because it allows us to find a support point without explicitly computing the Minkowsky difference.

Lemma 2.6. *For all $a \in \mathbb{R}^n$ and for all $b \in \mathbb{R}_{\geq 0}$:*

$$v(H^+(a,b)) = \begin{cases} \frac{b}{a^\top a}a, & \text{if } 0 < b \\ \mathbf{0}, & \text{otherwise} \end{cases}$$

Proof. If $b \leq 0$, then $a^\top \mathbf{0} = b$, therefore $\mathbf{0} \in H^+(a,b)$. If $0 < b$, then $v(H^+(a,b))$ must be in the boundary of $H^+(a,b)$, which is $H(a,b)$. Therefore:

$$v(H^+(a,b)) = v(H(a,b)).$$

It is easy to verify that:

$$\frac{b}{a^\top a}a \in H(a,b).$$

Since this vector is also orthogonal to $H(a,b)$:

$$\frac{b}{a^\top a}a = v(H(a,b)) = v(H^+(a,b)).$$

□

Definition 2.5. (Simplex)

A simplex is the convex hull of an affine basis. More formally, a region $S_k \subset \mathbb{R}^n$ is a k-simplex iff there exist a set of $k+1$ affinely independent points $P \subset \mathbb{R}^n$ such that:

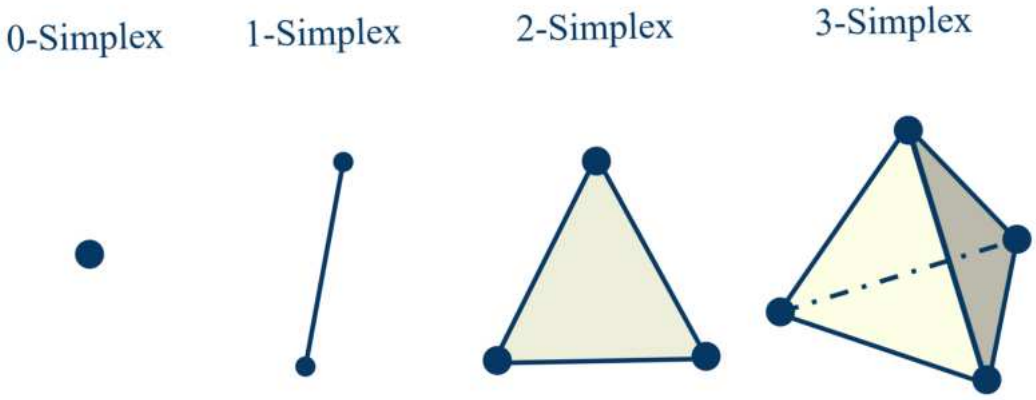
$$S_k = \text{conv}(P).$$

If $\text{conv}(P)$ is a simplex and P is a finite set of affinely independent points, then P is called a simplex set.

Another way of thinking of a simplex is as the generalization of a point, a line, a triangle, and a tetrahedron, but for any number of dimensions. Roughly speaking, a k-simplex is a k-dimension polytope with the least amount of vertices ($k+1$ vertices to be precise), so a 0-simplex is a point, a 1-simplex is a line, a 2-simplex is a triangle, a 3-simplex is a tetrahedron, and so on.

With all those essential concepts and properties explained, we are ready to fully understand the GJK algorithm as well as the next chapters.

Figure 5 – Visualization of simplices



3 THE GILBERT-JOHNSON-KEERTHI ALGORITHM

Given two finite sets of points $A, B \subset \mathbb{R}^n$, the Gilbert-Johnson-Keerthi algorithm (GJK for short) computes the distance vector to the convex hull of A from the convex hull of B. The main strategy of the GJK algorithm is to find the $v(\text{conv}(A \ominus B))$ by constructing a sequence of simplices where, for any simplex in the sequence, the next one is constructed using a support point in a way that it is strictly closer to the origin.

3.1 A BRIEF DESCRIPTION OF THE ALGORITHM

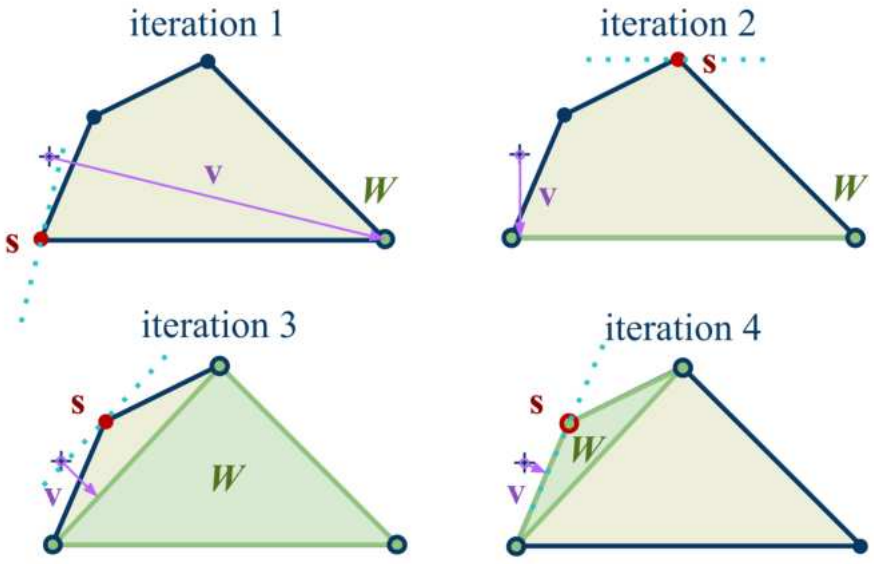
As input, we have a set of points $A, B \subset \mathbb{R}^n$ representing the convex polytopes $\text{conv}(A), \text{conv}(B)$. The algorithm starts by picking any simplex set $W_0 \subset \mathbb{R}^n$, i.e., any finite set of affinely independent points representing the simplex $\text{conv}(W_0)$. It also assigns $\mathbf{v}_0 = v(W_0)$ to be the point of the simplex nearest to the origin.

While the current distance vector is not the zero vector, that is, $\mathbf{v}_i \neq \mathbf{0}$ (first halt condition), it proceeds by computing the support point towards the origin from $\mathbf{v}_i$, that is, $\mathbf{s}_i = s_{A \ominus B}(-\mathbf{v}_i)$. Using theorem 2.5, the computation of the support point can be done by $\mathbf{s}_i = s_A(-\mathbf{v}_i) - s_B(\mathbf{v}_i)$.

If $\mathbf{v}_i^\top(\mathbf{v}_i - \mathbf{s}_i) = 0$ (second halt condition), then $\mathbf{v}_i$ is a proven distance vector and we break the while-loop. Otherwise, proceed to call the distance sub-algorithm in the simplex set $W_i \cup \{s_i\}$ to calculate the new distance vector $\mathbf{v}_{i+1} = v(\text{conv}(W_i \cup \{s_i\}))$ and the smallest simplex set $W_{i+1} \subseteq W_i \cup \{s_i\}$ such that $\mathbf{v}_{i+1} \in \text{conv}(W_{i+1})$, and then we go back to the while-loop above.

After the while-loop, we return the proven distance vector $\mathbf{v}_k$.

Figure 6 – Visualization of the GJK algorithm



Before giving a complete description of the GJK algorithm, there are some important properties for the correctness and termination proof.

3.2 IMPORTANT PROPERTIES

Here is an alternative proof of theorem 1 in (GILBERT; JOHNSON; KEERTHI, 1988):

Theorem 3.1 (GJK's second halt condition)**.** *For any convex region $X \subset \mathbb{R}^n$ and $x \in X$:*

1. *if $0 < x^\top(x - s_X(-x))$, there is a vector y in the line segment $\text{conv}\{x, s_X(-x)\}$ where $\|y\| < \|x\|$.*
2. *$x^\top(x - s_X(-x)) = 0$ iff $x = v(X)$.*
3. *$\|x - v(X)\|^2 \leq x^\top(x - s_X(-x))$.*

Proof. By the definition of support points, for all points $y \in X$:

$$(-x)^\top y \leq (-x)^\top s_X(-x),$$

therefore:

$$(-x)^\top x \leq (-x)^\top s_X(-x)$$
$$0 \leq x^\top(x - s_X(-x)),$$

so there are two possible cases

case 1: $0 = x^\top(x - s_X(-x))$,

case 2: $0 < x^\top(x - s_X(-x))$.

First, suppose the first case is true, therefore:

$$(-x)^\top x = (-x)^\top s_X(-x).$$

The equality above means that x is a support point of X in the direction $-x$, which means that:

$$X \subset H^-(-x, -x^\top x),$$

equivalently:

$$X \subset H^+(x, x^\top x).$$

By using lemma 2.6, we conclude that:

$$v(H^+(x, x^\top x)) = x.$$

Since $X \subset H^+(x, x^\top x)$:

$$v(X) = x,$$

which proves that (2.): $x^\top(x - s_X(-x)) = 0 \implies x = v(X)$.

Second, suppose that the second case is true. By multiplying both sides by -2:

$$2x^\top(s_X(-x) - x) < 0$$

Suppose $y \in \mathbb{R}^n$ is a point in the line segment between $\text{conv}\{x, s_X(-x)\}$, that is:

$$y = (1 - \lambda)x + \lambda s_X(-x)$$

$$0 \leq \lambda \leq 1,$$

so $y \in X$ as X is convex. The derivative of $y^\top y$ with respect to λ is:

$$\partial_\lambda y^\top y = 2y^\top \partial_\lambda y,$$

so, when $\lambda = 0$:

$$\partial_\lambda y^\top y = 2x^\top(s_X(-x) - x) < 0.$$

Since $y = x$ when $\lambda = 0$, from the definition of derivative, there is a $\lambda > 0$ where $y^\top y < x^\top x$, proving (1.). As a consequence, if $x = v(X)$, then the first case is the only one actually possible:

$$0 = y^\top y + h_X(-y),$$

proving (2.): $x^\top(x - s_X(-x)) = 0 \impliedby x = v(X)$.

Third, suppose $\mathbf{v} = v(X)$. By the already proven (2.):

$$\mathbf{v}^\top(\mathbf{v} - s_X(-\mathbf{v})) = 0$$

$$\mathbf{v}^\top\mathbf{v} + (-\mathbf{v})^\top s_X(-\mathbf{v}) = 0$$

$$(-\mathbf{v})^\top\mathbf{v} = (-\mathbf{v})^\top s_X(-\mathbf{v})$$

By the definition of support points, for all points $x \in X$:

$$(-\mathbf{v})^\top x \leq (-\mathbf{v})^\top s_X(-\mathbf{v})$$

substituting $(-\mathbf{v})^\top\mathbf{v} = (-\mathbf{v})^\top s_X(-\mathbf{v})$:

$$(-\mathbf{v})^\top x \leq (-\mathbf{v})^\top\mathbf{v}$$

$$(-\mathbf{v})^\top x + \mathbf{v}^\top\mathbf{v} \leq 0$$

adding both sides by $x^\top x - \mathbf{v}^\top x$:

$$x^\top x - 2\mathbf{v}^\top x + \mathbf{v}^\top\mathbf{v} \leq x^\top x - \mathbf{v}^\top x$$

$$\|x - \mathbf{v}\|^2 \leq x^\top x - \mathbf{v}^\top x.$$

Again, by the definition of support points:

$$(-x)^\top \mathbf{v} \leq (-x)^\top s_X(-x)$$

adding $x^\top x$ both sides:

$$x^\top x - x^\top \mathbf{v} \leq x^\top x - x^\top s_X(-x)$$

$$x^\top x - \mathbf{v}^\top x \leq x^\top (x - s_X(-x)).$$

Therefore:

$$\|x - \mathbf{v}\|^2 \leq x^\top x - \mathbf{v}^\top x \leq x^\top (x - s_X(-x)),$$

finally proving (3.).

□

3.3 PSEUDO-CODE, CORRECTNESS AND TERMINATION PROOF

The description of the GJK algorithm presented in (GILBERT; JOHNSON; KEERTHI, 1988) can be defined, based on a pseudo-code from (BERGEN, 1999), as:

Algorithm 1: GJK algorithm

```
Data: A, B ⊂ ℝ^n
Result: ‖v(conv(A ⊖ B))‖
W ← "arbitrary simplex set from A ⊖ B"
v ← "precomputed v(conv(W))"
while v ≠ 0 do
    s ← s_A(−v) − s_B(v)
    if v^⊤(v − s) = 0 then
        break
    end
    (W, v) ← distanceSubalgorithm(W ∪ {s})
end
return v
```

Given a finite set of affinely independent points Y representing a simplex, the distance sub-algorithm computes $v(\text{conv}(Y))$ as well as the smallest set $X \subseteq Y$ such that $v(\text{conv}(Y)) \in \text{conv}(X)$. There are three different distance sub-algorithms such as Johnson's distance sub-algorithm (GILBERT; JOHNSON; KEERTHI, 1988), Ericson's method (ERICSON, 2004) also called Voronoi search, and the Signed Volumes method (MONTANARI; PETRINIC, 2016). In the next chapter, only Johnson's algorithm will be covered since it is the one used in the original GJK algorithm.

Before proving the correctness and the termination proof of the GJK algorithm, here are some notations and useful properties:

1. For any two finite sets of points $A, B \subset \mathbb{R}^n$, let $X = A \ominus B$ represent the Minkowsky difference.

2. Let W_i be the sequence of simplex sets assigned, in order, to the variable W, where W_0 is the first value assigned to W, that is, an arbitrary simplex set:

$$W_0 \subseteq X.$$

3. Let $\mathbf{v}_i$ be the sequence of current distance vectors assigned, in order, to the variable $\mathbf{v}$, where $\mathbf{v}_0$ is assigned to be $v(\text{conv}(W_0))$.

4. Let $\mathbf{s}_i$ be the sequence of support points assigned, in order, to the variable $\mathbf{s}$. The only line of code where the variable $\mathbf{s}$ is assigned implies that, for all $\mathbf{s}_i$ of the sequence:

$$\mathbf{s}_i = s_X(-\mathbf{v}_i).$$

5. The distance sub-algorithm guarantees that $\mathbf{v}_i = v(\text{conv}(W_i \cup \{\mathbf{s}_i\}))$ as well as it guarantees the simplex set $W_{i+1} \subseteq W_i \cup \{\mathbf{s}_i\}$ is the minimum subset such that the simplex $\text{conv}(W_{i+1})$ contains $\mathbf{v}_i$. More formally:

$$W_{i+1} \subseteq W_i \cup \{\mathbf{s}_i\}$$

$$v(\text{conv}(W_i \cup \{\mathbf{s}_i\})) \in \text{conv}(W_{i+1}).$$

   Therefore, for all $\mathbf{v}_i$ of the sequence:

$$\mathbf{v}_i = v(\text{conv}(W_i)).$$

6. In the GJK algorithm, the first simplex set W_0 is an arbitrary one such that:

$$W_0 \subseteq X.$$

   By the distance sub-algorithm, for all W_i of the sequence:

$$W_{i+1} \subseteq W_i \cup \{\mathbf{s}\}.$$

   Therefore, one can conclude that:

$$W_i \subseteq X.$$

Theorem 3.2 (Correctness of the GJK algorithm)**.** *For any two finite sets of points $A, B \subset \mathbb{R}^n$ as inputs, the GJK algorithm returns $v(\text{conv}(A \ominus B))$.*

Proof. The idea is to prove that $\mathbf{v} = v(\text{conv}(X))$ after the while-loop. The final value of $\mathbf{v}$ comes from the last assigned value before the first or second halt conditions.

For the first halt condition, if $\mathbf{v}_i = \mathbf{0}$, then the while-loop breaks, and the zero vector is returned as the distance vector. From the properties above, $\mathbf{v}_i = v(\text{conv}(W_i))$ and $W_i \subseteq X$, hence $\mathbf{v}_i$ is also a point of $\text{conv}(X)$. If $\mathbf{v}_i = \mathbf{0}$, then $\mathbf{0} \in \text{conv}(X)$, therefore, $v(\text{conv}(X)) = \mathbf{0}$, and the algorithm returns a proven distance vector.

For the second halt condition, if $\mathbf{v}_i^\top(\mathbf{v}_i - \mathbf{s}_i) = 0$, then $\mathbf{v}_i = v(\text{conv}(X))$ by theorem 3.1, the while-loop breaks, and the algorithm returns a proven distance vector.

□

As those are the only two halt conditions possible for the GJK algorithm, if the GJK algorithm halts for some inputs, the output is proven to be correct. The next problem is to prove that the GJK algorithm always halts for any input.

Theorem 3.3 (Termination proof of the GJK algorithm)**.** *For any two finite sets of points $A, B \subset \mathbb{R}^n$ as inputs, the GJK algorithm always halts.*

Proof. In the i-step, suppose the GJK algorithm didn't halt, hence $\mathbf{v}_i^\top(\mathbf{v}_i - \mathbf{s}_i) \neq 0$. Substituting $\mathbf{s}_i = s_X(-\mathbf{v}_i)$:

$$\mathbf{v}_i^\top(\mathbf{v}_i - s_X(-\mathbf{v}_i)) \neq 0$$

By definition, $(-\mathbf{v}_i)^\top x \leq (-\mathbf{v}_i)^\top \mathbf{s}_i$ for all $x \in X$. Because $W_i \cup \{\mathbf{s}_i\} \subseteq X$, then $\mathbf{s}_i$ is also a support point of $W_i \cup \{\mathbf{s}_i\}$ in the direction $-\mathbf{v}_i$, therefore:

$$s_X(-\mathbf{v}_i) = s_{W_i \cup \{\mathbf{s}_i\}}(-\mathbf{v}_i)$$

and, by substitution:

$$\mathbf{v}_i^\top(\mathbf{v}_i - s_{W_i \cup \{\mathbf{s}_i\}}(-\mathbf{v}_i)) \neq 0$$

by the support computability theorem 2.3:

$$\mathbf{v}_i^\top(\mathbf{v}_i - s_{\text{conv}(W_i \cup \{\mathbf{s}_i\})}(-\mathbf{v}_i)) \neq 0$$

As consequence of theorem 3.1, $\mathbf{v}_i \neq v(\text{conv}(W_i \cup \{\mathbf{s}_i\}))$, which means that $v(\text{conv}(W_i \cup \{\mathbf{s}_i\}))$ is strictly closer to the origin than $\mathbf{v}_i$. From the definition of the distance sub-algorithm, $\mathbf{v}_{i+1} = v(\text{conv}(W_i \cup \{\mathbf{s}_i\})) = v(\text{conv}(W_{i+1}))$. Therefore, for all elements $\mathbf{v}_i$ of the sequence:

$$\|\mathbf{v}_i\| < \|\mathbf{v}_{i+1}\|.$$

Let d_i be the sequence defined as:

$$d_i = \|\mathbf{v}_i\|,$$

which is, by the property above, a strictly decreasing sequence. Suppose by contradiction that $W_i = W_j$ for some $i \neq j$. Thus:

$$v(\text{conv}(W_i)) = v(\text{conv}(W_j))$$

$$\|v(\text{conv}(W_i))\| = \|v(\text{conv}(W_j))\|$$

$$d_i = d_j,$$

which is a contradiction. Therefore, the sequence W_i has distinct elements. the set X is finite by definition, hence the power set $\mathcal{P}(X)$ is finite. Since the sequence W_i has distinct elements that are also subsets from X, the sequence W_i is finite as well, so the algorithm always halts.

□

To prove the correctness of the GJK algorithm completely, the next chapter covers Johnson's distance sub-algorithm, its correctness, and termination proof.

4 DISTANCE SUB-ALGORITHM

As mentioned before, given a simplex set Y, the distance sub-algorithm computes $v(\text{conv}(Y))$ and the smallest subset $X \subseteq Y$ such that the minimum norm vector $v(\text{conv}(Y))$ is also a vector of the simplex $\text{conv}(X)$. To understand better what is this smallest subset, let $I = \{1, \ldots, n\}$ be the index set, $Y = \{y_1, \ldots, y_n\} \subset \mathbb{R}^r$, and $\mathbf{v} = v(\text{conv}(Y))$. Because $\mathbf{v} \in \text{conv}(Y)$, this minimum norm vector can be expressed by its barycentric coordinates:

$$\mathbf{v} = \sum_{i \in I} \lambda_i y_i$$

where

$$\sum_{i \in I} \lambda_i = 1$$
$$\forall i \in I (0 \leq \lambda_i).$$

If there is any coordinate $\lambda_i = 0$, then the point y_i can be safely removed from the summation, thus it is possible to remove all points of zero coordinates, that is, to select only the points with positive coordinates. From this idea, let $I_+ \subseteq I$ be the index subset defined as:

$$I_+ = \{i \mid i \in I, 0 < \lambda_i\}.$$

In other words, I_+ is the index subset of all positive coordinates. Also, let $Y_+ = \{y_i \mid i \in I_+\}$, therefore:

$$\mathbf{v} = \sum_{i \in I_+} \lambda_i y_i$$

where

$$\sum_{i \in I_+} \lambda_i = 1$$
$$\forall i \in I_+ (0 < \lambda_i).$$

Since the points of Y are affinely independent, the coordinates of $\mathbf{v}$ are unique, therefore Y_+ is the smallest subset of Y where $\mathbf{v} \in \text{conv}(Y_+)$. Let $\mathbf{v}_+ = v(\text{conv}(Y_+))$, so another property is that $\mathbf{v} = \mathbf{v}_+$. The proof follows:

$\mathbf{v} \in \text{conv}(Y_+)$, thus $\|\mathbf{v}_+\| \leq \|\mathbf{v}\|$. $\mathbf{v}_+ \in \text{conv}(Y_+) \subset \text{conv}\, Y$, thus $\|\mathbf{v}\| \leq \|\mathbf{v}_+\|$. Therefore:

$$\|\mathbf{v}\| = \|\mathbf{v}_+\|.$$

By theorem 2.1, $\mathbf{v} = \mathbf{v}_+$.

From the result above, since all coordinates of $\mathbf{v}$ are positive, $\mathbf{v}_+$ shares the same property, hence, by theorem 2.2, $\mathbf{v}_+ = v(\text{aff}(Y_+))$.

4.1 INTRODUCTION TO JOHNSON'S DISTANCE SUB-ALGORITHM

The first proposal for the distance sub-algorithm is one made by Daniel W. Johnson described in (GILBERT; JOHNSON; KEERTHI, 1988). This algorithm uses dynamic programming on top of a recursive formula based on Cramer's rule and Laplace expansion to compute the barycentric coordinates of $v(\text{aff}(Y_s))$, traversing each non-empty subset $Y_s \subseteq Y$ until $v(\text{conv}(Y)) = v(\text{aff}(Y_s))$ is satisfied. In order to better understand the idea behind this algorithm, let Y_s be any non-empty subset of Y where $Y_s = \{y_1, \ldots, y_n\}, n \leq \#Y$ and let $x_1, \ldots, x_n$ be any ordering of Y_s elements. Any point $y \in \text{aff}(Y_s)$ can be modeled as:

$$y = \sum_{i \in 1}^{n} \lambda_i x_i$$

where

$$\sum_{i \in 1}^{n} \lambda_i = 1.$$

This approach constrains the coordinates λ_i, so another way of modeling y with unconstrained coordinates is:

$$\lambda_1 = 1 - \sum_{i \in 2}^{n} \lambda_i$$

and then:

$$y = x_1 + \sum_{i \in 2}^{n} \lambda_i (x_i - x_1)$$

where $\lambda_i, 2 \leq i \leq n$ can be any real number. In order to compute $v(\text{aff}(Y_s))$, we must find the coordinates $\lambda_i, 2 \leq i \leq n$ that minimizes the function $y^\top y$, which is equivalent of solving the coordinates where the gradient is zero, that is, for all $2 \leq j \leq n$:

$$\partial_{\lambda_j}(y^\top y) = 2y^\top \partial_{\lambda_j}(y) = 0$$

$$y^\top \partial_{\lambda_j}(y) = 0$$

substituting y by its affine combination and $\partial_{\lambda_j}(y) = (x_j - x_1)$:

$$\sum_{i=1}^{n} \lambda_i x_i^\top (x_j - x_1) = 0.$$

The result above can be modeled as a linear system $A_s \lambda = \mathbf{b}$, that is:

$$\begin{bmatrix} 1 & \ldots & 1 \\ (x_2 - x_1)^\top x_1 & \ldots & (x_2 - x_1)^\top x_n \\ \vdots & \ddots & \vdots \\ (x_n - x_1)^\top x_1 & \ldots & (x_n - x_1)^\top x_n \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Using Cramer's rule, we get:

$$\lambda_j = \frac{\det A_s^j}{\det A_s}$$

where A_s^j is the matrix A_s with the j-th column substituted by $\mathbf{b}$. Using Laplace expansion of A_s^j on the first row, we get:

$$\det A_s^j = (-1)^{1+j} \det A_s^{1,j}$$

where $A_s^{1,j}$ is the matrix A_s with the first row and the j-th column removed. Combining both results, one can easily verify that:

$$\lambda_j = \frac{(-1)^{1+j} \det A_s^{1,j}}{\det A_s}.$$

Before describing how Johnson's method computes the coordinates recursively, here are some important properties of $\det A_s$:

1. $\det A_s$ is invariant under the choice of $x_1, \ldots, x_n \in Y_s$. To prove this:

   Suppose we replace x_i with x_j where $2 \leq i \leq j \leq n$, that is, keeping x_1 in $(x_i - x_1)^\top x_j$. Since the determinant is anti-symmetric, one can easily swap the i-th row with the j-th row and the i-th column with the j-th columns to restore the original matrix. Since we only swapped rows or columns two times, we flipped the determinant signal two times, which does not change $\det A_s$.

   Now, suppose we want to replace x_1 with x_k where $2 \leq k \leq n$, that is, we want to change the terms $(x_i - x_1)^\top x_j$ to $(x_i - x_k)^\top x_j$. To restore the original matrix, we can take the k-th row, which is:

   $$\begin{bmatrix}(x_k - x_1)^\top x_1 & \ldots & (x_k - x_1)^\top x_n\end{bmatrix}$$

   and subtract all rows of the matrix by the k-th row, except for the first row and the k-row, so all the terms where $i \neq 1, i \neq k$ will be $(x_i - x_k)^\top x_j$ as intended. Here, the determinant is still invariant because of its alternating nature. After that, we first multiply the k-th row by -1, and then we swap the second row by the k-row. By doing it, we again flip the signal two times, which does not change $\det A_s$.

2. $\det A_s$ is strictly positive. Suppose that, for every $2 \leq i \leq n$, we subtract the i-th row by the 1-row multiplied by $(x_i - x_1)^\top x_1$, that is:

   $$\begin{bmatrix}(x_i - x_1)^\top x_1 & \ldots & (x_i - x_1)^\top x_n\end{bmatrix} - \Big((x_i - x_1)^\top x_1\Big) \begin{bmatrix}1 & \ldots & 1\end{bmatrix}.$$

   This will give a modified matrix A'_s whose terms are $A'_s[i,j] = (x_i - x_1)^\top (x_j - x_1)$, that is:

   $$A'_s = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ 0 & (x_2 - x_1)^\top (x_2 - x_1) & \ldots & (x_2 - x_1)^\top x_n \\ \vdots & \vdots & \ddots & \vdots \\ 0 & (x_n - x_1)^\top (x_n - x_1) & \ldots & (x_n - x_1)^\top x_n \end{bmatrix}.$$

From there, the determinant is still the same by its alternating property. Let G_s to be the matrix:

$$G_s = \begin{bmatrix} | & & | \\ (x_2 - x_1) & \dots & (x_n - x_1) \\ | & & | \end{bmatrix}.$$

Therefore, by the Laplace expansion of A'_s on the first row, $\det A_s = \det A'_s = \det(G_s^\top G_s)$. Because $G_s^\top G_s$ is a Gram matrix, $0 < \det(G_s^\top G_s)$ iff the columns of G_s are linear independent, proving that $0 < \det A_s$.

By the property (1.), the coordinates λ_j are indeed invariant under the choice of $x_1, \dots, x_n \in Y_s$. From this idea, define $\Delta_i(Y_s)$ to be:

$$\Delta_i(Y_s) = (-1)^{1+j} \det A_s^{1,j}$$

where $x_j = y_i$. In other words, $\Delta_i(Y_s)$ is the cofactor $C_{1,j}$ of the matrix A_s. Again, by Laplace expansion:

$$\det A_s = \sum_{j=1}^{n} C_{1,j} = \sum_{i \in I_s} \Delta_i(Y_s).$$

From this idea, define $\Delta(Y_s)$ to be:

$$\Delta(Y_s) = \sum_{i \in I_s} \Delta_i(Y_s).$$

As consequence, the coordinates $\lambda_j, 1 \leq j \leq n$ can be expressed as:

$$\lambda_j = \frac{\Delta_i(Y_s)}{\Delta(Y_s)}$$

satisfying $x_j = y_i$. By the property (2.), $0 < \Delta(Y_s)$, thus it is easy to show that:

$$\operatorname{sign}(\lambda_j) = \operatorname{sign}(\Delta_i(Y_s))$$

where:

$$\operatorname{sign}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \\ -1, & \text{if } x < 0 \end{cases}$$

This property is extremely important to remember because the concept of convexity can be evaluated by this. In other words, $v(\operatorname{aff}(Y_s)) \in \operatorname{conv}(Y_s)$ iff $0 \leq \lambda_j$ for all $1 \leq j \leq n$, which is equivalent of $0 \leq \Delta_i(Y_s)$ for all $i \in I_s$.

The recursion of Johnson's distance sub-algorithm tells us that:

$$\Delta_j(Y_s \cup \{y_j\}) = \sum_{i \in I_s} \Delta_i(Y_s)(y_k - y_j)^\top y_i$$

for any $k \in I_s$ and any $j \notin I_s$. In advance, let $Y_{s+} = Y_s \cup \{y_j\}$. To understand how this recursion is correct, suppose we want to add $x_{n+1} = y_j$ into the matrix A_s to construct A_{s+}, that is:

$$A_{s+} = \begin{bmatrix} 1 & \dots & 1 \\ (x_2 - x_1)^\top x_1 & \dots & (x_2 - x_1)^\top x_{n+1} \\ \vdots & \ddots & \vdots \\ (x_{n+1} - x_1)^\top x_1 & \dots & (x_{n+1} - x_1)^\top x_{n+1} \end{bmatrix}.$$

By definition, we have that:

$$\Delta_j(Y_s \cup \{y_j\}) = (-1)^{1+(n+1)} \det A_{s+}^{1,(n+1)} = (-1)^n \det A_{s+}^{1,(n+1)}$$

By Laplace expansion of $A_{s+}^{1,(n+1)}$ on the last row, one can verify that:

$$\det A_{s+}^{1,(n+1)} = \sum_{i=1}^{n} (-1)^{n+i} \Big((x_{n+1} - x_1)^\top x_i \Big) \det A_s^{1,i}.$$

Therefore:

$$\Delta_j(Y_s \cup \{y_j\}) = (-1)^n \det A_{s+}^{1,(n+1)}$$

$$\Delta_j(Y_s \cup \{y_j\}) = (-1)^n \sum_{i=1}^{n} (-1)^{2n+i} \Big((x_{n+1} - x_1)^\top x_i \Big) \det A_s^{1,i}$$

$$\Delta_j(Y_s \cup \{y_j\}) = \sum_{i=1}^{n} (-1)^{i} \Big((x_{n+1} - x_1)^\top x_i \Big) \det A_s^{1,i}$$

$$\Delta_j(Y_s \cup \{y_j\}) = \sum_{i=1}^{n} (-1)^{1+i} \Big((x_1 - x_{n+1})^\top x_i \Big) \det A_s^{1,i}$$

$$\Delta_j(Y_s \cup \{y_j\}) = \sum_{i \in I_s} \Delta_i(Y_s) (x_1 - x_{n+1})^\top y_i.$$

Since x_1 could be y_k for any $k \in I_s$, the recursive formula is correct.

In summary, let $I_s \subseteq I = \{1, \dots, n\}$ be any non-empty subset, let $Y_s = \{y_i \mid i \in I_s\} \subseteq Y$, and let the affine combination of $v(\text{aff}(Y_s))$ be:

$$v(\text{aff}(Y_s)) = \sum_{i \in I_s} \lambda_i y_i$$

where:

$$\sum_{i \in I_s} \lambda_i = 1.$$

The Johnson's distance sub-algorithm express the coordinates $\lambda_i, i \in I_s$ by:

$$\lambda_i = \frac{\Delta_i(Y_s)}{\Delta(Y_s)}.$$

To do so, it starts by defining, for all $i \in I_s$:

$$\Delta_i(\{y_i\}) = 1.$$

Then, it uses tabulation by doing:

$$\Delta_j(Y_s \cup \{y_j\}) = \sum_{i \in I_s} \Delta_i(Y_s)(y_k - y_j)^\top y_i,$$

to compute the coordinates $\lambda_i, i \in I_s$.

So far, we have shown how Johnson's sub-algorithm can compute $v(\text{aff}(Y_s))$ for every non-empty $Y_s \subseteq Y$. But, to verify if $v(\text{conv}(Y)) = v(\text{aff}(Y_s))$, the sub-algorithm also relies on a theorem for one of the halt conditions. Basically, during the traversal of the subsets Y_s, the sub-algorithm checks two things:

1. if, for all $i \in I_s$, $0 < \Delta_i(Y_s)$
2. if, for all $j \in I \setminus I_s$, $\Delta_j(Y_s) \leq 0$

If both conditions are met, then $Y_s = Y_+$, that is, Y_s is guaranteed to be the minimum subset where $v(\text{conv}(Y)) \in \text{conv}(Y_s)$. It is important to remember that (1.) implies that $v(\text{aff}(Y_s)) \in \text{conv}(Y)$ since both the coordinate and the Δ term have the same sign.

Before jumping to the algorithm, here is the proof of the mentioned theorem, which is an alternative version of theorem 3 in (GILBERT; JOHNSON; KEERTHI, 1988):

Theorem 4.1 (Johnson's sub-algorithm halt condition)**.** *Let $Y_s \subseteq Y$ be any subset where, for all $i \in I_s$, $0 < \Delta_i(Y_s)$.*

$$v(\text{conv}(Y)) = v(\text{aff}(Y_s))$$

iff, for all $j \in I \setminus I_s$:

$$\Delta_j(Y_s \cup \{y_j\}) \leq 0.$$

Proof. For the sake of simplicity, let $\mathbf{v} = v(\text{conv}(Y))$ and $\mathbf{v}_s = v(\text{aff}(Y_s))$. First, we prove these two auxiliary properties below:

1. For all $i \in I$, $\mathbf{v}^\top(\mathbf{v} - y_i) \leq 0$,
2. For all $k \in I_s$, $\mathbf{v}_s^\top(\mathbf{v}_s - y_k) = 0$.

To prove (1.), we use theorem 3.1 to conclude that:

$$\mathbf{v}^\top(\mathbf{v} - s_Y(-\mathbf{v})) = 0.$$

By the definition of support points, for all points $x \in \text{conv}(Y)$:

$$(-\mathbf{v})^\top x \leq (-\mathbf{v})^\top s_X(-\mathbf{v})$$

adding $\mathbf{v}^\top\mathbf{v}$ both sides, we get:

$$\mathbf{v}^\top(\mathbf{v} - x) \leq \mathbf{v}^\top(\mathbf{v} - s_Y(-\mathbf{v})) = 0.$$

Since x could be any point of $\text{conv}(Y)$ and $Y \subseteq \text{conv}(Y)$, we conclude that, for all $i \in I$:

$$\mathbf{v}^\top(\mathbf{v} - y_i) \leq 0.$$

To prove (2.), we borrow the idea from before used to build the linear system $A_s\lambda = \mathbf{b}$, which is the function:

$$y = x_1 + \sum_{i \in 2}^{n} \lambda_i(x_i - x_1),$$

where $n = \#Y_s$ and $x_1, \ldots, x_n$ is any ordering of Y_s elements. For the function $y^\top y$, let us recall that:

$$\partial_{\lambda_j} y^\top y = 2y^\top(x_j - x_1).$$

Because $\mathbf{v}_s$ is, by definition, the vector from $\text{aff}(Y_s)$ that minimizes the function $y^\top y$, we know that, for all $2 \leq j \leq n$:

$$\partial_{\lambda_j} y^\top y(\mathbf{v}_s) = 2\mathbf{v}_s^\top(x_j - x_1) = 0.$$

Therefore, for all $i, k \in I_s$:

$$\mathbf{v}_s^\top(y_i - y_k) = 0.$$

Let λ_i be the coordinates from the affine combination of $\mathbf{v}_s$, that is:

$$\mathbf{v}_s = \sum_{i \in I_s} \lambda_i y_i.$$

Therefore, for all $k \in I_s$:

$$\sum_{i \in I_s} \lambda_i 0 = 0$$

$$\sum_{i \in I_s} \lambda_i \mathbf{v}_s^\top(y_i - y_k) = 0$$

$$\mathbf{v}_s^\top \left(\sum_{i \in I_s} \lambda_i y_i \right) - \mathbf{v}_s^\top y_k \left(\sum_{i \in I_s} \lambda_i \right) = 0$$

$$\mathbf{v}_s^\top \mathbf{v}_s - \mathbf{v}_s^\top y_k = 0$$

$$\mathbf{v}_s^\top(\mathbf{v}_s - y_k) = 0.$$

Now that (1.) and (2.) are proved, we proceed by using (1.) to conclude that, for all $i \in I$:

$$\mathbf{v} = \mathbf{v}_s \Longrightarrow \mathbf{v}_s^\top(\mathbf{v}_s - y_i) \leq 0.$$

Next, it is desired to prove that, for all $i \in I$:

$$\mathbf{v} = \mathbf{v}_s \Longleftarrow \mathbf{v}_s^\top(\mathbf{v}_s - y_i) \leq 0.$$

In order to do it, assume that $\mathbf{v}_s^\top(\mathbf{v}_s - y_i) \leq 0$ for all $i \in I$. Also, let α_i be the coordinates of $s_Y(-\mathbf{v}_s)$, that is:

$$s_Y(-\mathbf{v}_s) = \sum_{i \in I} \alpha_i y_i.$$

Therefore:

$$\sum_{i\in I} \alpha_i \mathbf{v}_s^\top (\mathbf{v}_s - y_i) \leq 0$$

$$\mathbf{v}_s^\top \mathbf{v}_s \left(\sum_{i\in I} \alpha_i \right) - \mathbf{v}_s^\top \left(\sum_{i\in I} \alpha_i y_i \right) \leq 0$$

$$\mathbf{v}_s^\top \mathbf{v}_s - \mathbf{v}_s^\top s_Y(-\mathbf{v}_s) \leq 0$$

$$\mathbf{v}_s^\top (\mathbf{v}_s - s_Y(-\mathbf{v}_s)) \leq 0.$$

$\mathbf{v}_s^\top (\mathbf{v}_s - s_Y(-\mathbf{v}_s)) < 0$ is impossible due to the basic inequality $d^\top x \leq d^\top \mathbf{s}$ in the definition of support points, therefore, we get:

$$\mathbf{v}_s^\top (\mathbf{v}_s - s_Y(-\mathbf{v}_s)) = 0.$$

As $\mathbf{v}_s$ has positive coordinates, we conclude that $\mathbf{v}_s \in \text{conv}(Y_s) \subseteq \text{conv}(Y)$, which means that we can use theorem 3.1 to get:

$$\mathbf{v} = \mathbf{v}_s.$$

Combining both properties, we have that, for all $i \in I$:

$$\mathbf{v} = \mathbf{v}_s \iff \mathbf{v}_s^\top (\mathbf{v}_s - y_i) \leq 0.$$

Just by (2.), $\mathbf{v}_s^\top (\mathbf{v}_s - y_i) \leq 0$ holds by itself for $i \in I_s$, so we can narrow the index and get that, for all $j \in I \setminus I_s$:

$$\mathbf{v} = \mathbf{v}_s \iff \mathbf{v}_s^\top (\mathbf{v}_s - y_j) \leq 0.$$

If we take the inequality $\mathbf{v}_s^\top (\mathbf{v}_s - y_j) \leq 0$ and subtract both sides by (2.), then, for all $k \in I_s$ and $j \in I \setminus I_s$:

$$\mathbf{v} = \mathbf{v}_s \iff \mathbf{v}_s^\top (y_k - y_j) \leq 0.$$

Again, as in (2.), for $i \in I_s$, let λ_i be the coordinates of $\mathbf{v}_s$. Let us recall two properties of the Δ notation defined in this section:

$$\lambda_i = \frac{\Delta_i(Y_s)}{\Delta(Y_s)},$$

$$0 < \Delta(Y_s).$$

Substituting $\mathbf{v}_s$ by its affine combination in the inequality $\mathbf{v}_s^\top (y_k - y_j) \leq 0$, we get:

$$\sum_{i\in I_s} \lambda_i y_i^\top (y_k - y_j) \leq 0$$

$$\sum_{i\in I_s} \frac{\Delta_i(Y_s)}{\Delta(Y_s)} y_i^\top (y_k - y_j) \leq 0$$

multiplying both sides by $\Delta(Y_s)$:

$$\sum_{i \in I_s} \Delta(Y_s) y_i^\top (y_k - y_j) \leq 0$$

$$\Delta_j(Y_s \cup \{y_j\}) \leq 0.$$

Finally, we conclude that, for all $j \in I \setminus I_s$:

$$\mathbf{v} = \mathbf{v}_s \iff \Delta_j(Y_s \cup \{y_j\}) \leq 0$$

as desired.

□

4.2 DESCRIPTION AND PSEUDO-CODE

As it was mentioned before, Johnson's distance sub-algorithm traverses each non-empty subset $Y_s \subseteq Y$ until the conditions of theorem 4.1 are met. During the traversal, the values of $\Delta_i(Y_s)$ for all $i \in I_s$ are computed and the halt condition is checked. We label each subset with a positive integer s where its binary decomposition indicates whether an element belongs to the subset. More formally:

$$s = \sum_{i \in I_s} 2^{i-1}.$$

Note that, if the first element of Y was y_0, then the power of two in the summation above would be just 2^i. This model guarantees that s has a one-to-one correspondence to Y_s since the i-th bit is true iff $y_i \in Y_s$.

Briefly describing Johnson's distance sub-algorithm, everything happens inside a for-loop, counting a label s from 1 to 2^{n-1}, where $n = \#Y$. Inside, it computes $\Delta_j(Y_s \cup \{y_j\})$ for all $j \in I \setminus I_s$ using the recursive formula:

$$\Delta_j(Y_s \cup \{y_j\}) = \sum_{i \in I_s} \Delta_i(Y_s)(y_k - y_j)^\top y_i.$$

If $0 < \Delta_i(Y_s)$ for all $i \in I_s$ and $\Delta_j(Y_s \cup \{y_j\}) \leq 0$ for all $j \in I \setminus I_s$, then we return Y_s as the smallest set containing $v(\text{conv}(Y))$ and $\mathbf{v}$, where:

$$\mathbf{v} = \sum_{i \in I_s} \frac{\Delta_i(Y_s)}{\Delta(Y_s)} y_i = v(\text{conv}(Y)).$$

Here is a pseudo-code:

Algorithm 2: Johnson's distance sub-algorithm

Data: A simplex set $Y \subset \mathbb{R}^n$

Result: $(Y_s, \mathbf{v})$, where $Y_s = Y_+$ and $\mathbf{v} = v(\text{conv}(Y))$

for $i \in I$ **do**
  $\Delta_i(\{y_i\}) \leftarrow 1$
end
for $s \in \{1, \ldots, 2^{\#Y-1}\}$ **do**
  $k \leftarrow$ "arbitrary value from I_s"
  for $j \in I \setminus I_s$ **do**
    $\Delta_j(Y_s \cup \{y_j\}) \leftarrow \sum_{i \in I} \Delta_i(Y_s)(y_k - y_j)^\top y_i$
  end
  if $\forall i \in I\Big(0 < \Delta_i(Y_s)\Big)$ *and* $\forall j \in I \setminus I_s\Big(\Delta_j(Y_s \cup \{y_j\}) \leq 0\Big)$ **then**
    $\Delta(Y_s) \leftarrow \sum_{i \in I_s} \Delta_i(Y_s)$
    $\mathbf{v} \leftarrow \frac{1}{\Delta(Y_s)} \sum_{i \in I_s} \Delta_i(Y_s) y_i$
    return $(Y_s, \mathbf{v})$
  end
end
return error

In (GILBERT; JOHNSON; KEERTHI, 1988), they set k to be $\min I_s$. For termination proof, the for-loops ensure that the algorithm always halts for any input. For correctness proof, the algorithm just searches for every non-empty subset of Y until we get Y_+, which exists, is unique for every simplex set Y, and satisfies the halt condition from theorem 4.1 since $v(\text{aff}(Y_+)) = v(\text{conv}(Y_+))$, so all that remains is to prove that, for all $s \in \{1, \ldots, 2^{n-1}\}$, the values from $\Delta_i(Y_s)$ for all $i \in I_s$ are available when computing $\Delta_j(Y_s \cup \{y_j\})$ for all $j \in I \setminus I_s$. Here is a proof below:

Suppose the algorithm is in the s-th iteration. If $\#I_s = 1$, then $\Delta_i(Y_s) = 1$ for all $i \in I_s$, hence suppose $2 \leq \#I_s$ and let $\mathbf{i} \in I_s$ be any index, $I_r = I_s \setminus \{\mathbf{i}\}$, and $Y_r = Y_s \setminus \{y_\mathbf{i}\}$ so that, by the recursive formula, we get:

$$\Delta_\mathbf{i}(Y_s) = \Delta_\mathbf{i}(Y_r \cup \{y_\mathbf{i}\}) = \sum_{h \in I_r} \Delta_h(Y_r)(y_k - y_\mathbf{i})^\top y_h,$$

where $k \in I_r$. By the equation above, in order to have $\Delta_\mathbf{i}(Y_s)$ already computed, we must have processed Y_r before Y_s. Because $I_r \subset I_s$ has one less element, the power-of-2 summation of r is the summation of s with one less positive number. Therefore:

$$r = \sum_{i \in I_r} 2^{i-1} = \sum_{i \in I_s} 2^{i-1} - 2^\mathbf{i} < \sum_{i \in I_s} 2^{i-1} = s$$

$$r < s,$$

which implies that Y_r was processed before Y_s.

BACKUP PROCEDURE

Johnson's distance sub-algorithm is mathematically correct, but the halt condition from theorem 4.1 may never be satisfied due to numerical error, hence returning *error*. To overcome this problem, a simpler approach made by (GILBERT; JOHNSON; KEERTHI, 1988) called the Backup Procedure is used whenever the first one fails. This procedure is pretty similar to Johnson's distance sub-algorithm except that, instead of checking for the halt condition from theorem 4.1, it merely picks the vector with the least norm from $v(\text{aff}(Y_s))$ as long as its a point in $\text{conv}(Y)$. Equivalently, it searches only the minimum norm vectors $v(\text{aff}(Y_s))$ where $0 \leq \Delta_i(Y_s)$ for all $i \in I_s$, and picks the one with the least norm. Since there is a unique Y_+, this algorithm will pick $v(\text{aff}(Y_+)) = v(\text{conv}(Y))$ eventually, therefore it is correct and always halts. Here is a pseudo-code:

Algorithm 3: Backup Procedure distance sub-algorithm

Data: A simplex set $Y \subset \mathbb{R}^n$
Result: $(Y_{\mathbf{p}}, \mathbf{v})$, where $Y_{\mathbf{p}} = Y_+$ and $\mathbf{v} = v(\text{conv}(Y))$
$\mathbf{p} \leftarrow 1$
$\mathbf{v} \leftarrow y_1$
for $i \in I$ **do**
  $\Delta_i(\{y_i\}) \leftarrow 1$
end
for $s \in \{1, \ldots, 2^{\#Y-1}\}$ **do**
  $k \leftarrow$ "arbitrary value from I_s"
  for $j \in I \setminus I_s$ **do**
    $\Delta_j(Y_s \cup \{y_j\}) \leftarrow \sum_{i \in I} \Delta_i(Y_s)(y_k - y_j)^\top y_i$
  end
  if $\forall i \in I\Big(0 < \Delta_i(Y_s)\Big)$ **then**
    $\Delta(Y_s) \leftarrow \sum_{i \in I_s} \Delta_i(Y_s)$
    $\mathbf{w} \leftarrow \frac{1}{\Delta(Y_s)} \sum_{i \in I_s} \Delta_i(Y_s) y_i$
    if $\|\mathbf{w}\| < \|\mathbf{v}\|$ **then**
      $\mathbf{p} \leftarrow s$
      $\mathbf{v} \leftarrow \mathbf{w}$
    end
  end
end
return $(Y_{\mathbf{p}}, \mathbf{v})$

5 ANALYSIS, IMPROVEMENTS AND MODIFICATIONS

In all papers and books mentioned throughout this work and even in the original paper (GILBERT; JOHNSON; KEERTHI, 1988), there is no formal proof for the average and worst-time complexity of the GJK algorithm. Alternatively, numerical experiments detailed in (GILBERT; JOHNSON; KEERTHI, 1988; CAMERON, 1997) infer, on average, a linear time complexity in the total number of points from the two input sets, that is, $\#A + \#B$.

Three years after, a novel algorithm for computing the distance between two convex polyhedra, the Lin-Canny algorithm (LIN; CANNY, 1991), achieved an impressive almost-constant time complexity for scenarios with frame coherency (BERGEN, 1999), that is, scenarios where pre-computed information from the previous frame could be used to speed up computation for the next frame, e.g., two objects moving slowly. This method of using previous data to speed up computation is also called tracking, which appears in (LIN; CANNY, 1991; CAMERON, 1997; BERGEN, 1999). Because this novel algorithm is limited only to polyhedra, new improvements to GJK were proposed in order to also achieve the almost-constant time complexity.

In this chapter, we discuss the GJK time complexity, show, in detail, an improvement for the GJK in (CAMERON, 1997) that uses the *hill climbing* optimization method and a modification called Incremental Separating Axis GJK, or ISA-GJK for short, from (BERGEN, 1999). Regarding *hill climbing*, we show a counterexample where this method gives an incorrect output, and so we show an extra step to ensure this method works correctly. We also mention briefly a modification of GJK for continuous collision detection (BERGEN, 2004) and the Expanding Polytope Algorithm, or EPA for short, from (BERGEN, 2001), which computes penetration depth.

5.1 COMPLEXITY ANALYSIS

As described in Chapter 3, the GJK algorithm essentially computes a support point, checks for the second halt condition, and runs the distance sub-algorithm, all until the first halt condition is met in a while-loop. Depending on the choice for the distance sub-algorithm, the number of steps can grow exponentially when increasing the number of points of a simplex. Also, the GJK algorithm was designed to be used mainly in a 3D environment as described in (GILBERT; JOHNSON; KEERTHI, 1988). Due to these two issues, the number of dimensions of the Euclidean space is set constant.

The procedure for computing a support point is done by using theorem 2.5, which is $\mathbf{s} \leftarrow s_{A\ominus B}(-v) = s_A(-\mathbf{v}) - s_B(\mathbf{v})$. If the support mapping $s_X(d)$ is performed by testing all elements in X and selecting the candidate that maximizes $d^\top x, x \in X$, then the time

complexity for computing a support point is $\Theta(n)$, where n is the total number of points in A and B, that is, $n = \#A + \#B$. The space complexity is $O(1)$ due to the selection method. The second halt condition just checks if $\mathbf{v}^\top(\mathbf{v} - \mathbf{s}) = 0$, thus the time and space complexity is $O(1)$. The distance sub-algorithm is always $O(1)$ in both time and space complexity since the input is a simplex set of which the number of points is limited by the number of dimensions, which is set constant, plus one. In short, one iteration inside the while-loop of the GJK algorithm performs in $\Theta(n)$ time complexity and $O(1)$ space complexity. As a consequence, if numerical experiments infer $O(n)$ on average, then the average number of iterations must be $O(1)$.

At the end of Section VII in (GILBERT; JOHNSON; KEERTHI, 1988), it is mentioned, but not shown, that there is a 2D class of inputs that causes a quadratic time complexity. If such class exists, then the worst-case time complexity is $\Omega(n^2)$. Even if this class is two-dimensional, it is still a special case for higher dimensions. Since every iteration costs $\Theta(n)$, then, in the worst-case scenario, the number of iterations is $\Omega(n)$. In Chapter 4 from (BERGEN, 1999), there is an unproven assumption that the discarded support points do not reappear. If this assumption is true, the number of iterations is $O(n)$. Combining Section VII in (GILBERT; JOHNSON; KEERTHI, 1988) with Chapter 4 from (BERGEN, 1999), we have that the worst-case scenario of the GJK algorithm has $\Theta(n)$ number of iterations, that means, if using the original GJK, the worst-case scenario is quadratic.

5.2 ENHANCED GJK: HILL CLIMBING

At **3.2 Enhancing GJK** in (CAMERON, 1997), it is described a better method for computing a support point that, by using *hill climbing* to maximize $d^\top x, x \in X$, makes GJK achieve almost constant time complexity. Instead of using two sets of points $A, B \subset \mathbb{R}^k$, the *hill climbing* relies on the input being two graphs represented as adjacency lists $(A, \text{adj}_a), (B, \text{adj}_b)$ of the polytopes $\text{conv}(A), \text{conv}(B)$ respectively, where the functions $\text{adj}_a : A \to \mathcal{P}(A)$ and $\text{adj}_b : B \to \mathcal{P}(B)$ are the adjacent vertices of a given vertex. A common way of converting a set of points to its convex hull adjacency list is by using a convex hull algorithm such as the general Quickhull (BARBER; DOBKIN; HUHDANPAA, 1995), which was used in (CAMERON, 1997).

For computing a support point, the algorithm starts at the previous support point $\mathbf{s}_{\text{old}}$. In a for-loop, the algorithm takes the first adjacent vertex $a_M \in \text{adj}(\mathbf{s}_{\text{old}})$ that increases the function, that is:

$$d^\top \mathbf{s}_{\text{old}} < d^\top a_M,$$

then set $\mathbf{s}_{\text{old}} \leftarrow a_M$ and repeat the same process over again until there is no adjacent vertex that increases the function. Here is a pseudo-code based on "**3.2 Enhancing GJK**" from (CAMERON, 1997):

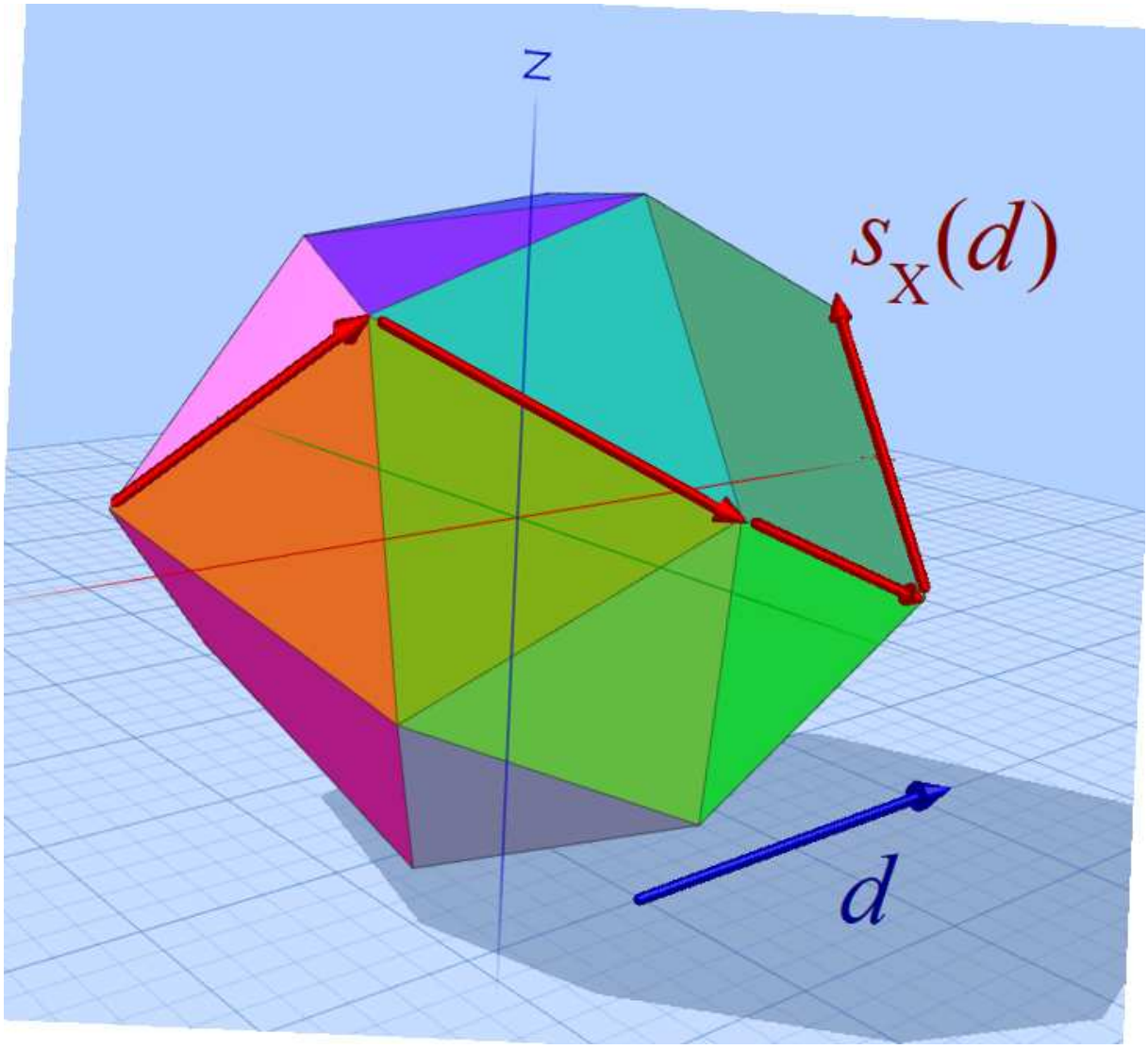
Algorithm 4: Hill Climbing Support Mapping

Data: The adjacency list (X, adj) of a convex polytope, a direction $d \in \mathbb{R}^k$, and a vertex $\mathbf{s} \in X$

Result: $s_X(d)$

```
for a ∈ adj(s) do
    if d^⊤ s < d^⊤ a then
        return hillClimbing((X, adj), d, a) # recursion
    end
end
return s
```

Figure 7 – Visualization of the *hill climbing* support mapping



The algorithm walks to adjacent vertices with higher values.

In (CAMERON, 1997), it claims that, by caching the previous support point, if frame coherency is high, we get 0 or 1 recursions most times, that is, we get a support mapping with $O(1)$ time complexity, hence we get a GJK algorithm with $O(1)$ time complexity.

5.3 CORRECTING HILL CLIMBING

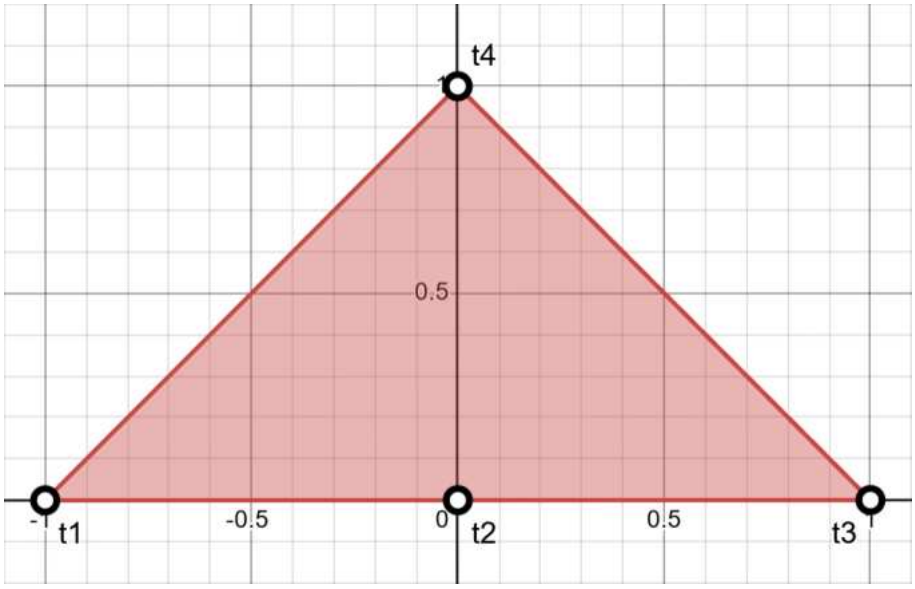
This algorithm is based on a property of convex functions, which is that a local minimum is also a global minimum, whose proof can be found in Section 1.5 from (SCHNEIDER, 1993). As a consequence, a local maximum is also a global maximum since it can be converted to a minimum problem by multiplying the function by -1. On a convex region X, the function $f(x) = d^\top x$ is a convex function because, for all $x_1, x_2 \in X$ and for all $0 \leq \lambda \leq 1$:

$$d^\top((1-\lambda)x_1 + \lambda x_2) = (1-\lambda)d^\top x_1 + \lambda d^\top x_2$$

$$f((1-\lambda)x_1 + \lambda x_2) \leq (1-\lambda)f(x_1) + \lambda f(x_2),$$

which follows the definition of a convex function.

Figure 8 – A counterexample for the *hill climbing* method



However, the problem with this algorithm is that, if $d^\top \mathbf{s} = d^\top a$ for all $a \in \text{adj}(\mathbf{s})$, then we cannot conclude that $\mathbf{s}$ is a support point. A counter-example that shows this problem is an upright triangle with one extra vertex at the base edge. More formally, let $T = \{t_1, t_2, t_3, t_4\} \subset \mathbb{R}^2$ where $t_1 = (-1, 0), t_2 = (0, 0), t_3 = (1, 0), t_4(0, 1)$ and:

$$\text{adj}(t_1) = \text{adj}(t_3) = \{t_2, t_4\}$$

$$\text{adj}(t_2) = \text{adj}(t_4) = \{t_1, t_3\}$$

By doing *hill climbing* on $(T, \text{adj}), d = (0, 1), \mathbf{s} = t_2$, we get:

$$d^\top t_2 = d^\top t_1 = d^\top t_3 = 0,$$

and then t_2 is returned. But:

$$0 = d^\top t_2 < d^\top t_4 = 1,$$

then t_4 is a support point and not t_2, which proves the *hill climbing* method is incorrect. In practice, having $d^\top \mathbf{s} = d^\top a$ for all $a \in \text{adj}(x)$ is extremely unlikely, so the algorithm works fine in almost all cases.

To avoid these extreme cases, we check if the first $\mathbf{s}$ falls in this problematic property. If it does not fall, we can run *hill climbing* normally on $\mathbf{s}$. If it does fall, we run a graph search on (X, adj) to find a vertex $x \in X$ where $d^\top \mathbf{s} \neq d^\top x$. If such x is found, if $d^\top \mathbf{s} < d^\top x$, then we run *hill climbing* normally on x, otherwise $\mathbf{s}$ or any other point with the same value is returned. If such x is not found, then $\mathbf{s}$ is returned because all vertices are support points.

The reason why we only do this for the first $\mathbf{s}$ is that, if the *hill climbing* is called recursively, then it means the previous vertex has a smaller value guaranteed, so the problematic property will never happen. To ensure the base case shares the same property, we do this check before running *hill climbing*. Here is a pseudo-code:

Algorithm 5: Hill Climbing Base Case

```
Data: The adjacency list (X, adj) of a convex polytope, a direction d ∈ ℝ^k, and
      a vertex s ∈ X
Result: s_X(d)
for x ∈ X do
| visited(x) = false
end
visited(s) = true
M ← {s}
while M ≠ ∅ do
| x = M.pop()
| for a ∈ adj(x) do
| | if visited(a) = false then
| | | if d^⊤x < d^⊤a then
| | | | return hillClimbing((X, adj), d, a)
| | | end
| | | if d^⊤x > d^⊤a then
| | | | return x
| | | end
| | | visited(a) = true
| | | M.push(a)
| | end
| end
end
return s
```

5.4 INCREMENTAL SEPARATING AXIS GJK

Separating axis is a very useful concept in collision detection problems since there is a powerful theorem called separating axis theorem, or SAT for short, that states the following: two convex regions $A, B \in \mathbb{R}^n$ are not colliding iff there is an axis separating them. There is a modification for GJK called the GJK separating axis algorithm from (BERGEN, 1999) that, instead of computing the distance vector, computes a separating axis. An advantage of this is a decrease in the number of iterations and the same separating axis can be used in scenarios with frame coherency, sparing computation. Before presenting this modification, it is important to understand the concept of a separating axis.

Definition 5.1 (Separating Axis)**.** Let $\text{proj}_v X = \{v^\top x \mid x \in X\}$ be the set of all projections of X in the vector v. Given two regions $A, B \subset \mathbb{R}^n$, a vector $d \in \mathbb{R}^n$ separates A, B iff

$$\left(\underset{d}{\text{proj}}\, A\right) \cap \left(\underset{d}{\text{proj}}\, B\right) = \emptyset.$$

In other words, d represents the axis $\{\lambda d \mid \lambda \in \mathbb{R}\}$ where, when projecting all points of A and B in this axis, the projections do not collide.

Theorem 5.1. *Let $A, B \subset \mathbb{R}^n$ be two convex regions.*
$A \cap B = \emptyset$ iff there exists a vector $d \in \mathbb{R}^n$ separating A, B.

Proof. We have two statements to prove:

1. $A \cap B = \emptyset \implies \exists d \in \mathbb{R}^n$ (d separates A, B),
2. $A \cap B = \emptyset \impliedby \exists d \in \mathbb{R}^n$ (d separates A, B).

To prove (1.), let $\mathbf{v} = v(A \ominus B)$, let $\mathbf{s} = s_{A \ominus B}(-\mathbf{v})$, and assume A and B are not colliding. As consequence, $\mathbf{v} \neq \mathbf{0}$, which is equivalent to:

$$0 < \mathbf{v}^\top \mathbf{v}.$$

Using theorem 3.1, we have that $\mathbf{v}^\top(\mathbf{v} - \mathbf{s}) = 0$ or, equivalently:

$$\mathbf{v}^\top \mathbf{v} = \mathbf{v}^\top \mathbf{s}.$$

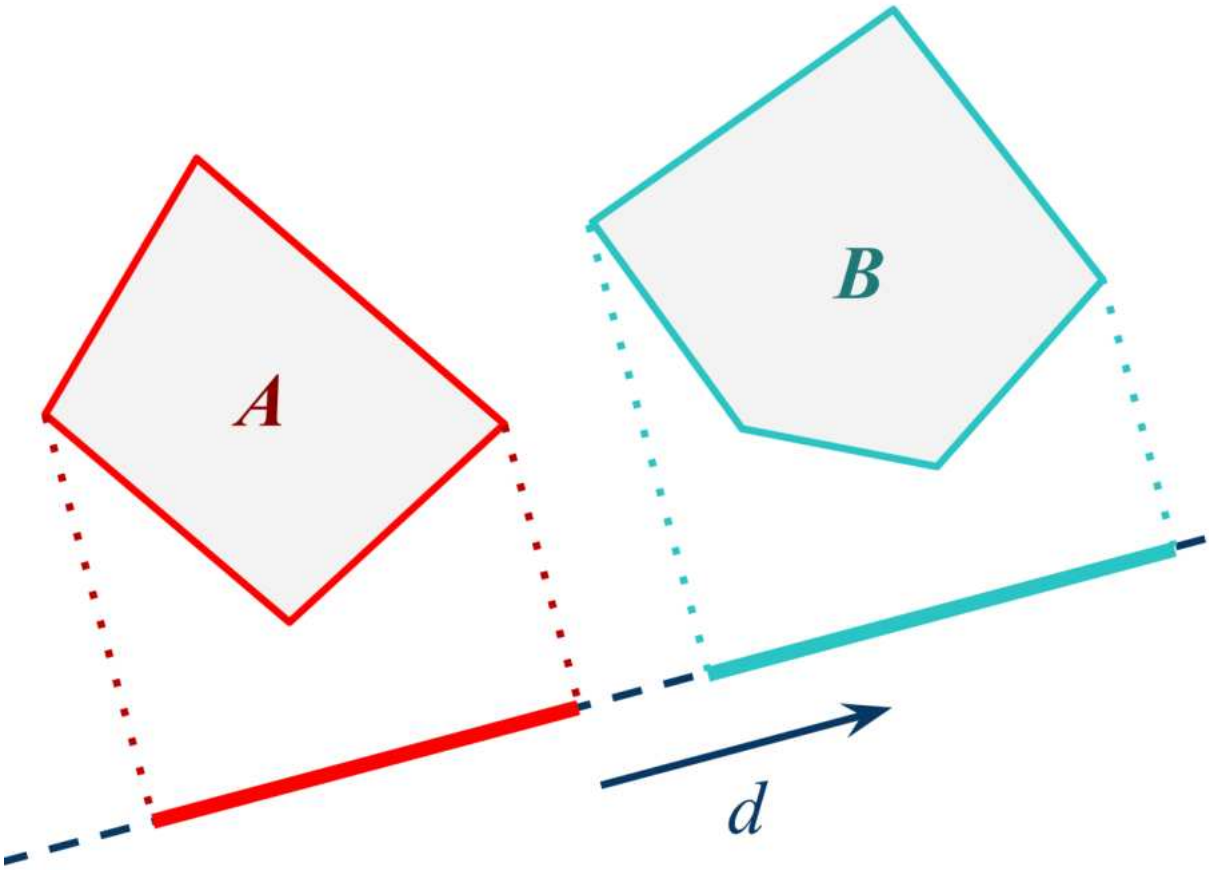
Since $\mathbf{s}$ is a support point in the direction $-\mathbf{v}$, by definition of support points, for all points $x \in A \ominus B$:

$$(-\mathbf{v})^\top x \leq (-\mathbf{v})^\top \mathbf{s},$$

equivalently:

$$\mathbf{v}^\top \mathbf{s} \leq \mathbf{v}^\top x.$$

Figure 9 – Visualization of a separating axis



The axis generated by the vector d separates the regions A and B.

Stacking the inequalities, for all $x \in A \ominus B$, we have:

$$0 < \mathbf{v}^\top \mathbf{v} = \mathbf{v}^\top \mathbf{s} \leq \mathbf{v}^\top x$$

$$0 < \mathbf{v}^\top x,$$

which, for all $a \in A, b \in B$, is equivalent to:

$$0 < \mathbf{v}^\top (a - b)$$

$$\mathbf{v}^\top b < \mathbf{v}^\top a,$$

proving that $\mathbf{v}$ separates A and B.

To prove (2.), suppose, by contradiction, that there exists a $d \in \mathbb{R}^n$ that separates A and B and there exists a point $x \in A \cap B$. Hence, $d^\top x \in \operatorname{proj}_d A$ because $x \in A$ and $d^\top x \in \operatorname{proj}_d B$ because $x \in B$. Therefore:

$$\left(\operatorname*{proj}_d A\right) \cap \left(\operatorname*{proj}_d B\right) \neq \emptyset,$$

which, by definition of separating axis, contradicts the fact that d separates A and B.

□

From the proof above, we can see that, if $v(A \ominus B) \neq \mathbf{0}$, then it separates A and B, so the GJK algorithm already returns a separating axis. Besides that, we can modify the second halt condition to just check if $0 < \mathbf{v}^\top \mathbf{s}$ instead of $\mathbf{v}^\top(\mathbf{v} - \mathbf{s}) = 0$, sparing unnecessary iterations. The idea of this modification is that, if true, then there is a point $a \in A$ and $b \in B$ where $0 < \mathbf{v}^\top(a - b)$, which is equivalent to $\mathbf{v}^\top b < \mathbf{v}^\top a$, proving that $\mathbf{v}$ represents a separating axis. Here is a pseudo-code based on (BERGEN, 1999):

Algorithm 6: The GJK separating axis algorithm

Data: $A, B \subset \mathbb{R}^n$
Result: $\mathbf{v} \in \mathbb{R}^n$ separating A and B
$W \leftarrow$ "arbitrary simplex set from $A \ominus B$"
$\mathbf{v} \leftarrow$ "arbitrary vector in $\mathbb{R}^n$"
while $\mathbf{v} \neq \mathbf{0}$ **do**
  $\mathbf{s} \leftarrow s_A(-\mathbf{v}) - s_B(\mathbf{v})$
  if $0 < \mathbf{v}^\top \mathbf{s}$ **then**
    break
  end
  $(W, \mathbf{v}) \leftarrow$ distanceSubalgorithm$(W \cup \{\mathbf{s}\})$
end
return $\mathbf{v}$

The choice of W and $\mathbf{v}$ does not matter, W can also be empty as in the original GJK separating axis. That is because, after one iteration, we get a simplex with its minimum norm vector for the next iteration, which satisfies the same input conditions of the original GJK, thus it is still correct.

The Incremental Separating Axis GJK, or ISA-GJK, is a tracking version of the GJK separating axis algorithm, that is, it is exactly like the GJK separating axis algorithm with the addition of using previous data to speed up computation. The idea is to reuse the previous separating axis as the initial vector $\mathbf{v}$ regardless of the choice for the initial simplex W, which can be empty. This algorithm also uses *hill climbing* for computing support points and numerical experiments in (BERGEN, 1999) infer an almost constant time complexity and appears to be faster than the Lin-Canny algorithm.

5.5 OTHER MODIFICATIONS

The GJK algorithm is one of the most versatile narrow-phase collision detection algorithms because it can be modified to solve other problems. For example, discrete collision detection, or DCD for short, is a class of collision problems for static objects, but continuous collision detection, or CCD for short, is a class of collision problems for objects in a trajectory. As in DCD, where the main problem is to decide whether two objects are colliding, the main problem for CCD is the *time of impact* problem, or TOI for short, which consists of computing the time or position when objects collide if they ever collide. For linear translations, (BERGEN, 2004) shows a method that uses ISA-GJK for solving TOI by doing a ray cast on the Minkowski difference.

Another example of the problem is a DCD problem called penetration depth, which consists of computing how deep two objects are intersecting, that is, computing the minimum translation vector that separates two objects. More formally, given two convex regions $A, B \subset \mathbb{R}^n$, a penetration vector $p \in \mathbb{R}^n$ is a vector that satisfies:

$$\|p\| = \inf\{\|x\| \mid x \in \mathbb{R}^n \setminus (A \ominus B)\}.$$

To solve this, there is another algorithm called the Expanding Polytope Algorithm from (BERGEN, 2001) that, together with GJK (or the ISA-GJK), computes the penetration vector in case of collision.

6 CONCLUSIONS AND FUTURE WORK

This work mainly focused on giving a simplified approach to the Gilbert-Johnson-Keerthi algorithm described in (GILBERT; JOHNSON; KEERTHI, 1988) by using simple mathematical concepts and showing step-by-step proofs of its properties. In Chapter 5, we analyzed the time and space complexity of the GJK algorithm and covered an improvement and a modification in order to show why the GJK algorithm is one of the fastest and most versatile algorithms for narrow-phase collision detection. These are described in detail with the same simplicity as in the previous chapters.

As discussed in Section 5.1, there is an unproven statement in Chapter 4 from (BERGEN, 1999) saying that the discarded support points do not reappear, which, if correct, would imply that the number of iterations the GJK algorithm does is $O(n)$ on the total number of vertices. Moreover, in (GILBERT; JOHNSON; KEERTHI, 1988) is mentioned the existence of a polygon, which has not been shown, that would prove a quadratic-time lower bound for the original GJK worst-case. Because the original algorithm uses the $\Theta(n)$ separating mapping, this would imply that the number of iterations the GJK algorithm does is $\Omega(n)$ in the worst case.

Up to the present date, a formal proof for the $O(n)$ number of iterations could not be completed due to how difficult it was to prove properties that seemed to be geometrically obvious. There is a class of inputs called the *death polytope* that has been designed to force this quadratic behavior mentioned in (GILBERT; JOHNSON; KEERTHI, 1988), proving the $\Omega(n)$ number of iterations. However, as with the incomplete proof for the $O(n)$, formalizing the constructions of such a class and proving that it causes the $\Omega(n)$ number of iterations demands difficult concepts of analysis in $\mathbb{R}^n$, only to formalize seemingly obvious geometrical properties. Besides that, a few experiments indicated the *death polytope* possibly causes the supposed worst-case scenario of the GJK algorithm, and this class oddly coincides with the description given in (GILBERT; JOHNSON; KEERTHI, 1988) that says the vertices cluster closer to a certain point as the number of vertices increases. In a future work, these two challenging proofs might be attempted with the assistance of a mathematician with expertise in geometry or convex analysis.

REFERÊNCIAS

BARBER, C. B.; DOBKIN, D. P.; HUHDANPAA, H. The quickhull algorithm for convex hulls. **ACM Transactions on Mathematical Software**, v. 22, n. 4, p. 469–483, 1995.

BERGEN, G. van den. Fast and robust gjk implementation for collision detection of convex objects. **Journal of Graphics Tools**, v. 4, n. 2, p. 7–25, 1999.

BERGEN, G. van den. Proximity queries and penetration depth computation on 3d game objects. 2001.

BERGEN, G. van den. Ray casting against general convex objects with application to continuous collision detection. **Journal of Graphics Tools**, 2004.

CAMERON, S. Enhancing gjk: Computing minimum and penetration distances between convex polyhedra. **IEEE Proceedings of International Conference on Robotics and Automation**, 1997.

ERICSON, C. **Real-Time Collision Detection**. [S.l.]: CRC Press, 2004. 632 p.

GILBERT, E. G.; JOHNSON, D. W.; KEERTHI, S. S. A fast procedure for computing the distance between complex objects in three-dimensional space. **IEEE Journal of Robotics and Automation**, v. 4, n. 2, p. 193–203, 1988.

LIDECN, L. M. Q. L.; SIVIC, V. P. J.; CARPENTIER, J. Collision detection accelerated: An optimization perspective. **Robotics: Science and Systems 2022**, 2022.

LIN, M. C.; CANNY, J. F. A fast algorithm for incremental distance calculation. **Proceedings. 1991 IEEE International Conference on Robotics and Automation**, v. 2, n. 2, p. 1008–1014, 1991.

MONTANARI, M.; PETRINIC, N. Improving the gjk algorithm for faster and more reliable distance queries between convex objects. **ACM Transactions on Graphics**, v. 36, n. 4, p. 193–203, 2016.

SCHNEIDER, R. G. **Convex bodies: the Brunn-Minkowski theory**. [S.l.]: Cambridge University Press, 1993. 490 p.