Improving the GJK Algorithm for Faster and More Reliable Distance Queries Between Convex Objects

MATTIA MONTANARI and NIK PETRINIC
University of Oxford
and
ETTORE BARBIERI
Queen Mary University of London

This article presents a new version of the Gilbert-Johnson-Keerthi (GJK) algorithm that circumvents the shortcomings introduced by degenerate geometries. The original Johnson algorithm and Backup procedure are replaced by a distance subalgorithm that is faster and accurate to machine precision, thus guiding the GJK algorithm toward a shorter search path in less computing time. Numerical tests demonstrate that this effectively is a more robust procedure. In particular, when the objects are found in contact, the newly proposed subalgorithm runs from 15% to 30% times faster than the original one. The improved performance has a significant impact on various applications, such as real-time simulations and collision avoidance systems. Altogether, the main contributions made to the GJK algorithm are faster convergence rate and reduced computational time. These improvements may be easily added into existing implementations; furthermore, engineering applications that require solutions of distance queries to machine precision can now be tackled using the GJK algorithm.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Physically based modeling*

General Terms: Accurate simulations

Additional Key Words and Phrases: Distance measurement, collision detection, Gilbert-Johnson-Keerthi algorithm

ACM Reference Format:

Mattia Montanari, Nik Petrinic, and Ettore Barbieri. 2017. Improving the GJK algorithm for faster and more reliable distance queries between convex objects. ACM Trans. Graph. 36, 3, Article 30 (June 2017), 17 pages.
DOI: http://dx.doi.org/10.1145/3083724

The authors gratefully acknowledge the financial support by the UK Technology Strategy Board and Rolls-Royce plc. through Strategic Investment in LOw-carbon Engine Technology (SILOET) project.

Authors' addresses: mattia.montanari@eng.ox.ac.uk, nik.petrinic@eng.ox.ac.uk, e.barbieri@qmul.ac.uk.

DOI: http://dx.doi.org/10.1145/3083724

1. INTRODUCTION

The Gilbert-Johnson-Keerthi (GJK) algorithm [Gilbert et al. 1988] is a descent method for computing a pair of closest points between two convex objects. Because of its extreme versatility and simple implementation, this procedure is being adopted in a wide range of applications such as robotics [Zheng et al. 2011; Dietrich et al. 2012; Zheng and Yamane 2013], real-time haptic rendering [Jimnez et al. 2001; Laycock and Day 2007], rigid-body dynamics [Redon et al. 2002; Tasora and Anitescu 2011], medical surgery [Liu et al. 2003], computer graphics [Museth et al. 2005; Seiler et al. 2008], physics [Movshovitz et al. 2012; Haji-Akbari et al. 2013; Millan et al. 2014], and, more rarely, in computational mechanics [Wachs et al. 2012].

Several studies comparing the GJK algorithm with similar procedures can be found in the literature. Cameron [1997a] compared his enhanced GJK with the Lin and Canny (LC) algorithm [Lin and Canny 1991], showing superiority of the former. More recently, two incremental algorithms have been proposed: Voronoi-clip (V-Clip) [Mirtich 1998] and Chung-Wang (CW) [Chung and Wang 1996]. Both studies concluded that the GJK algorithm has comparable computational costs. Nevertheless, unlike other procedures, the GJK algorithm is not limited to polytopes and is therefore the most versatile. It can compute the minimum distance between two arbitrary geometrical representations—such as polytopes, quadrics [van den Bergen 2003], and Nonuniform Rational B-Splines (NURBS) [Turnbull and Cameron 1998]. Incremental implementations, such as enhanced GJK [Cameron 1997b], ISA-GJK [van den Bergen 1999], or R-GJK [Ong and Gilbert 2001], are particularly efficient for dynamic simulations with high temporal coherence. The advantage over direct (or one-shot) algorithms, for example, Held [1998], is that, by exploiting temporal coherence, the GJK algorithm reuses data cached at previous solution steps to speed up the contact search. For overlapping objects this procedure can also be tailored to estimate the penetration vector [Cameron 1997b]. Overall, GJK is one of the fastest and certainly most versatile algorithms for solving distance queries between convex bodies.

Previous studies emphasised that the GJK algorithm suffers from numerical instability when dealing with degenerate geometries [van den Bergen 2003]; in such situations, the procedure becomes inefficient and not suitable for applications requiring high performance and accuracy [Nye et al. 2014]. The main source of numerical error is the *distance subalgorithm*, also known as the Johnson algorithm [Johnson 1987]. Its limited accuracy reduces the convergence rate and may yield to false positives that can, for example, ruin the user experience in a computer game, produce instabilities in finite element simulations, or lead a real-time collision avoidance system to failure. For these reasons the *Backup procedure* was originally added to handle pathological cases at the expense of CPU time and implementation effort. A decade

thereafter van den Bergen [1999] replaced the Backup procedure with a more robust exit condition that, however, does not fully counterbalance the instabilities affecting the Johnson algorithm. To the best of our knowledge, the only attempt to replace the original distance subalgorithm was formalised by Ericson [2004] and used in Tereshchenko et al. [2013]. Despite being mathematically equivalent, the implementation of this method requires a cascade of conditional statements whose results are less accurate than the original subdistance algorithm and are computationally more expensive. Therefore, the lack of success of the GJK algorithm in scientific computing is probably due to its insufficient reliability.

The distance subalgorithm essentially computes, in the most general case, the point of a tetrahedron that is closer to the origin than any other point of the tetrahedron. This problem finds many practical applications and is therefore widely studied in computational geometry, but to achieve maximum performance a subalgorithm must take advantage of the descent nature of the GJK procedure. Existing general-purpose approaches to this problem, for example, Edelsbrunner and Mucke [1990], examine 15 subsets (four vertices, six edges, four faces, and the interior) defined by an arbitrary tetrahedron and test which one of these contains the closest point to the origin. Nevertheless, in the particular context of the GJK algorithm, whilst the orientation of the tetrahedron is arbitrary, its construction is not. This allows one to discard a priori six subsets and thus to save computational time. Overall, despite its numerical instabilities, the Johnson subalgorithm results extremely efficient.

With the aim of improving the performance of the GJK algorithm, this article presents a new method that replaces the Johnson subalgorithm and Backup procedure. The proposed method is computationally efficient as it discards a priori incompatible subsets but, unlike the Johnson algorithm, it handles naturally degenerate geometries. The enhanced robustness leads to faster convergence rates and more accurate distance queries. Moreover, our method benefits from a simple implementation that can be readily included into existing software. The aim of this study is indeed to improve the GJK algorithm and to inspire new applications in computational sciences that use accurate numerical methods (e.g., Finite Element Method (FEM) [Reddy 2006], Discrete Element Method (DEM) [Cundall and Strack 1979], meshless method [Gingold and Monaghan 1977], and Isogeometric Analysis (IGA) [Hughes et al. 2005]).

The structure of this article is as follows. In Sections 2 and 3, background notions of convex analysis and the original GJK algorithm are reviewed. Section 4 describes the new method and its implementation. Numerical results comparing the two subalgorithms are presented in Section 5. Concluding remarks are reported in Section 6.

2. BACKGROUND

The Euclidean norm $\|\mathbf{x}\|$ of a vector $\mathbf{x}$ in $\mathbb{R}^n$ is given by $\|\mathbf{x}\| = \sqrt{\mathbf{x}\cdot\mathbf{x}}$, where the dot represents the inner product. The components of any vector are denoted by superscripts, for example, x^i for $i = 1, 2, 3$ or, equivalently, $[x^x, x^y, x^z]$. The convex hull of a finite set of points $Y = \{y_1, \ldots, y_m\}$ is given by

$$\mathrm{conv}(Y) = \left\{ \sum_{i=1}^{m} \lambda_i y_i : y_i \in Y, \lambda_i \geq 0, \sum_{i=1}^{m} \lambda_i = 1 \right\}. \quad (1)$$

The difference between two points y_1 and y_2 is the vector $\mathbf{y_2y_1}$. In what follows, the vector $\mathbf{y}_i$ corresponds to the difference between the point y_i and the origin O.

For the sake of clarity, this work will consider only objects represented by nonempty and finite sets in $\mathbb{R}^n$: polygons in $\mathbb{R}^2$ or polytopes in $\mathbb{R}^3$. Moreover, we shall deal with convex objects only, so that the space occupied by an object is defined, from Equation (1), by $Y = \mathrm{conv}(Y)$. Concave objects may be decomposed and treated as separate convex sets [Gilbert et al. 1988].

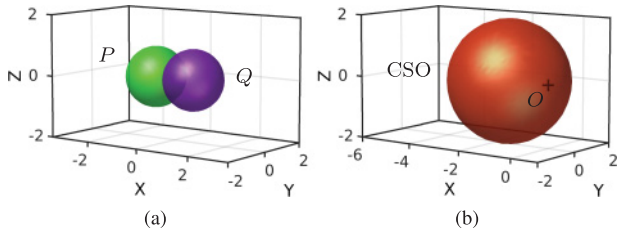


Fig. 1. Physical (a) and configuration space (b) for colliding spheres P and Q. The Configuration Space Obstacle (CSO) includes the origin.

We shall refer to the space $\mathbb{R}^n$ in which two objects P and Q reside as *physical space*, and for these a distance query returns a pair of *witness points* $z_P \in P$ and $z_Q \in Q$ such that

$$d(P, Q) = \|z_P - z_Q\| = \min\{\|p - q\| : p \in P,\ q \in Q\}. \quad (2)$$

In the preceding equation, $d(P, Q)$ is the *minimum Euclidean distance* that corresponds to the norm of the *separating vector* $\mathbf{z_Qz_P}$. This is a natural choice for measuring the distance since it offers an intrinsic description of practical interest for many applications.

Notice that for polytopes (polygons) with parallel facets (edges) the pair of witness points is not unique. Under these circumstances a distance query has infinite solutions, and thus the physical space is not best suited for computing $d(P, Q)$.

2.1 Configuration Space

It is convenient to recast the distance query in a *configuration space* where a unique solution exists and can be found by solving a *point-in-polytope* problem. The Minkowski difference between two objects P and Q results in the *Configuration Space Obstacle* (CSO):

$$\mathrm{CSO} = P - Q = \{x_P - x_Q : x_P \in P, x_Q \in Q\}, \quad (3)$$

which is a set of vectors embedded in an affine space with a fixed frame and origin O [Lozano-Perez 1983].

It can be proven that the CSO is convex [Webster 1995] and, from Equations (2) and (3), that a distance query translates into finding the *point of minimum norm* v (CSO) [Cameron and Culley 1986; Gilbert et al. 1988]. Because the CSO is convex, a unique vector $\boldsymbol{v}$ (CSO) exists, which is optimal in the sense that

$$\|\boldsymbol{v}\,(\mathrm{CSO})\| = d(\mathrm{CSO}, O). \quad (4)$$

The new Minimum Norm Duality theorem [Dax 2006] proves geometrically that to the minimum distance d(CSO, O) corresponds a *separating hyperplane* between the CSO and the origin O. Together with Equations (1)–(3), this theorem links configuration and physical spaces with the following equality:

$$\boldsymbol{v}\,(\mathrm{CSO}) = \mathbf{z_Qz_P}. \quad (5)$$

Intuitively, the most important consequence of the Minkowski operator is that if the objects intersect, the origin O is found inside the CSO. Consider, for example, the two overlapping spheres P and Q in Figure 1(a). The CSO resulting from the Minkowski difference $P - Q$, depicted in Figure 1(b), includes the origin of the configuration space, that is, $O \cap \mathrm{CSO} \neq \emptyset$.

For distant objects the distance $d(P, Q)$ is greater than zero. The example in Figure 2(a) shows two distant objects and their separating plane defined at equal distance. In this configuration the CSO, illustrated in Figure 2(b), does not include the origin O.

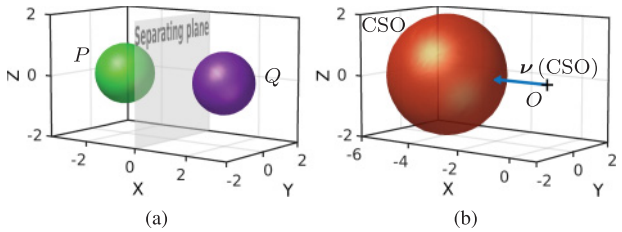


Fig. 2. Physical (a) and configuration space (b) for separated spheres P and Q. The configuration space obstacle (CSO) does not include the origin and has a unique point of minimum norm ν (CSO).

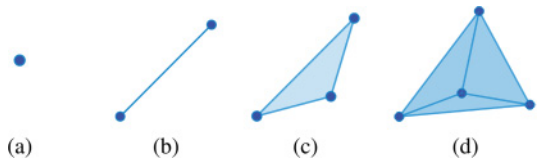


Fig. 3. In $\mathbb{R}^3$ a simplex can either be a point (a), a line segment (b), a triangle (c), or a tetrahedron (d).

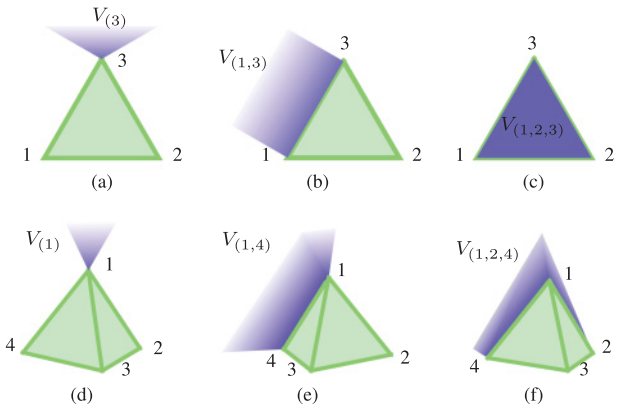


Fig. 4. Voronoi regions of a 2-simplex (a)–(c) and 3-simplex (d)–(f).

2.2 Simplices

Evaluating the entire Minkowski difference at runtime for every solution step is computationally infeasible. Instead of computing explicitly the CSO, ν (CSO) is approximated by the point of minimum norm of a *simplex*.

An m-simplex $\tau = \{s_1, \ldots, s_{m+1}\}$ is the convex hull of a set of $m+1$ affinely independent points in $\mathbb{R}^n$; moreover, by selecting the vertices $\{s_1, \ldots, s_{m+1}\}$ on the boundary of CSO, the simplex is a subset of CSO. Because the CSO is a compact set, every convex combination of points of CSO belongs to CSO, that is, $\tau \subset$ CSO. The simplices for three-dimensional problems are depicted in Figure 3.

Each m-simplex has $(2^{m+1}-1)$ *faces* (e.g., vertices, edges, faces, volume). A face is a subset of $r+1$ vertices of τ, and is called r-face of τ.

To each face is associated a *region* of points of $\mathbb{R}^n$ defined as the set of points which are at least as close to a point of the face as to any other point of τ not in the face. For example, the region associated to one of the vertices of a two-dimensional simplex is represented in Figure 4(a). Other examples of regions are shown in Figures 4(b) and 4(c) for a planar simplex, whereas Figures 4(d)–4(f) show examples in $\mathbb{R}^3$. Because the notion of region is similar to the one of the higher order Voronoi diagram, we refer to these simply as *Voronoi regions*. The Voronoi region associated to a face of τ is denoted by V_κ, where κ is an ordered and nonempty tuple. Such a tuple lists the indices i of the set $\{s_i\}$ defining the face to which V_κ is associated. Moreover, if a point $w \in \text{conv}(W)$, κ defines the region V_κ associated to the subset $W \subset \tau$ that *supports* w.

A theorem due to Carathéodory establishes that a point $y \in \text{conv}(Y) \subset \mathbb{R}^n$ can be expressed as a convex combination of $n+1$ or fewer points of Y [Webster 1995]. This allows one to express a point w laying on an r-face of a simplex τ as a convex combination of $n+1$ or fewer vertices:

$$\begin{aligned} w &= \sum_{i \in I} \lambda_i s_i \ : \ \lambda_i \geq 0, \ \sum_{i=1}^{n+1} \lambda_i = 1, \\ &= \sum_{j \in \kappa} \lambda_j s_j \ : \ \lambda_j > 0, \ \sum_{i=1}^{r+1} \lambda_j = 1, \end{aligned} \tag{6}$$

where $I = \{1, \ldots, n+1\}$ and $r \leq n$. The coefficients λ are called *barycentric coordinates*. They inherit properties from Eq. (1) and it can be shown that they are invariant to affine transformations [Coxeter 1989].

3. GJK ALGORITHM

3.1 Main Procedure

The GJK algorithm solves the distance query in Equation (2) iteratively. At each k-th iteration a simplex τ_k is moved toward the origin O. When $\nu(\tau_k)$ is sufficiently close to ν (CSO), the algorithm terminates as the solution to the equivalent problem in Equation (4) is found.

The iterative search descends in the sense that the simplex τ_{k+1} offers a better approximation to ν (CSO) than τ_k. This is done by removing from τ_k a point that is "far from O", and adding a closer vertex $w_k \in$ CSO to form τ_{k+1}; namely,

$$\| \boldsymbol{\nu}(\tau_{k+1}) \| = \| \boldsymbol{\nu}(\text{conv}(\{\tau_k, w_k\})) \| \leq \| \boldsymbol{\nu}(\tau_k) \|. \tag{7}$$

The preceding equation states that a sequence of simplices indeed converges monotonically to ν (CSO) [Gilbert et al. 1988], provided that a point $w_k \in$ CSO is available without having to compute the entire Minkowski difference in Equation (3).

The point w_k in Equation (7) can be computed efficiently by evaluating a *support function* $h_{\text{CSO}} : \mathbb{R}^n \to \mathbb{R}$. This is defined for all vectors $\mathbf{k} = k - O$, with $k \in$ CSO, by $h_{\text{CSO}}(\mathbf{v}) = \max\{\mathbf{v} \cdot \mathbf{k}\}$. The GJK algorithm computes the support function on each object independently:

$$h_{\text{CSO}}(\mathbf{v}_k) = h_{P-Q}(\mathbf{v}_k) = h_P(\mathbf{v}_k) - h_Q(-\mathbf{v}_k). \tag{8}$$

Equation (8) identifies the furthest point $w_k \in$ CSO from the hyperplane defined by $\mathbf{v}_k$ along the direction $\mathbf{v}_k$. This point, not necessarily unique, lays on the outer boundary of CSO and is such that $h_{\text{CSO}}(\mathbf{v}_k) = \mathbf{w}_k \cdot \mathbf{v}_k$. The vector $\mathbf{w}_k$ is a solution of h_{CSO}.

The extreme versatility of the GJK algorithm is a consequence of the fact that h_{CSO} admits solutions for any representation of convex body, for example, polytopes, quadrics, and NURBS. For two polytopes P and Q with m_P and m_Q points, respectively, $h_P(\mathbf{v}) = \max\{\mathbf{v} \cdot \mathbf{p}_i : i = 1, \ldots, m_P\}$ and $h_Q(-\mathbf{v}) = \max\{-\mathbf{v} \cdot \mathbf{q}_i : i = 1, \ldots, m_Q\}$. Therefore, the cost of the evaluation of h_{CSO} scales linearly with the number of vertices in P and Q. In fact, the GJK algorithm solves a problem of complexity $m_P + m_Q$, and not $m_P m_Q$. Incremental versions of the GJK algorithm reduce this complexity

to almost a constant [Cameron and Culley 1986; Cameron 1997b; van den Bergen 1999; Ong and Gilbert 2001]. More details on the numerical evaluation of Equation (8) for nonpolytopes can be found in van den Bergen [2003] and de Berg et al. [2008], where the case $\mathbf{v}_k = \mathbf{0}$ is also discussed.

The numerical GJK algorithm essentially consists of a conditional loop, as presented in Algorithm 1. The loop begins by evaluating Equation (8). It then tests convergence; if the test is negative a new search direction $\mathbf{v}_{k+1}$ is computed. The algorithm repeats these steps until (a) the objects are found in contact, or (b) the simplex τ_k cannot move closer to the origin. In this work, we adopt similar exit conditions to those proposed in van den Bergen [2003]: the objects are in contact if the simplex contains the origin O, or if

$$\|\mathbf{v}_k\|^2 \leq \varepsilon_{tol} \max\left\{\|y - O\|^2 : y \in W\right\}, \tag{9}$$

where W is the smallest subset of vertices in τ_k that supports $\nu(\tau_k)$. Additionally, the GJK algorithm terminates when $\nu(\tau_k)$ is sufficiently close to ν(CSO). That is, if $w_k \in \tau_k$ or if

$$\|\mathbf{v}_k\|^2 - \mathbf{v}_k \cdot \mathbf{w}_k \leq \varepsilon_{rel}^2 \|\mathbf{v}_k\|^2. \tag{10}$$

The accuracy of the GJK algorithm is set by the arbitrary value ε_{tol}. The user specifies this value. For usual applications this is within 10^{-8} and machine precision; namely, $10^{-8} \leq \varepsilon_{tol} \leq 10^{-16}$.

Upon termination of the GJK algorithm, a pair of witness points z_P and z_Q is computed. By keeping track of the vertices $\{s_1, \ldots, s_{m+1}\}$ of the m-simplex τ_k, for $p_i \in P$, $q_i \in Q$, we write

$$\begin{aligned}
\nu(\tau_k) &= \sum_{i=1}^{m+1} \lambda_i s_i = \sum_{i=1}^{m+1} \lambda_i (O + \mathbf{s}_i) = \sum_{i=1}^{m+1} \lambda_i (O + \mathbf{q_i p_i}) \\
&= \sum_{i=1}^{m+1} \lambda_i (O + \mathbf{O p_i} + \mathbf{q_i O}) = \sum_{i=1}^{m+1} \lambda_i p_i + \sum_{i=1}^{m+1} \lambda_i \mathbf{q_i O} \\
&= z_P + \sum_{i=1}^{m+1} \lambda_i \mathbf{q_i O} = \mathbf{z_Q z_P} + z_Q + \sum_{i=1}^{m+1} \lambda_i \mathbf{q_i O} \\
&= \mathbf{z_Q z_P} + \sum_{i=1}^{m+1} \lambda_i (q_i + \mathbf{q_i O}) = \mathbf{z_Q z_P} + O \\
&= \sum_{i=1}^{m+1} \lambda_i (p_i - q_i) + O.
\end{aligned} \tag{11}$$

This proof makes use of Equation (6) to derive the relationship that leads to the solution of the initial problem in Equation (2).

From Equation (11) we obtain the vector $\mathbf{v}_k$ which is of utmost importance since it governs both search direction and termination tests. $\mathbf{v}_k$ is defined as follows:

$$\nu(\tau_k) - O = \|\mathbf{v}_k\| = \min\left\{\|s - O\| : s \in \tau_k\right\}. \tag{12}$$

The distance subalgorithm is responsible for computing $\nu(\tau_k)$ and it will be presented in the next section. This section terminates with an example that illustrates the GJK procedure.

Let us consider the problem of measuring the minimum distance $d(P, Q)$ between the polygons P and Q in Figure 5(a). Since the objects are distant, a pair of witness points defining the separating vector (in red) exists. The GJK algorithm begins by initialising $W = \tau_0 = \emptyset$ and an arbitrary search direction $\mathbf{v}_0$. In our example, $\mathbf{v}_0$ is set downward, such that the farthest point of the CSO along the direction $-\mathbf{v_0}$ is D-G. This identifies a solution vector for the support function, in fact $h_{\text{CSO}}(\mathbf{v}_0) = \mathbf{D\text{-}G} \cdot \mathbf{v}_0$. For the first iteration $k = 1$, the simplex $\tau_1 = \{\text{D-G}\}$ and $\mathbf{v}_1 = \mathbf{D\text{-}G}$. All quantities involved are illustrated in Figures 5(b) (notice that the CSO is shown for illustration purposes only). In the second iteration B-E is found to be the farthest point of the CSO along $-\mathbf{v}_1$, and is listed first in τ_2. Figure 5(c) shows that both vertices support the point of minimum norm $\nu(\tau_2)$, which is then used to update the search direction $\mathbf{v}_2$. A similar situation occurs in the third iteration. As shown in Figure 5(d), the subset supporting $\nu(\tau_3)$ is $W = \{\text{C-E}, \text{B-E}\} \subset \tau_3$; moreover, because $\nu(\tau_3) = \nu$(CSO), the GJK algorithm terminates at the end of this iteration.

This example illustrates geometrically that the GJK is a descent method; in fact, the radius of the dashed circles in Figures 5(b)–5(d), centred at the origin and passing by the point of minimum norm of each simplex, decreases monotonically at every iteration.

ALGORITHM 1: GJK Algorithm (Adapted from van der Bergen [2003])

```
1:  procedure GJK(P, Q, v_0)
2:      k = 0
3:      v_1 = v_0
4:      τ_k = ∅, W_k = ∅
5:      repeat
6:          k = k + 1
7:          w_k = s_{P−Q}(−v_k);
8:          if w_k ∈ Y or ‖v_k‖² − v_k · w_k ≤ ε²_rel ‖v_k‖² then
9:              continue
10:         τ_k = {w_k} ∪ W_{k−1};
11:         [W_k, λ] = DistanceSubalgorithm(τ_k);
12:         v_{k+1} = Σ λ_i y_i : y_i ∈ W_k, i = 1, ..., |W_k|;
13:     until |W_k| = 4 or ‖v_k‖² ≤ ε_tol max{‖y_k‖² : y_k ∈ W_k}
14:     return ‖v_k‖
```

3.2 Distance Subalgorithm

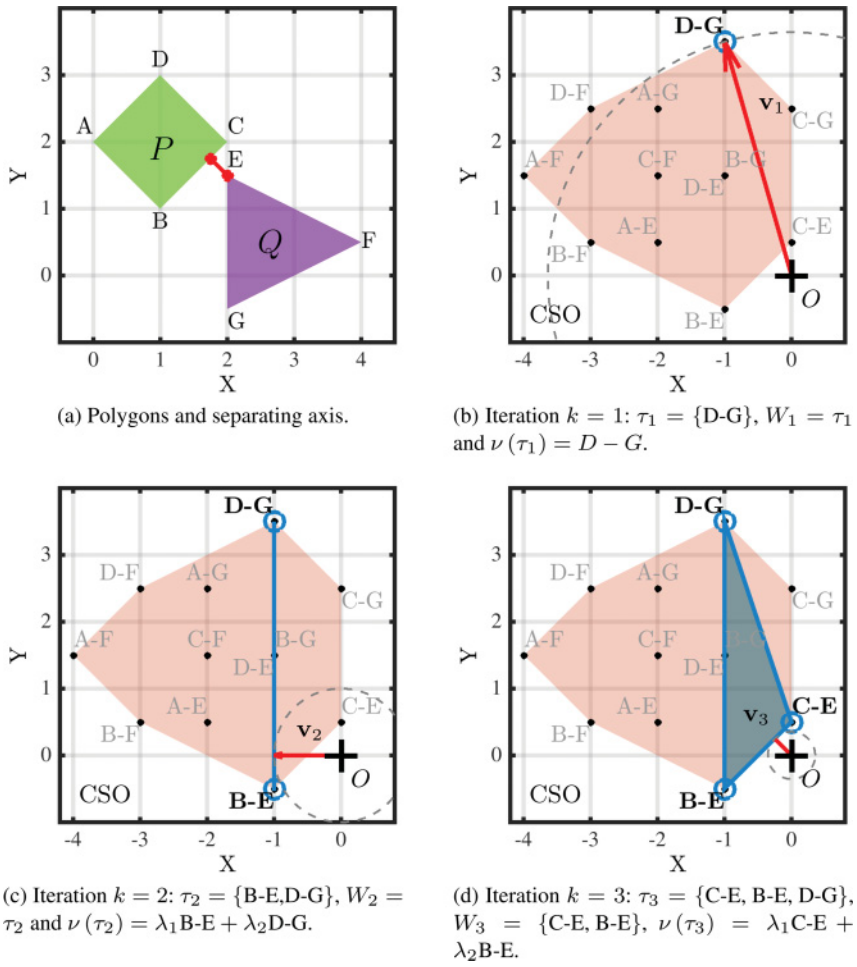
The GJK algorithm relies heavily on the distance subalgorithm. This is responsible for computing the point $\nu(\tau_k)$, which dictates the search direction $\mathbf{v}_k$, and hence controls both exit conditions and support function evaluation. More importantly, the convergence rate of the GJK procedure decays if the distance subalgorithm is not accurate to machine precision. We shall demonstrate this fact in Section 5 with an important result: there are cases where the CPU time is reduced by deactivating the Johnson algorithm and relying entirely on the Backup procedure, rather than using the usual combination of both.

The distance subalgorithm computes the barycentric coordinates and a subset $W \subseteq \tau$ to express $\nu(\tau_k)$ as a convex combination of the smallest set of vertices. The subset W identifies, with a tuple κ, the face of an m-simplex supporting $\nu(\tau_k)$. It has been shown [Gilbert et al. 1988] that from Equation (6) comes a representation of $\nu(\tau_k)$ that satisfies

$$\lambda_j > 0 \text{ and } \lambda_i \leq 0 \ \forall \ j \in \kappa : i = 1, \ldots, m+1, \ i \neq j. \tag{13}$$

Equation (13) guarantees that $\nu(\text{aff}(W)) = \nu(\text{conv}(\tau))$ [van den Bergen 1999], which geometrically means that the vector $\boldsymbol{\nu}(\tau_k)$ is perpendicular to a face of the simplex. This face is "optimal" in the sense of Equation (12), for which reason Equation (12) is also called *orthogonal condition*. This can be seen from Figure 5(d): the simplex does have three vertices, but only two of these support $\nu(\tau_3)$. The two vertices form a face that is perpendicular to $\mathbf{v}_3$, and their barycentric coordinate will be strictly positive.

The Johnson algorithm [Johnson 1987] is the distance subalgorithm originally adopted in Gilbert et al. [1988] to compute $\nu(\tau_k)$. This is implemented, together with the Backup procedure, in the functions `DistanceSubalgorithm` in Algorithm 1. Johnson



(a) Polygons and separating axis.

(b) Iteration $k = 1$: $\tau_1 = \{\text{D-G}\}$, $W_1 = \tau_1$ and $\nu(\tau_1) = D - G$.

(c) Iteration $k = 2$: $\tau_2 = \{\text{B-E,D-G}\}$, $W_2 = \tau_2$ and $\nu(\tau_2) = \lambda_1\text{B-E} + \lambda_2\text{D-G}$.

(d) Iteration $k = 3$: $\tau_3 = \{\text{C-E, B-E, D-G}\}$, $W_3 = \{\text{C-E, B-E}\}$, $\nu(\tau_3) = \lambda_1\text{C-E} + \lambda_2\text{B-E}$.

Fig. 5. Iterative GJK procedure for the pair of distant polygons in (a). Each k-th iteration (b)–(d) involves points on the configuration space CSO, subset of support vertices W_k, point closest to origin $\nu(\tau_k)$, and simplex τ_k.

algorithm recursively inspects all the Voronoi regions of a m-simplex until the region V_κ satisfying Equation (13) is found, that is, until the signs of the barycentric coordinates comply with Equation (13).

The barycentric coordinates are computed by solving an algebraic system of equations that embeds the orthogonal condition from Equation (12). Let a simplex $\tau = \{s_i\}$, with $i \in I = \{1, \ldots, m+1\}$, have its point of minimum norm $\nu(\tau)$ laying on an r-face defined by the points in $\{s_j\}$, with $j \in \kappa$. Then this face is perpendicular to a vector $\mathbf{v}$ or, equivalently, $(s_j - s_l) \cdot \mathbf{v} = 0$, for all $j \neq l$, where l is an arbitrary element of κ. The vector $\mathbf{v}$ was defined in Equation (12) and can be written in terms of $r+1$ barycentric coordinates λ_j to derive an algebraic system $\mathbf{A}\boldsymbol{\lambda} = \mathbf{b}$. In extended form this writes:

$$\begin{bmatrix} 1 & \ldots & 1 \\ \ldots & \ldots & \ldots \\ (\mathbf{s}_j - \mathbf{s}_l)\cdot\mathbf{s}_1 & \ldots & (\mathbf{s}_j - \mathbf{s}_l)\cdot\mathbf{s}_r \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \ldots \\ \lambda_r \end{bmatrix} = \begin{bmatrix} 1 \\ \ldots \\ 0 \end{bmatrix}. \quad (14)$$

The index $l \in \kappa$ must be kept constant, usually $l = 1$, and j takes the remaining r values in κ.

Behind the computational efficiency of the Johnson algorithm stands the recursive solution of Equation (14) for all subsets of τ. Because κ has cardinality at the most 4, the system $\mathbf{A}\boldsymbol{\lambda} = \mathbf{b}$ can be efficiently solved by using Cramer's rule. However, neither κ nor its cardinality are known a priori. The procedure for computing these quantities is detailed below.

A solution of the algebraic system in Equation (14) can be written by combining Cramer's rule with a cofactor expansion as follows:

$$\lambda_j = \frac{-1^{1+j} \det \mathbf{A}_{1j}}{\det \mathbf{A}}, \quad (15)$$

where $\mathbf{A}_{1j}$ are minors of $\mathbf{A}$ obtained by removing the first row and the j-th column from $\mathbf{A}$.

To understand the combinatorial logic of the Johnson algorithm, let us look at $\mathbf{A}$ as a minor of another matrix built in the same fashion but for a larger subset. This larger subset is obtained by adding to W a vertex of the simplex s_i not yet included in W, namely, $\{s_j : j \in \kappa\} \cup s_i$ for an $i \in I - \kappa$. This leads to a recursive solution for $\mathbf{A}\boldsymbol{\lambda} = \mathbf{b}$:

$$\lambda_j = \frac{\Delta_j(W)}{\sum_{j\in\kappa} \Delta_j(W)}, \quad (16)$$

where $\Delta_j(W)$ is a cofactor of $\mathbf{A}$ for one of the $(2^{m+1} - 1)$ subsets W of τ. Since the vertices of τ are linearly independent:

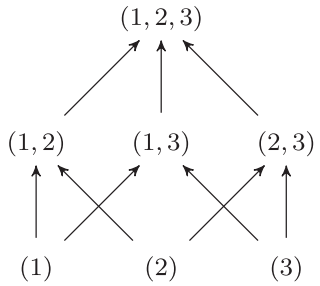


Fig. 6. Johnson algorithm conducts a bottom-up search across all Voronoi regions of a 2-simplex.

$$\Delta_j(W) = -1^{1+j} \det \mathbf{A}_{1j}. \tag{17}$$

The Johnson algorithm computes the values $\Delta_j(W)$ in order of increasing cardinality of W. For the first $m+1$ singleton the solution is trivial:

$$\Delta_j(\{s_j\}) = 1, \quad j \in I, \tag{18}$$

whereas for the remaining subsets:

$$\Delta_i(W \cup s_i) = \sum_{j \in \kappa} \Delta_j(W)(\mathbf{s}_j \cdot \mathbf{s}_l - \mathbf{s}_j \cdot \mathbf{s}_i) \ : \ i \in I - \kappa. \tag{19}$$

The preceding equation is tested for all subsets of τ, keeping $l \in \kappa$ constant, until Equation (13) is verified.

A theorem demonstrated in Gilbert et al. [1988] transfers the constraints of Equation (13) on Δ_j. The theorem states that the barycentric coordinates for the vertices of a subset $W = \{s_j : j \in \kappa\}$ comply with Equation (13) if (i) $\Delta_j(W) > 0$ and (ii) $\Delta_i(W \cup s_i) \leq 0$ for all i in the complement of κ in $I = \{1, \ldots, m+1\}$. If all barycentric coordinates of the vertices in W are strictly positive, then all the subsets or cardinality $|W| + 1$ are inspected. If these have nonpositive barycentric coordinates the algorithm terminates, otherwise it continues.

The diagram in Figure 6 illustrates graphically the order in which the Johnson algorithm seeks the point of minimum norm $v(\tau)$ for a 2-simplex. The bottom-up arrows indicate that the method begins the search by inspecting the Voronoi regions associated to the vertices of the simplex as formulated in Equations (18) and (19).

More details on the formulation and the implementation of the Johnson subalgorithm may be found in Gilbert et al. [1988] and van den Bergen [1999]. It should be noted, however, that since the GJK algorithm updates the simplices by adding a single vertex at the time, only 2^m (rather than $2^{m+1} - 1$) Voronoi regions can possibly verify Equation (13). This simplifies the search and reduces the minimum number of operations, but it requires one to store all $\Delta_i(W)$, thus introducing overheads and making the data management within the Johnson algorithm more complex.

3.3 Numerical Instabilities

In practise the GJK algorithm tends to form degenerate simplices which have severe consequences on accuracy and performance of the algorithm itself. However, we observed that not all degenerate simplices imply failure. A degenerate m-simplex τ is the convex hull of a set of affinely dependent points in $\mathbb{R}^n$; our interest here is to study under which circumstances affinely dependent points cause instabilities to the Johnson distance subalgorithm.

This section presents two numerical tests using Algorithm 1 without the Backup procedure. It is well known that Johnson's procedure cannot accurately compute the point of minimum norm for a degenerate simplex [Gilbert et al. 1988]; however, disabling the Backup procedure allows one to cut down the CPU time [van den Bergen 2003] and to study more closely the effect of numerical instabilities on the GJK algorithm.

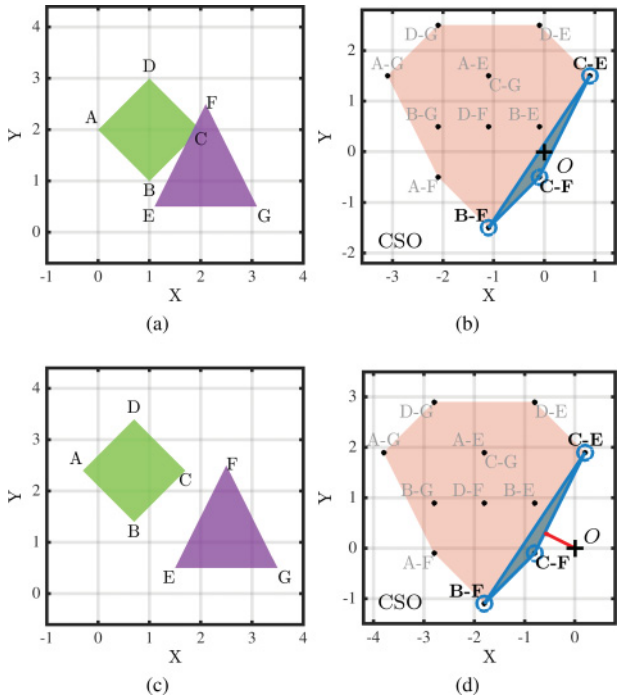


Fig. 7. Overlapping polygons (a) and distant polygons (c) with the highly deformed simplices generated upon termination of the GJK procedure; (b) and (d), respectively.

The first test considers two configurations for a pair of polygons: overlapping (Figure 7(a)) and distant (Figure 7(c)). To compute the minimum distance with the GJK algorithm we follow the same procedure illustrated in Figure 5. At the k-th iteration $v(\tau_k) = v(\text{CSO})$ and the GJK algorithm terminates. The particular orientation of the polygons however is such that the simplices τ_k result nearly-degenerate for both configurations. Figures 7(b) and 7(d) present the simplex for the overlapping and distant polygons, respectively.

The simplices in Figures 7(b) and 7(d) have the same nearly infinitesimal area but, since they have different sets of points supporting $v(\tau_k)$, the Johnson algorithm fails only for the overlapping configuration. The reason is that the subalgorithm converges when testing different Voronoi regions: $V_{(1,2,3)}$ for the overlapping polygons, and $V_{(1,2)}$ for the distant polygons. The difference between these Voronoi regions is that the first one is nearly degenerate, and the second one is not.

The second test presented in this section focuses on the algebraic system solved by the Johnson algorithm, and in particular on $|\det \mathbf{A}|$ to study the conditioning of Equation (14).

Let us consider the three-dimensional meshes of the gear teeth P and Q in Figure 8. The GJK algorithm measures, at each solution step, the distance between the teeth as they approach each other. Figure 9 shows the values of $|\det \mathbf{A}|$ and $d(P, Q)$ for consecutive simulation steps. The red markers indicate that the GJK algorithm, without Backup procedure, fails three times. The failures occur when $|\det \mathbf{A}|$ is in the order or the machine precision; in fact, they are a consequence of numerical instabilities in the distance subalgorithm.

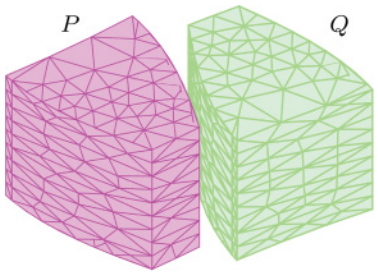


Fig. 8. Geometries used for the gear teeth benchmark.

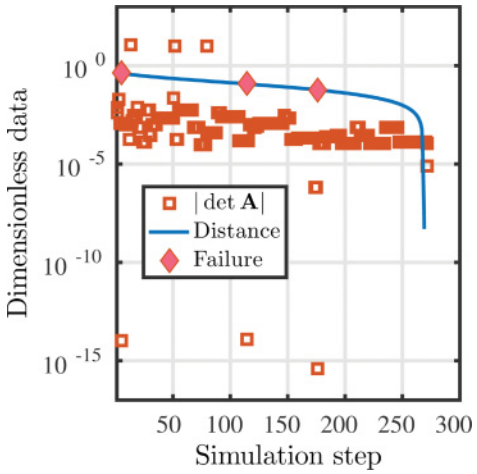


Fig. 9. Distance measurements for the test in Figure 8. |det **A**| is the determinant of the coefficient matrix assembled by the Johnson algorithm; when this value approaches the machine precision the GJK algorithm returns erroneous results (red marks).

From these two experiments, we conclude that numerical instabilities affecting the Johnson algorithm are not due to degenerate simplices themselves, but rather to the orthogonal condition embedded into Equation (14). The strategy adopted for evaluating Equation (17) indeed lacks robustness because the combination of dot products and differences amplifies the numerical error. When two vertices are at a distance from the origin, the difference will be inaccurate and the error amplified by the product. By performing these two operations on affinely dependent sets (i.e., degenerate simplices), the algorithm returns a determinant for **A** in the order of the rounding error (this issue was also noticed, but not addressed, in van den Bergen [2003]).

These observations on the numerical instabilities affecting the Johnson algorithm suggests that a simpler system of equations, which does not include products or differences, will retain accuracy even for degenerate simplices.

4. ENHANCED DISTANCE SUBALGORITHM

4.1 Method

This section presents a new distance subalgorithm that does not require the Backup procedure and circumvents the numerical instabilities affecting the Johnson algorithm. The idea is to build a system of equations, simpler than Equation (14), which does not involve multiplications of potentially small quantities.

The newly proposed procedure is named the *Signed Volumes method* because it evaluates the volume form $\mu(\tau)$ of an m-simplex τ and of other m *fictitious simplices* associated to it. $\mu(\tau)$ is the signed measure of length, area, or volume of the simplices in Figure 3 [Gallier 2011], whilst a fictitious simplex is obtained by substituting the origin O to a vertex of τ.

This method differs from the Johnson algorithm in the way it computes the points of minimum norm $v(\tau)$. Rather than solving a system of equations for each subset $W \subset \tau$ and to test whether Equation (13) is verified or not, our method identifies the unique set of points that satisfies Equation (13) and only then it solves a system of equations to compute the barycentric coordinates. This is done in three steps:

(1) Project the vertices of τ into a lower-dimensional space.
(2) Discard vertices, if any, not supporting $v(\tau)$.
(3) Solve $\mathbf{M}\boldsymbol{\lambda} = \mathbf{p}$ for the barycentric coordinates $\boldsymbol{\lambda}$.

The procedure relies on the fact that the barycentric coordinates are invariant to affine transformations, allowing one to compute $\boldsymbol{\lambda}$ in a lower-dimensional space $\mathbb{R}^r$, and to use these values in the space $\mathbb{R}^n$, for $r \leq n$, where both the simplex and CSO reside.

The dimension r of the reduced space $\mathbb{R}^r$ is sought recursively by projecting (Step 1) and discarding the vertices of τ (Step 2). This search is guided toward a space in which it is "safer" to compute the barycentric coordinates. By virtue of Carathéodory's theorem, only $r+1$ vertices of the simplex τ are needed in $\mathbb{R}^r$ to express the projection of $v(\tau)$; thus, our recursive search seeks the minimum set of vertices $W \subset \tau$ supporting $v(\tau)$ and, at the same time, gets rid of affinely dependent vertices.

Once the subset W is found, our method computes the barycentric coordinates λ_j (Step 3) for $r+1$ vertices in $W = \{s_j\}$. From Equation (6) we assemble an elementary system of equations for $r+1$ strictly positive λ_j: $\mathbf{M}\boldsymbol{\lambda} = \mathbf{p}$. In extended form this writes:

$$\begin{bmatrix} s_1^l & \dots & s_{r+1}^l \\ \dots & & \dots \\ 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \dots \\ \lambda_{r+1} \end{bmatrix} = \begin{bmatrix} p^l \\ \dots \\ 1 \end{bmatrix}. \tag{20}$$

The first rows of $\mathbf{M}$ contain the l-th coordinate of $\mathbf{s}_j$ and the last one consists of the $(r+1)$-dimensional unit vector. The column vector $\mathbf{p}$ contains the coordinates of the point obtained from the projection of the origin O onto the affine-hull of the points s_j, followed by 1. Since $r \leq 3$, a solution for this system can be efficiently computed using Cramer's rule.

The increased robustness provided by the Signed Volumes method over the Johnson algorithm is due to the different enforcement of the condition of minimum distance (Equation (4)). The former embeds it directly into the system in Equation (14); the latter transfers it to a phase that does not directly affect the computation of the barycentric coordinates. While the robustness of the Johnson algorithm depends on the mutual orthogonality of the faces supporting $v(\tau)$, the Signed Volumes method relies on the simple matrix $\mathbf{M}$ whose determinant is proportional to the volume form of τ:

$$\det \mathbf{M} = r!\,\mu(\tau). \tag{21}$$

The diagram in Figure 10 illustrates graphically how the Signed Volumes method seeks $v(\tau)$ for a 2-simplex. The top-down arrows indicate that the method begins the search by looking inside the simplex. This is in contrast to what the Johnson algorithm does (see Figure 6). Moreover, some tuples are disconnected and in grey colour to emphasise that these are excluded a priori from the recursive search. The Signed Volumes method selects for inspection

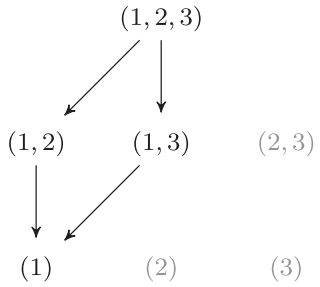


Fig. 10. The Signed Volumes method conducts a recursive search by excluding a priori the Voronoi regions that cannot possibly contain the origin.

2^m Voronoi regions, rather than $2^{m+1} - 1$, without any additional cached data.

4.2 Numerical Algorithm

The Signed Volumes method has a simple geometrical interpretation that makes its implementation straightforward. Algorithm 2 shows the main program from which three functions `S3D`, `S2D`, and `S1D` may be called. Apart from the trivial case that consists of a 0-simplex, each function is specific to an m-simplex and follows the three steps described in the previous section. Step 2 is, from the numerical point of view, the most challenging as it requires an accurate computation of Equation (21) and particularly of the sign of $\det \mathbf{M}$.

ALGORITHM 2: Signed Volumes Distance Subalgorithm

```
1:  procedure SignedVolumes (τ)
2:      τ has r + 1 vertices
3:      if r = 3 then
4:          [W,λ] = S3D(τ)
5:      else if r = 2 then
6:          [W,λ] = S2D(τ)
7:      else if r = 1 then
8:          [W,λ] = S1D(τ)
9:      else
10:         λ = 1, W = τ
        return W, λ
```

A vertex s_j can be discarded from an m-simplex if it resides in the opposite side of the hyperplane defined by the other s_k points, with $k \in \{1, \ldots, m+1\} - \{j\}$. In practise, this is done by comparing the signs of $\mu(\tau)$ and of other m fictitious simplices. The comparison is carried out by the `CompareSigns` function. To take into account rounding errors and null and NaN values, we define the function `CompareSigns` as follows:

$$\texttt{CompareSigns(a,b)} = \begin{cases} 1, & \text{if } a > 0, b > 0 \\ 1, & \text{if } a < 0, b < 0 \\ 0, & \text{otherwise.} \end{cases} \quad (22)$$

In order to minimise the number of operations, we observe from Equation (21) that $\mu(\tau)$ and $\det \mathbf{M}$ have the same sign. Therefore, rather than computing explicitly the volume form of the simplices, in practise we use $\det \mathbf{M}$ in Step 2, when calling `CompareSigns`, and in Step 3, when solving Equation (20) with Cramer's rule.

The functions `S3D`, `S2D`, and `S1D` called from Algorithm 2 are described in detail with a general three-dimensional example. Consider the tetrahedron in Figure 11(a). This simplex is generated by the GJK iterative search; that is, the vertex s_1 is added to the simplex

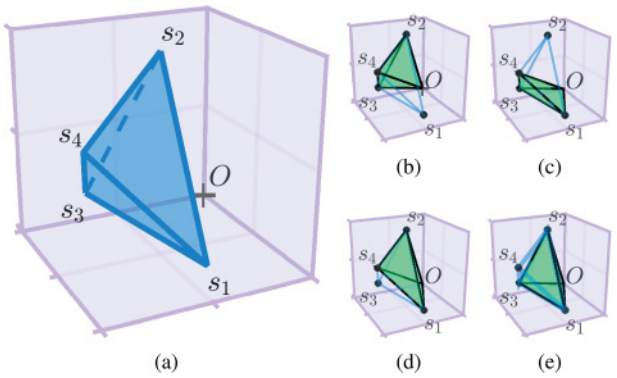


Fig. 11. From a 3-simplex (a) four fictitious simplices can be defined (b)-(e) by substituting the origin O to the vertices of the simplex.

consecutively to s_2, s_3, and s_4. We use the Signed Volumes method to compute the point of minimum norm $v(\tau)$ and to express it as a convex combination of the least number of vertices s_j, for $j \in \kappa$. Since τ has four vertices, $r = 3$ and `SignedVolumes` invokes `S3D` which is described below.

4.2.1 *Function `S3D`.* The function `S3D` computes the point of minimum norm $v(\tau)$ of a 3-simplex generated by the GJK iterative search. The simplex has 15 faces, each one associated to a Voronoi region; however, only the nine regions that can possibly contain the origin O will be inspected.

The input to this function is the ordered list of vertices $\{s_i\}$, for $i \in I = \{1, \ldots, m+1\}$, and the outputs are a subset of points W and the relative barycentric coordinates.

`S3D` takes only two of the three steps described in Section 4.1. In the first one, Step 2, the procedure tries to discard a vertex from τ using the function `CompareSigns`. Afterwards it solves $\mathbf{M}\boldsymbol{\lambda} = \mathbf{p}$.

For comparing the signs of the volume form $\mu(\tau)$ and the other m simplices, it is sufficient to compute $\det \mathbf{M}$, where

$$\mathbf{M} = \begin{bmatrix} s_1^x & s_2^x & s_3^x & s_4^x \\ s_1^y & s_2^y & s_3^y & s_4^y \\ s_1^z & s_2^z & s_3^z & s_4^z \\ 1 & 1 & 1 & 1 \end{bmatrix}. \quad (23)$$

The determinant of $\mathbf{M}$, after a cofactor expansion, writes:

$$\det \mathbf{M} = \sum_{\substack{i=4 \\ j=1,\ldots,4}} (-1)^{i+j} M_{i,j} = C_{4,1} + C_{4,2} + C_{4,3} + C_{4,4}. \quad (24)$$

In the preceding equation, $C_{i,j}$ is a *cofactor* and $M_{i,j}$ is a *first minor* of $\mathbf{M}$: the determinant of matrix obtained by removing the i-th row and j-th column from $\mathbf{M}$. The volume form of all fictitious simplices is proportional to the cofactors $C_{4,j}$. Geometrically these can be represented as the volume of the fictitious simplices depicted in Figures 11(b)–11(e).

The pseudocode in Algorithm 3 shows a possible implementation for computing $\det \mathbf{M}$ and $C_{4,j}$. Once these values are available, their signs are compared using the function `CompareSigns` to test whether a vertex can be discarded or not. If all signs are equal, the origin lays inside the simplex and thus all four vertices are supporting the point of minimum norm. In this case, the barycentric coordinates are given by $\lambda_j = C_{4,j} / \det \mathbf{M}$ for all j. Otherwise, the function `CompareSigns` has to be called for each $C_{4,j}$, with $j = \{2, 3, 4\}$, and the j-th vertex can be discarded if the output is 0.

The computation of the barycentric coordinates is then carried out by the function S2D. However, if this is invoked more than once, for example, if two or more fictitious simplices have sign different to the sign of det **M**, its outcomes must be compared to find which subset W returns the minimum norm $\|\boldsymbol{\nu}(W)\|$.

For the example in Figure 11 $\text{sign}(\det \mathbf{M}) \neq \text{sign}(C_{4,4})$, thus the algorithm discards s_4 and invokes the function S2D which is described below.

ALGORITHM 3: Subroutine for 3-simplex

```
1:  procedure S3D( {s1, s2, s3, s4} )
2:      for j = 1 : 4 do
3:          C_{4,j} = −1^{j+4} M_{4,j}
4:          det(M) = det(M) + C_{4,j}
5:      if CompareSigns(det(M), C_{4,j}) for all j then
6:          λ_j = C_{4,j} / det(M)
7:          W = {s1, s2, s3, s4}
8:      else
9:          for j = 2 : 4 do
10:             if CompareSigns(det(M), −C_{4,j}) then
11:                 [W*, λ*] = S2D({s_i : i ∈ {1, 2, 3, 4} − j})
12:                 d* = ‖ Σ_{i∈W*} λ*_i s_i ‖
13:                 if d* < d then
14:                     W = W*
15:                     λ = λ*
16:                     d = d*
        return W, λ
```

4.2.2 *Function S2D.* Similarly to the previous function, S2D computes the point of minimum norm $\nu(\tau)$ of a 2-simplex generated by the GJK iterative search. It receives in input an ordered list of three vertices $\{s_j\}$, with $j = \{1, 2, 3\}$, to express $\nu(\tau)$ as a convex combination of the smallest subset $W \subset \tau$.

S2D begins by projecting the origin O onto the affine hull of the vertices in $\{s_j\}$ to obtain the point p_O. To do so, we first compute vector $\mathbf{p}_O$:

$$\mathbf{p}_O = \frac{\mathbf{s}_1 \cdot \mathbf{n}}{\|\mathbf{n}\|^2}\,\mathbf{n}, \tag{25}$$

where $\mathbf{n}$ is the normal of the triangle: $\mathbf{n} = (s_2 - s_1) \times (s_3 - s_1)$. As shown in Figure 12(c), the vector $\mathbf{p}_O$ has direction $\mathbf{n}$ and length equal to the distance between the origin O and the affine hull of $\{s_j\}$.

To obtain the projected point p_O, we consider the vector $\mathbf{h} = \mathbf{p}_O - \mathbf{s}_1$ laying on the affine hull of $\{s_j\}$. The projection point is given by $s_1 + \mathbf{h} = s_1 + \mathbf{p}_O - \mathbf{s}_1 = s_1 + \mathbf{p}_O - s_1 + O = \mathbf{p}_O + O$. In fact, the projected point is $p_O = \mathbf{p}_O + O$.

In order to descend from $\mathbb{R}^3$ to $\mathbb{R}^2$, the point p_O and all the vertices of τ are projected onto the "safest" Cartesian plane. This is identified as the plane on which the simplex shades the largest area. For example, the triangle highlighted in Figure 12(a) can be projected on the Cartesian planes to define the other three triangles shown in Figure 12(b). The triangle with the largest area defines which Cartesian plane will be used for projecting p_O and s_i. The procedure projects and computes the areas of the fictitious simplices all at once by evaluating the minors $M_{1,4}$, $M_{2,4}$, $M_{3,4}$ of the matrix in Equation (23).

The pseudocode in Algorithm 4 shows an implementation of S2D. A for loop computes $\mu_{max} = \max\{|M_{1,4}|, |M_{2,4}|, |M_{3,4}|\}$ and the coordinate J that will be discarded to project the face and p_O onto a Cartesian plane. The projection of p_O is substituted to the projection of $\{s_j\}$ to set the j-th fictitious simplex, for $j = \{2, 3\}$. Afterwards, the signs of fictitious simplices and μ_{max} are compared using the usual function CompareSigns.

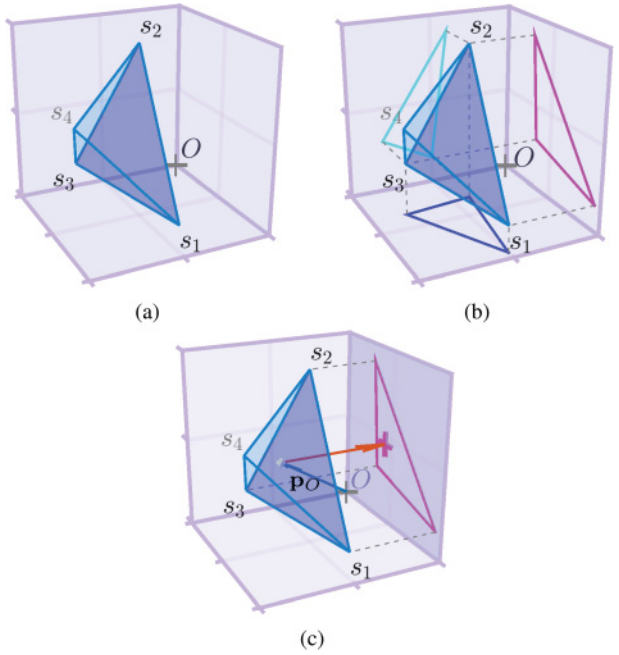


Fig. 12. The selected face of a 3-simplex (a) is projected onto the three Cartesian planes (b). The point p_O is projected on the plane onto which the face shades the largest area (c).

ALGORITHM 4: Subroutine for 2-simplex

```
1:  procedure S2D( {s1, s2, s3} )
2:      n = (s2 − s1) × (s3 − s1)
3:      p_o = (s1 · n) n / ‖n‖^2
4:      μ_max = 0, k = 2, l = 3
5:      for i = 1 : 3 do
6:          μ = s2^k s3^l + s1^k s2^l + s3^k s1^l − s2^k s1^l − s3^k s2^l − s1^k s3^l
7:          if | μ | > | μ_max | then
8:              μ_max = μ, J = i
9:          k = l, l = i
10:     Discard the J-th coordinate from {s_i} so that s_i = [s_i^x, s_i^y]
11:     k = 2, l = 3
12:     for j = 2 : 3 do
13:         C_j = (−1)^j (p_o^x s_k^y + p_o^y s_l^x + s_k^x s_l^y − p_o^x s_l^y − p_o^y s_k^x − s_l^x s_k^y)
14:         k = l, l = i
15:     if CompareSigns(μ_max, C_j) for all j then
16:         λ_j = C_j / μ_max
17:         W = {s1, s2, s3}
18:     else
19:         for j = 2 : 3 do
20:             if CompareSigns(μ_max, −C_j) then
21:                 [W*, λ*] = S1D({s_i : i ∈ {1, 2, 3} − j})
22:                 d* = ‖ Σ_{i∈W*} λ*_i s_i ‖
23:                 if d* < d then
24:                     W = W*
25:                     λ = λ*
26:                     d = d*
        return W, λ
```

For the 2-simplex in Figure 12(c), the system $\mathbf{M}\boldsymbol{\lambda} = \mathbf{p}$ is obtained by discarding the x-coordinate from the vertices of τ and p_O. In this

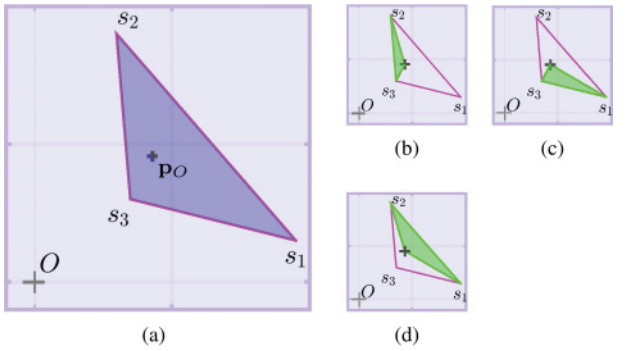


Fig. 13. From a 2-simplex (a) three fictitious simplices can be defined (b)–(d) by substituting the projection of p_O to the vertices of the simplex.

case the system of equations writes:

$$\begin{bmatrix} s_1^y & s_2^y & s_3^y \\ s_1^z & s_2^z & s_3^z \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} p_o^y \\ p_o^z \\ 1 \end{bmatrix}. \tag{26}$$

A view of the xy-plane is shown in Figure 13. Only two of the three fictitious simplices in Figures 13(b)–13(d) are tested to conclude that p_O lays inside the projected triangle. In fact, for this example, $C_{3,2}$, $C_{3,3}$, and $\det \mathbf{M}$ have the same sign. This proves that all vertices support the point of minimum norm $v(\tau)$ and that they are required to express it as a convex combination for the barycentric coordinates $\lambda_j = C_{3,j}/\det \mathbf{M}$ for all j.

For cases in which p_O lays outside the projected triangle, the function `CompareSigns` is invoked for each $C_{3,j}$ and the j-th vertex can be discarded if the output is 0. The computation of the barycentric coordinates is then carried out by the function `S1D`. However, if `CompareSigns` is called more than once, for example, if one or more fictitious simplices have sign different to $\det \mathbf{M}$, its outcomes must be compared to find which subset W returns the minimum norm $\|\boldsymbol{v}(W)\|$.

It should be highlighted that the robustness of this algorithm cannot possibly be compromised by degenerate simplices. If the triangle defined by $\{s_j\}$ is a needle (i.e., almost affinely dependent), the vector $\mathbf{n}$ will indeed suffer from cancellation and Equation (25) will result in an NaN value, but this pathological case is handled naturally by Algorithm 4: the NaN value will be passed to `CompareSigns` that triggers independent searches on selected subsets of τ. Each search is carried out by `S1D` which is described below.

4.2.3 *Function S1D.* The function `S1D` computes the point of minimum norm $v(\tau)$ of a 1-simplex generated by the GJK iterative search. The two vertices s_1 and s_2 in $\mathbb{R}^3$ are the input to this function. As for the other functions, the output is the smallest subset W of vertices which support $v(\tau)$.

`S1D` begins by projecting the origin O onto the affine hull of $\{s_1, s_2\}$ to obtain the coordinates of the vector $\mathbf{p}_O$:

$$\mathbf{p}_O = \mathbf{s}_2 + \frac{\mathbf{s}_2 \cdot \mathbf{t}}{\mathbf{t} \cdot \mathbf{t}} \mathbf{t}, \tag{27}$$

where $\mathbf{t} = (s_2 - s_1)$. A proof similar to the one in Section 4.2.2 demonstrates that the projection point is given by $p_O = O + \mathbf{p}_O$.

In order to descend from $\mathbb{R}^3$ to $\mathbb{R}^1$, the point p_O and all the vertices s_j are then projected onto the "safest" axis of the Cartesian coordinate system. This is identified as the axis on which the simplex shades the largest length.

ALGORITHM 5: Subroutine for 1-simplex

```
1:  procedure S1D( {s1, s2} )
2:      t = s2 − s1
3:      p_o = (s2 · t)/(t · t) t + s2
4:      μ_max = 0
5:      for i = 1 : 3 do
6:          μ = s1^i − s2^i
7:          if |μ| > |μ_max| then
8:              μ_max = μ, I = i
9:      Keep only the I-th coordinate from {s1, s2}
10:     k = 2
11:     for j = 1 : 2 do
12:         C_j = (−1)^j (s_k^I − p_O^I)
13:         k = j
14:     if CompareSigns(μ_max, C_j) for all j then
15:         λ_j = C_j / μ_max
16:         W = {s1, s2}
17:     else
18:         λ1 = 1, W = s1
        return W, λ
```

The pseudocode in Algorithm 5 shows an implementation of `S1D` for computing p_O and projecting it, with all the vertices of the simplex, onto an axis.

The example considered in the previous sections terminates without invoking `S1D`; however, the logic for computing the barycentric coordinates is identical to the one described for the other functions. If both vertices s_1 and s_2 support the point of minimum norm $v(\tau)$, that is, `CompareSigns`$(\det \mathbf{M}, C_{2,j}) = 1$, then $\lambda_j = C_{2,j}/\det \mathbf{M}$ for all $j = \{1, 2\}$. Where $C_{2,j}$ are the minors of a matrix $\mathbf{M}$ which simply includes the l-th Cartesian coordinate of the two vertices of the simplex:

$$\mathbf{M} = \begin{bmatrix} s_1^l & s_2^l \\ 1 & 1 \end{bmatrix}. \tag{28}$$

Otherwise, if p_O lays outside the projected line segment, $v(\tau)$ is supported by the vertex s_1 and $\lambda = 1$.

4.3 Implementation

Before discussing in detail the implementation of the Signed Volumes method, we wish to emphasise the difference between accuracy of the GJK algorithm and accuracy of the distance subalgorithm. The former has an arbitrary accuracy ε_{tol} specified by the user. Regardless of the value ε_{tol}, the GJK algorithm will achieve optimal convergence rates only if the distance subalgorithm computes the barycentric coordinates as accurately as possible (i.e., to machine precision). This will be demonstrated in the next section for a range of values ε_{tol}; here we discuss numerical aspects for a robust implementation of our method.

A crucial part of the Signed Volumes algorithm is the computation of $\det \mathbf{M}$; its sign, in particular. The pseudocodes in the previous section use methods which we consider good compromises between robustness and computational effort. However, there are alternatives that improve the accuracy of the sign of $\det \mathbf{M}$.

Adaptive floating-point arithmetic was exploited in Shewchuk [1996] and Ozaki et al. [2012] to exactly compute the sign of the signed volume of a tetrahedron and the sign of the signed area of a triangle. Shewchuk released the source code from which we extracted two routines `orient3d` and `orient2d` to compute the value and the exact sign of the determinant of $\mathbf{M}$ and its minors. These routines are an accurate, but slightly more computationally intense

alternative to the approach presented in Algorithms 3 and 4. We have tested a version which includes the functionality presented in Shewchuk [1996]; however, this brings only a modest improvement to Algorithms 3 and 4. In fact, despite the fact that the computation of $\det \mathbf{M}$ is crucial, most of the numerical error comes from Step 2 when projecting the origin. The next section presents numerical results obtained from our implementation of Algorithms 1–5.

5. RESULTS

This section presents numerical experiments to compare speed and accuracy of the GJK algorithm calling different distance subalgorithms. The tests involve both convex and nonconvex partitioned polytopes. Throughout this section, four distance subalgorithms will be considered:

—Johnson algorithm and Backup procedure (JB)
—Johnson algorithm only (JH)
—Backup procedure only (BK)
—Signed Volumes (SV)

All tests are carried out on a double precision Linux machine with Intel® Xeon® E5-2630 and 32GB RAM. The source code is compiled with the GNU compiler. Our implementation of the Signed Volumes method and GJK algorithm is written in ANSI C and is inspired from the source code provided by van der Bergen [van den Bergen 2003]. In this framework, the `CompareSigns` function is implemented as macro.

5.1 Accuracy

The *numerical accuracy* is a measure of the level of detail achieved when computing the distance between two objects. A requisite to ensure accuracy is robustness. Herein we repeat the gear teeth benchmark introduced in Section 3.3 to compare the performance of two distance subalgorithms.

The first test investigates the robustness of the JH and SV subalgorithms, with particular attention paid to the algebraic system solved for computing the barycentric coordinates. The determinant of the coefficient matrix is an indicator of the conditioning of the system itself. If the absolute value of the determinant is in the order of the rounding error, the system results are ill-conditioned.

Our test considers the gear teeth in Section 3.3 and monitors the values of $d(P, Q)$, $|\det \mathbf{A}|$, and $|\det \mathbf{M}|$, respectively: the distance between the objects, the determinant of $\mathbf{A}$ in Equation (14), and the determinant of $\mathbf{M}$ in Equation (20). The results obtained from the GJK algorithm using the JH subalgorithm are shown in Figure 14(a), whilst Figure 14(b) shows the results for the SV subalgorithm. The coloured bands include the upper and lower limits of the determinant of the matrices. The values of $|\det \mathbf{M}|$ span a range well above the machine precision; instead, $|\det \mathbf{A}|$ reaches ϵ several times. Finally, the JH subalgorithm underestimates $d(P, Q)$ at the last step, that is, before collision.

The test terminates when the approaching objects are found in contact, that is, when $d(P, Q)$ reduces its magnitude below ϵ. This is the case for SV but not for the JH subalgorithm. The large difference between the minimum value of $d(P, Q)$ in Figures 14(a) and 14(b) prove that, regardless of the value ε_{tol}, the accuracy of the GJK algorithm is influenced by the distance subalgorithm.

This test confirms that, unlike the Johnson algorithm, the Signed Volumes method cannot possibly generate systems of linearly dependent equations. In fact, degenerate simplices are naturally handled as described in Section 4.

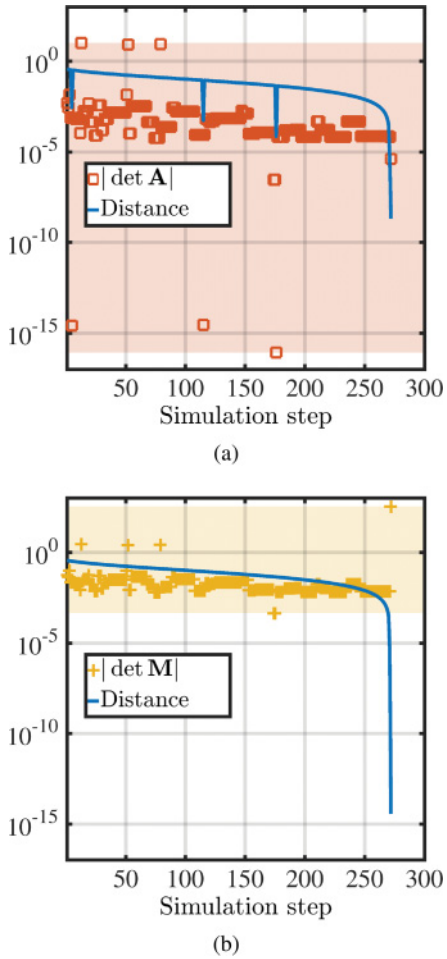


Fig. 14. Results for the gear teeth benchmark using two different distance subalgorithms. $\det \mathbf{A}$ and $\det \mathbf{M}$ are the determinants of the coefficient matrices associated to the Johnson subalgorithm (a) and Signed Volumes method (b), respectively.

By repeating the same test with different meshes, we verify that the accuracy achieved by the GJK algorithm is limited when using JH. Figure 15 shows the different levels of accuracy achieved by the two subalgorithms. The distance measured at the last solution step using JH is in the order of $\epsilon^{1/2}$, whereas SV reaches values very close to ϵ. This is due to the fact that at the last iteration the GJK algorithm generates a flat simplex that is close to the origin.

In some sense, degenerate simplices are intrinsic to the way the GJK algorithm solves distance queries. We observed that nearly flat objects are more likely to originate extremely deformed simplices. The reason is because a simplex takes the shape of the CSO; if the faces of the CSO closer to the origin are nearly flat, so will be the last simplex of the GJK procedure. However, the graph in Figure 15 proves that the SV subalgorithm is not compromised by such pathological cases.

5.2 Computing Time

The elapsed time from the call to the termination of the GJK algorithm is the most important measure of performance for real

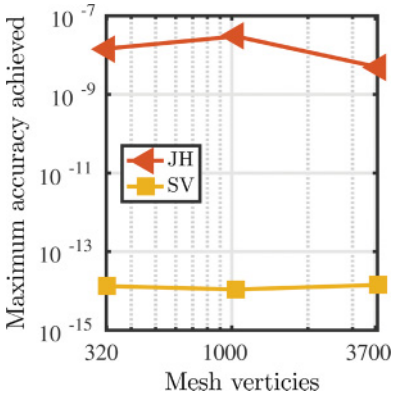


Fig. 15. Comparison of maximum accuracy achieved by the GJK algorithm using Johnson (JH) and Signed Volumes (SV) procedures for the gear teeth benchmark.

applications. The accuracy of the distance subalgorithm does affect the convergence rate of the GJK procedure, hence our aim is to assess the impact that the accuracy of a subalgorithm has on the CPU time required by the GJK procedure. Herein we look at the different subalgorithms JB, BK, and SV to compare their CPU time and the CPU time required by the GJK algorithm.

The geometries used for the following tests are simple polygonal spheres. Each query is repeated one million times and the CPU time, for both the GJK algorithm and the various subalgorithms, is the average between all runs. The reason for choosing simple spheres is purely numerical; this geometry reduces the changes of having to deal with degenerate simplices.

The CPU time is measured for values of the tolerance $10^{-8} \leq \varepsilon_{tol} \leq 10^{-14}$, with and without hill-climbing. The source code provided by Cameron [1997b] offers the basic capabilities to perform these tests. The code allows one to switch from JB to BK by defining the variable `TEST_BACKUP_PROCEDURE` and to adjust ε_{tol} by varying the variable `EPSILON`. Little coding is required to implement the exit conditions in Equations (9) and (10) and the SV subalgorithm.

Firstly, we measure the total CPU time required by the distance subalgorithm per GJK call. Figure 16 compares the total CPU time of the JB, BK, and SV subalgorithms for colliding, touching, and distant spheres. These results are obtained disabling hill-climbing and setting $\varepsilon_{tol} = 10^{-8}$. The same tests are repeated enabling hill-climbing and the measurements are shown in Figure 17. Apart from a few exceptions, all configurations have similar trends. As expected, the BK is the most computationally intense subalgorithm, whilst SV performs remarkably well. In fact, SV results are always faster or as fast as JB. The least improvement is measured for touching spheres, 5% on average; whilst the speedup achieved by SV over JB is between 10% and 25% for distant and overlapping spheres.

Figures 16 and 17 show that there are cases in which the accuracy of the subdistance algorithm plays a key role on the CPU time. For instance, Figure 17(a) reports an astonishing case[1] in which JB requires more CPU time than BK. Surprisingly, the CPU time is reduced by deactivating the Johnson algorithm and relying entirely on the Backup procedure, rather than using the usual combination of both algorithms. This is because the accuracy of the Johnson algorithm, as demonstrated in Section 5.1, is $\epsilon^{1/2}$ and not ϵ. An inaccurate computation of the search direction $\mathbf{v}$ leads the GJK algorithm to a path which is not optimal. Instead, SV and BK are accurate methods that compute $\mathbf{v}$ to machine precision and guide the GJK algorithm toward the shortest search path.

[1]The reader can verify this result by downloading the source code published by Cameron from http://www.cs.ox.ac.uk/stephen.cameron/distances and providing as input the following arguments: `-H -s577ccb17 1 400`.

Table I. All Combinations of Faces Supporting the Witness Points z_P and z_Q, and Resulting Supporting Faces of $\nu\,(\mathrm{CSO}) = \nu\,(\tau)$

$z_P \in$	$z_Q \in$	$\nu(\tau) \in$
Vertex	Vertex	Vertex
Vertex	Edge	Edge
Vertex	Polygon	Triangle
Edge	Edge	Triangle
Edge	Polygon	Triangle
Polygon	Polygon	Triangle

By repeating the same tests for $\varepsilon_{tol} = 10^{-14}$, we target applications that require solutions of distance queries accurate to machine precision. The CPU time required by different subdistance algorithms is presented in Figures 18 and 19; the latter regards tests exploiting frame coherence with hill-climbing. On average the SV outperforms all the other subalgorithms. Only for touching spheres its performance is comparable to that of JB. The speedup achieved by SV over JB is between 10% and 25% for distant spheres and between 15% and 30% for overlapping spheres.

On average, our distance subalgorithm reduces by 10% the total CPU time of the GJK algorithm. The CPU time for the previous tests is illustrated in Figures 20 and 21 for distant and overlapping spheres, respectively. Both figures show results using hill-climbing. The contour plots express the GJK CPU time as a function of two independent variables: accuracy ε_{tol} and number of vertices for each sphere. The graphs on the left use the JB subalgorithm, whilst the right ones use SV. All plots are in logarithmic scale and use the same colormap. From Figure 20 emerges a common pattern. As expected, the CPU time increases regularly as the mesh size increases. Moreover, $\varepsilon_{tol} = 10^{-10}$ appears to be a threshold value that identifies a region below which the CPU time increases. By comparing the two plots in that region we can observe a reduction of CPU time achieved by SV. Also, regardless of ε_{tol}, for 700 and above vertices in the mesh, GJK search conducted using JB results are more expensive.

The difference in CPU time is even more pronounced when the objects are found in contact (Figure 21). Under these circumstances, the SV subalgorithm improves the performance of the GJK algorithm by reducing its CPU time from a minimum of 5% to a maximum of 25%.

Let us now discuss the reasons behind the different performances of JB and SV. For overlapping objects the Signed Volumes is faster because it finds the solution at the very first inspected Voronoi region (see the diagrams in Figures 6 and 10). For distant configurations, we argue that SV is faster, or as fast as JH, because of geometrical and numerical reasons. Firstly, the CSO is given by the Minkowski difference $P - Q$ (Equation(3)) and its morphology is dictated by the mutual orientation of P and Q. A relationship exists between the face of the CSO supporting $\nu\,(\mathrm{CSO})$ and the faces supporting the witness points $z_P \in P$ and $z_Q \in Q$. Table I considers all possible combinations of faces (vertices, edges, and triangle) supporting a pair of witness points in physical space and

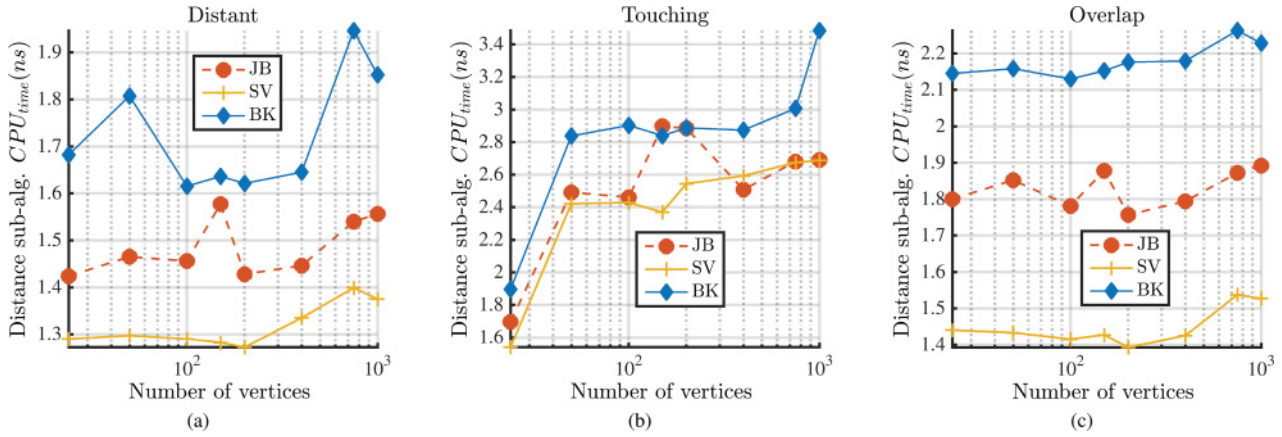


Fig. 16. Comparison of subalgorithms $CPU_{time}(ns)$ for $\varepsilon_{tol} = 1e{-}8$ without hill-climbing and for three configurations: (a) distant, (b) close, and (c) overlapping spheres.

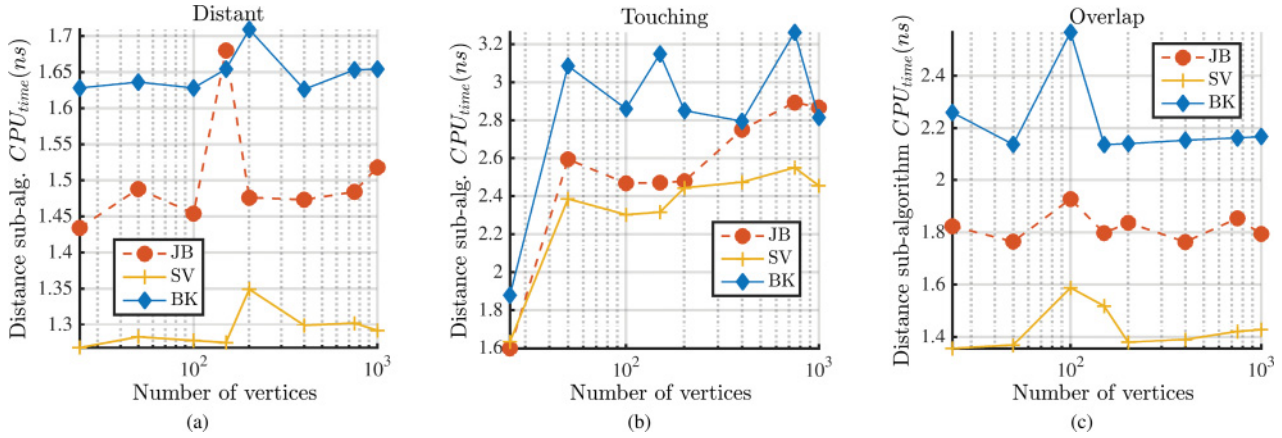


Fig. 17. Comparison of subalgorithms $CPU_{time}(ns)$ for $\varepsilon_{tol} = 1e{-}8$ exploiting hill-climbing and for three configurations: (a) distant, (b) close, and (c) overlapping spheres.

uses the Minkowski difference to recast this face in configuration space. As a result, the chances that the v (CSO) lays on a triangle of the CSO are four times higher than for any other face. The likelihood of finding v (CSO) on a triangle of a simplex suggests that the top-down search of the Signed Volumes method will return an answer in fewer operations. Secondly, the accuracy of the distance subalgorithm makes a bigger impact on meshes with large number of vertices. This is because on finer meshes a suboptimal search direction is more likely to compromise the evaluation of the support function. As a result, the support function provides the GJK with a simplex which is not directed toward the origin in the best possible way. Moreover, despite hill-climbing, larger meshes require more CPU time for evaluating the support function and thus, to minimise the GJK CPU time, is more important to optimise the GJK convergence rate by increasing the accuracy of the sub-algorithm, rather than minimise the number of operations with the risk of compromising the search path. Because the Signed Volumes method is more accurate than the Johnson algorithm, it reduces the CPU time by providing the GJK algorithm with an optimal search path.

A more elaborate implementation of the Signed Volumes method can improve its performance. This article presents an implementation that is simple and efficient in the sense that it inspects the least number of Voronoi regions. However, some of these regions might be inspected twice. This explains the modest speedup for touching configurations (Figures 16(b), 17(b), 18(b), and 19(b)). In fact, by looking at the tuple diagram in Figure 10 there are two arrows pointing at the region $V_{(1)}$, indicating that this region may be inspected twice. This phenomenon is even more pronounced for 3-simplices in $\mathbb{R}^3$. By enriching the cached data structure, it is possible to enforce the minimum number of inspections and save CPU time. Such a detailed implementation is, however, beyond the scope of this article and in the next section we present results for the same implementation adopted thus far.

5.3 Multiple Object Application

We now present the results of a finite element simulation involving nonconvex objects. A Stanford bunny model sits on top of a large rigid plane and eight knots fall from above. The bunny and the knots are deformable and discretised by solid tetrahedral finite elements. We use a leapfrog time integration scheme with a timestep in the order of $10^{-7}s$ to comply with the stability requirements [Cundall and Strack 1979]. In this simulation, the GJK algorithm is used

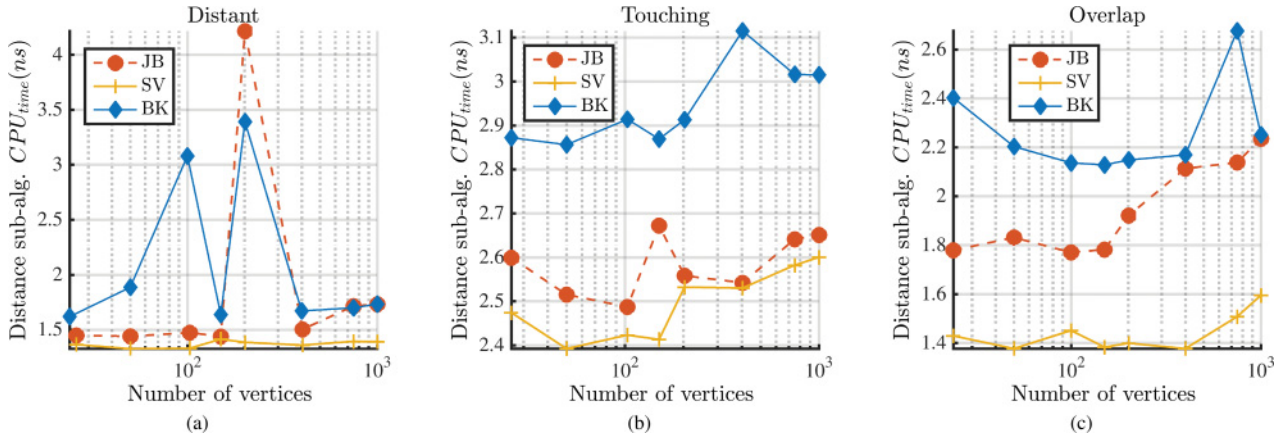


Fig. 18. Comparison of distance subalgorithms $CPU_{time}(ns)$ for $\varepsilon_{tol} = 1e{-}14$ without hill-climbing and for three configurations: (a) distant, (b) close, and (c) overlapping spheres.

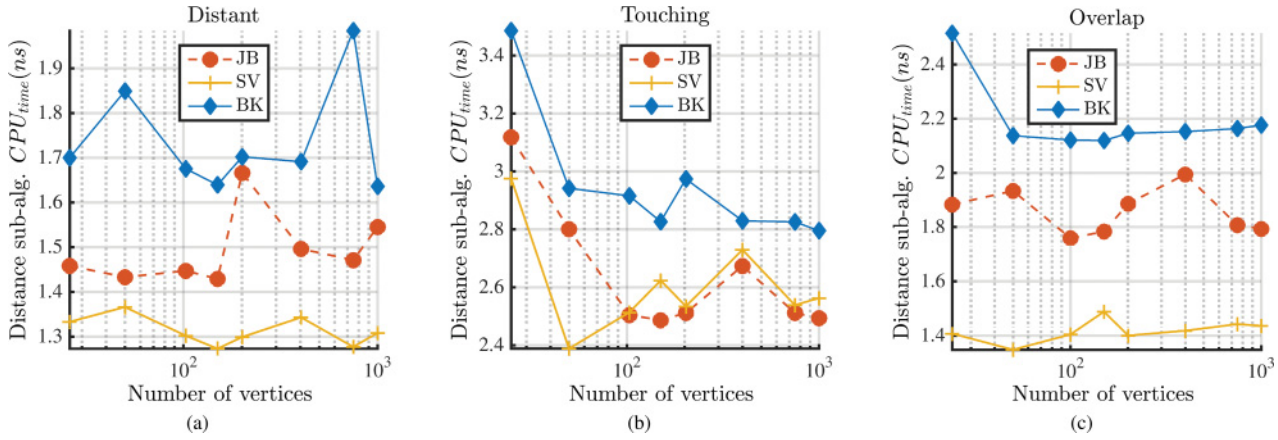


Fig. 19. Comparison for distance subalgorithm $CPU_{time}(ns)$ for $\varepsilon_{tol} = 1e{-}14$ exploiting hill-climbing and for three configurations: (a) distant, (b) close, and (c) overlapping spheres.

to perform a brute-force detect collision between the objects (each convex object is tested again every other object), whereas the computation of contact forces and internal stresses are carried out by the finite element solver.

The simulation is repeated for two levels of mesh refinement and using three different distance subalgorithms: BK, JB, and SV. The kinetic energy of the system is monitored during the simulations in order to verify that all experiments return the same values. A counter $N_{\texttt{SUB}}$ is incremented every time the distance subalgorithm is invoked. Similarly, $N_{\texttt{BACKUP}}$ is incremented every time the Backup procedure in invoked. The overall CPU time required by the subalgorithm is also measured.

Figure 22 shows a sequence of frames of the coarsest meshes; the model has approximately 4.5k outer facets. From the frame in Figure 22(a) to the frame in Figure 22(d) the simulation requires approximately 18 hours of CPU time, but the whole simulation lasts for about 31 hours. The results are presented in Table II. The CPU time required by the BK subalgorithm amounts to 63.7 minutes; however, the implementation using JB invokes the subalgorithm more often, around 322M times. Nearly 10% of the times the Johnson algorithm fails and the Backup procedure is invoked (30M times). When employing the SV subalgorithm, the GJK invokes `DistanceSubalgorithm` 299M times, reducing the number of calls by 7.4% compared to the JB. The overall CPU time spent by the JB subalgorithm is 59.2 minutes, while for SV this is reduced by 18%.

Table II. Measurements for the Test Case in Figure 22 with 4.5k Triangles. Total Simulation Time 31h and $\varepsilon_{tol} = 10^{-14}$

Distance sub-alg.	$N_{\texttt{SUB}}$	$N_{\texttt{BACKUP}}$	Sub-alg. CPU_{time} (min)
BK	297M	297M	63.7
JB	322M	30.0M	59.2
SV	299M	0	48.1

The same simulation is repeated using finer meshes which involve a total of 17.9k facets. For this scenario the improvement brought by the SV subalgorithm is even more significant. The measurements reported in Table III indicate that the CPU time spent by the distance subalgorithm is reduced by 20% when comparing SV to JB. This is a consequence of the fact that, for the same number of GJK calls, JB is invoked about 444M times, whilst SV is invoked only 380M times.

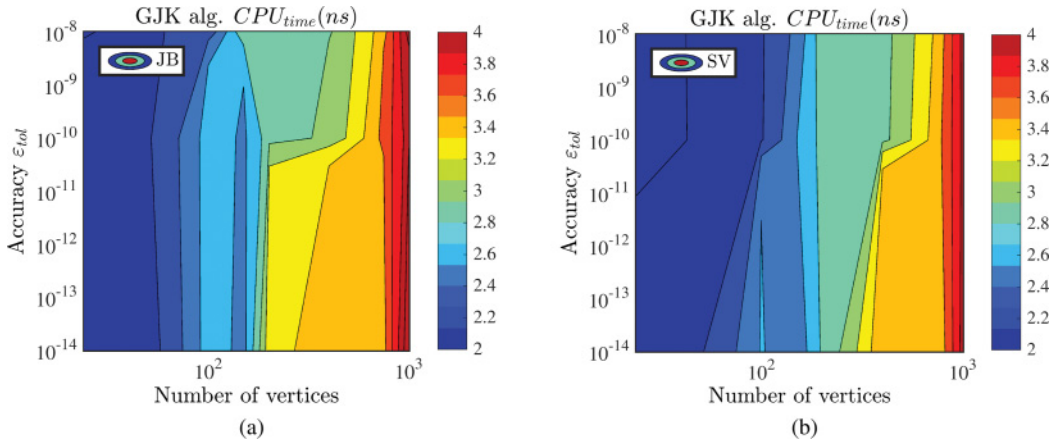


Fig. 20. Measurements of GJK CPU_{time} (ns) for distant polygonal spheres using the Johnson algorithm with Backup procedure (a) and the Signed Measures method (b).

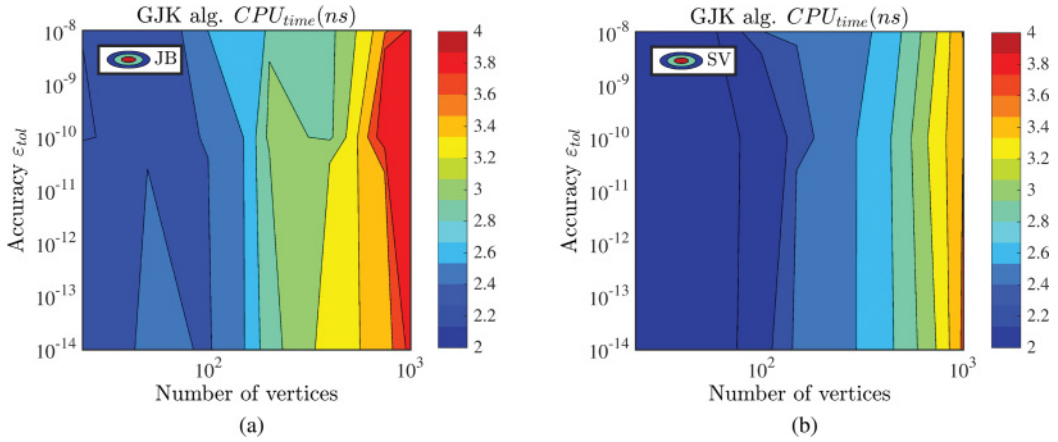


Fig. 21. Measurements of GJK CPU_{time} (ns) for overlapping polygonal spheres using the Johnson algorithm with Backup procedure (a) and the Signed Measures method (b).

Table III. Measurements for the Test Case in Figure 22 with 17.9 k Triangles. Total Simulation Time 129 h and $\varepsilon_{tol} = 10^{-14}$

Distance sub-alg.	N_{SUB}	N_{BACKUP}	Sub-alg. CPU_{time} (min)
BK	377M	377M	76.6
JB	444M	29.5M	73.1
SV	380M	0	58.1

6. CONCLUSIONS

This work shows counterexamples for the GJK algorithm that can lead to incorrect collision detection and addresses the shortcomings introduced by degenerate simplices.

Our study on the original subalgorithm concludes that the numerical instabilities are intrinsic to the algebraic system in the subalgorithm itself. The operators embedded in the algebraic system amplify the rounding error and compromise both barycentric coordinates and convergence rate.

Wary of this drawback, we replace the original Johnson subalgorithm and the Backup procedure with the novel Signed Volumes method. This is a recursive procedure designed to handle naturally degenerate simplices which relies on the sign of the volume form (e.g., length, area, volume) of the simplices to select the vertices supporting the point of minimum norm. The vertices thus selected are affinely independent and form a well-conditioned algebraic system from which we compute the barycentric coordinates.

The main difference between the Johnson and Signed Volumes subalgorithms is where the condition of minimum distance is imposed. The former embeds it into the linear system, and the latter transfers it to a phase that does not affect the computation of the point of minimum norm. The resulting algebraic system is simpler and less sensitive to rounding errors.

Numerical tests demonstrate that the Signed Volumes method effectively is a more robust procedure that reduces the computational time of the GJK algorithm. Our method does not require a Backup procedure, and yet it guides the GJK algorithm toward the shortest search path that results in fewer GJK iterations. Moreover, our tests show that the Signed Volumes method outperforms the original subalgorithms when the objects are found in contact. In such a configuration the GJK runs 5% to 25% times faster, whereas for other configurations it runs at least as fast as the original one or even

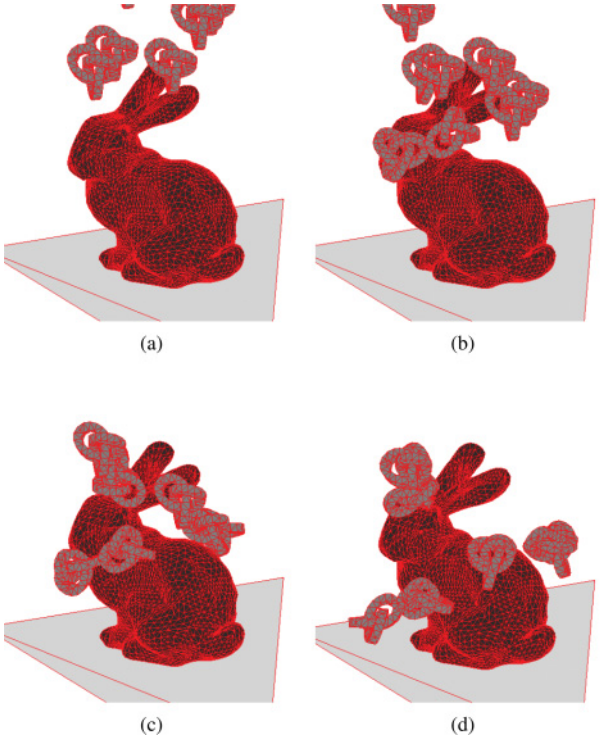


Fig. 22. Snapshots of the finite element simulation involving Stanford bunny and eight knots falling under gravitational load.

15% faster (depending on the prescribed tolerance and particularly on the regularity of the objects).

The improvements brought by this research are readily applicable to all existing implementations of the GJK algorithm. The Signed Volumes method can also be tailored to serve as a general-purpose algorithm for computing the distance between a point and a tetrahedron; moreover, it opens new possibilities for adopting the GJK algorithm in scientific computing. We illustrate an application for the FEM method. For other numerical frameworks such as meshless methods, the DEM method, and IGA, the GJK algorithm could efficiently solve distance queries for solid, shell, and truss elements as well as rigid particles, spheres, ellipsoids, and NURBS.

ACKNOWLEDGMENTS

The authors wish to acknowledge the helpful comments from the three anonymous reviewers.

REFERENCES

S. Cameron. 1997a. A comparison of two fast algorithms for computing the distance between convex polyhedra. *IEEE Transactions on Robotics and Automation* 13, 6 (Dec. 1997), 915–920.

S. Cameron. 1997b. Enhancing GJK: Computing minimum and penetration distances between convex polyhedra. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, Vol. 4.

S. A. Cameron and R. K. Culley. 1986. Determining the minimum translational distance between two convex polyhedra. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 591–596.

K. Chung and W. Wang. 1996. Quick collision detection of polytopes in virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*. 125–131.

H. S. M. Coxeter. 1989. *Introduction to Geometry*. Wiley.

P. A. Cundall and O. D. L. Strack. 1979. A discrete numerical model for granular assemblies. *Gotechnique* 29, 1 (Jan. 1979), 47–65.

A. Dax. 2006. The distance between two convex sets. *Linear Algebra and Its Applications* 416, 1 (2006), 184–213. DOI:http://dx.doi.org/10.1016/j.laa.2006.03.022 cited By 15.

M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. 2008. *Computational Geometry*. Springer, Berlin.

A. Dietrich, T. Wimbock, A. Albu-Schaffer, and G. Hirzinger. 2012. Reactive whole-body control: Dynamic mobile manipulation using a large number of actuated degrees of freedom. *IEEE Robotics & Automation Magazine* 19, 2 (June 2012), 20–33.

H. Edelsbrunner and E. P. Mucke. 1990. Simulation of simplicity. A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics* 9, 1 (1990), 66–104. DOI:http://dx.doi.org/10.1145/77635.77639 cited By 261.

C. Ericson. 2004. *Real-Time Collision Detection*. Morgan Kaufmann.

J. Gallier. 2011. *Geometric Methods and Applications*. Springer, New York.

E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. 1988. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation* 4, 2 (1988), 193–203.

R. A. Gingold and J. J. Monaghan. 1977. Smoothed particle hydrodynamics-theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society* 181 (1977), 375–389. http://adsabs.harvard.edu/full/1977MNRAS.181.375G

A. Haji-Akbari, E. R. Chen, M.l Engel, and S. C. Glotzer. 2013. Packing and self-assembly of truncated triangular bipyramids. *Physical Review E* 88, 1 (July 2013), 012127.

M. Held. 1998. ERIT—A collection of efficient and reliable intersection tests. *Journal of Graphics Tools* 2 (1998), 25–44.

T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs. 2005. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering* 194, 3941 (Oct. 2005), 4135–4195.

P. Jimnez, F. Thomas, and C. Torras. 2001. 3D collision detection: A survey. *Computers & Graphics* 25, 2 (April 2001), 269–285.

D. W. Johnson. 1987. *The Optimization of Robot Motion in the Presence of Obstacles*. Ph.D. dissertation. University of Michigan, Ann Arbor, MI. AAI8720285.

S. D. Laycock and A. M. Day. 2007. A survey of haptic rendering techniques. *Computer Graphics Forum* 26, 1 (March 2007), 50–65.

M. C. Lin and J. F. Canny. 1991. A fast algorithm for incremental distance calculation. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, Vol. 2.1008–1014.

A. Liu, F. Tendick, K. Cleary, and C. Kaufmann. 2003. A survey of surgical simulation: Applications, technology, and education. *Presence: Teleoperators and Virtual Environments* 12, 6 (Dec. 2003), 599–614.

T. Lozano-Perez. 1983. Spatial planning: A configuration space approach. *IEEE Transactions on Computers* 32, 2 (1983), 108–120. DOI:http://dx.doi.org/doi:10.1109/TC.1983.1676196

J. A. Millan, D. Ortiz, G. van Anders, and S. C. Glotzer. 2014. Self-assembly of Archimedean tilings with enthalpically and entropically patchy polygons. *ACS Nano* 8, 3 (March 2014), 2918–2928.

B. Mirtich. 1998. V-Clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics* 17, 3 (July 1998), 177–208.

N. Movshovitz, E. Asphaug, and D. Korycansky. 2012. Numerical modeling of the disruption of comet D/1993 F2 shoemaker-levy 9 representing the progenitor by a gravitationally bound assemblage of randomly shaped polyhedra. *The Astrophysical Journal* 759, 2 (2012).

K. Museth, D. E. Breen, R. T. Whitaker, S. Mauch, and D. Johnson. 2005. Algorithms for interactive editing of level set models. *Computer Graphics Forum* 24, 4 (Dec. 2005), 821–841.

B. Nye, A. V. Kulchitsky, and J. B. Johnson. 2014. Intersecting dilated convex polyhedra method for modeling complex particles in discrete element method. *International Journal for Numerical and Analytical Methods in Geomechanics* 38, 9 (June 2014), 978–990.

C.-J. Ong and E. G. Gilbert. 2001. Fast versions of the Gilbert-Johnson-Keerthi distance algorithm: Additional results and comparisons. *IEEE Transactions on Robotics and Automation* 17, 4 (Aug. 2001), 531–539.

K. Ozaki, T. Ogita, and S. Oishi. 2012. A robust algorithm for geometric predicate by error-free determinant transformation. *Information and Computation* 216 (2012), 3–13. DOI:http://dx.doi.org/10.1016/j.ic.2011.09.007. Special Issue: 8th Conference on Real Numbers and Computers.

J. N. Reddy. 2006. *An Introduction to the Finite Element Method*. McGraw-Hill Higher Education, New York, NY.

S. Redon, A. Kheddar, and S. Coquillart. 2002. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum* 21, 3 (Sept. 2002), 279–287.

L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. 2008. Larrabee: A many-core x86 architecture for visual computing. *ACM Transactions on Graphics* 27, 3, Article 18 (Aug. 2008), 15 pages.

J. R. Shewchuk. 1996. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry* 18 (1996), 305–363.

A. Tasora and M. Anitescu. 2011. A matrix-free cone complementarity approach for solving large-scale, nonsmooth, rigid body dynamics. *Computer Methods in Applied Mechanics and Engineering* 200, 5–8 (Jan. 2011), 439–453.

V. Tereshchenko, S. Chevokin, and A. Fisunenko. 2013. Algorithm for finding the domain intersection of a set of polytopes. *Procedia Computer Science* 18 (2013), 459–464. DOI:http://dx.doi.org/10.1016/j.procs.2013.05.209

C. Turnbull and S. Cameron. 1998. Computing distances between NURBS-defined convex objects. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, Vol. 4. 3685–3690.

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

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

A. Wachs, L. Girolami, G. Vinay, and G. Ferrer. 2012. Grains3D, a flexible DEM approach for particles of arbitrary convex shape Part I: Numerical model and validations. *Powder Technology* 224 (July 2012), 374–389.

R. Webster. 1995. *Convexity*. Oxford University Press, Oxford, England.

Y. Zheng, C. M. Chew, and A. H. Adiwahono. 2011. A GJK-based approach to contact force feasibility and distribution for multi-contact robots. *Robotics and Autonomous Systems* 59, 34 (March 2011), 194–207.

Y. Zheng and K. Yamane. 2013. Ray-shooting algorithms for robotics. *IEEE Transactions on Automation Science and Engineering* 10, 4 (Oct. 2013), 862–874.

Received January 2016; revised January 2017; accepted March 2017