# Classical contact detection algorithms for 3D DEM simulations: drawbacks and solutions

Yannick Descantes, Fabien Tricoire, Patrick Richard

***Accepted for publication in the journal Computers and Geotechnics (https://doi.org/10.1016/j.compgeo.2019.103134)***

# Classical contact detection algorithms for 3D DEM simulations: drawbacks and solutions.

Yannick Descantes[a,*], Fabien Tricoire[a], Patrick Richard[a]

[a]*IFSTTAR, MAST, GPEM, F-44340 Bouguenais, France*

**Abstract**

The present paper sheds new light on ongoing drawbacks of three classical algorithms dedicated to initial contact detection between overlapping convex polyhedra, (i) Cundall's common plane, (ii) Nezami's fast common plane and (iii) Gilbert-Johnson-Keerthi's algorithm (GJK). Solutions to these drawbacks are suggested and implemented into revised versions of those three algorithms, which are further benchmarked for accuracy and speed using nine overlapping contact situations. The benchmarking results show that the revised version of GJK, called $GJK-TD$, and Nezami (revised) return values of the contact normal components and overlap depth which are identical to machine precision, whereas Cundall (revised) results differ beyond the ninth decimal place. Furthermore, for a given contact situation, GJK-TD returns those values within a few tens of microseconds on average, whereas Nezami (revised) and Cundall (revised) are respectively 6 and 65 times more computationally intense. It is believed that the robustness and efficiency of GJK-TD will boost its use into DEM simulations, all the more that this versatile algorithm may easily be customized to detect contact between convex polyhedra and spheroid particles.

## 1. Introduction

Discrete Element Methods (DEM) are widely used techniques to perform numerical simulations of granular assemblies aiming at investigating their flow behaviour [1, 2] or packing properties [3, 4] to mention just a few. Among existing literature, most studies simulate granular assemblies of spherical particles [5, 6, 7, 1, 3], whereas non-spherical particles have only recently began to be simulated using ellipsoids [8], spherocylinders [9], tetrahedra [10], or other polyhedra [11, 12, 13, 14], despite their large presence in natural (*e.g.* ice debris, rocks) or manufactured state (*e.g.* sugar grains, crushed sand particles).

A well-known limitation of DEM applied to polyhedra is the lengthy calculation time needed to simulate large granular assemblies, say over a few tens of thousands particles, and still such assemblies only represent negligible quantities of granular materials (*e.g.* a 5 ml teaspoon already contains about 50000 powder sugar grains of size 0.5 mm). A significant share of this calculation time is dedicated to contacts detection. Indeed, though simple between two spheres - contact occurs when their centers are separated by a distance smaller than the sum of their radii - contacts detection between non-spherical particles is generally much more complex, hence more time-consuming. From the authors' experience, the calculation CPU time required for contacts detection between convex polyhedra generally lies in the range 10% to 20% of the total calculation CPU time. It may be lower for granular assemblies undergoing quasi-static strains, since new contact situations can generally be deduced from known contact situations at the previous time step. It may also reach higher values when simulating granular flows using highly faceted polyhedra.

The present paper focuses on initial contact detection between two overlapping convex polyhedra $A$ and $B$ having respectively $v_A$ and $v_B$ vertices, $e_A$ and $e_B$ edges, and $f_A$ and $f_B$ faces. Otherwise stated, we assume that the contact situation between these polyhedra has not previously been determined (e.g. during an earlier timestep). Initial contact

*Corresponding author
*Email address:* yannick.descantes@ifsttar.fr (Yannick Descantes)

detection is critical, since inappropriate detection may yield unacceptable overlap between the contacting particles that will be hard to eliminate during the simulation. Nevertheless it should be stressed that a small overlap between two contacting particles is often required by DEM methods in order to calculate repulsive contact forces (such a force is generally proportional to the overlap depth, e.g. see [15]). To avoid the lengthy testing of all interaction possibilities - $vertex/vertex$, $vertex/edge$, $vertex/face$, $edge/edge$, $edge/face$, $face/face$ -, a number of algorithms has been designed and published, namely in the frameworks of rock engineering [16, 17, 18], robotics [19] and computer graphics [20, 21]. Unfortunately, these algorithms have drawbacks which more or less strongly affect their use into DEM simulations.

The objectives of the present paper are to address the ongoing drawbacks of three of the most widely used algorithms dedicated to initial contact detection between overlapping convex polyhedra, and then to identify the most accurate and efficient revised algorithm among them.

The paper is organized as follows: section 2 reviews initial contact detection algorithms and highlights their drawbacks while suggesting solutions when possible. Then section 3 focuses on one of the most widely used algorithms and introduces a revised version which addresses its drawbacks. Eventually, section 4 benchmarks this revised algorithm against two other revised algorithms over a set of overlapping contact situations, in order to rank them in terms of accuracy and calculation speed.

## 2. Review and discussion of contact detection algorithms

Pionneering work by Cundall [16] has introduced the idea of a common plane ($CP$) separating two contacting polyhedra, which deflects to bisect the space between them. Basically, his idea is that the polyhedra are in contact with each other provided that each of them is in contact with the $CP$. Starting from the perpendicular bisector plane ($PB$) of the centroids of the particles (Fig 1), Cundall's algorithm consists in identifying the closest vertex of each particle to this plane, then translating the plane to the mid-point of the segment linking the identified closest vertices, and finally perturbating step by step the orientation of the unit normal of this plane until the gap between the closest vertex and the plane is maximum. Note that the gap may take negative values to account for situations in which particles overlap the $CP$ - hence particles overlap -, so that maximizing the gap means minimizing the overlap and closest vertex means most deeply burried into the plane. Overall, Cundall's algorithm detects contacts between two particles and, in this case, yields the coordinates of contact point(s) and $CP$ normal vector as well as gap width, all this with a reported complexity of order $O(n_{iter}.(v_A + v_B))$, where $n_{iter}$ denotes the number of perturbation steps.
In fact, a large perturbation step improves the algorithm complexity by decreasing the number of perturbation steps required to maximize the gap, but this is done at the expenses of the $CP$ orientation accuracy and may hence yield significant unwanted interpenetration. More embarrassing, depending on the shape and relative location of the contacting particles, initializing Cundall's algorithm with the $PB$ of the particle centroids may end up with an erroneous $CP$ as shown on Fig. 2, insofar as the common plane fails to minimize the overlap between the particles. This problem is caused by the limited range of $CP$ normal orientations explored when applying Cundall's perturbations, thus leading to a local maximum of the gap rather than the global one.
To overcome this problem, a simple solution consists in repeating Cundall's algorithm three times, first upon initializing the algorithm with the $PB$ of the centroids and then with two orthogonal planes which are perpendicular to the $PB$ and intersect at the centroids mid-point. Among the calculated $CP$ candidates, the one which minimizes the overlap should eventually be retained. Naturally, this solution triples the algorithm complexity from existing one. The revised Cundall algorithm is summarized as algorithm 1.

Nezami and coworkers [17] have sped up Cundall's algorithm by observing that the number of common plane candidates is finite. Their algorithm begins similarly by (step 1) finding the perpendicular bisector plane of the centroids of the particles and (step 2) identifying the closest vertex of each particle to this plane. In case several vertices minimize the distance of a particle to the plane, the authors state that any of them may be chosen. Then (step 3), the $CP$ is sought among the $PB$ of the closest vertices and the planes parallel to one of the edges or faces of each particle which pass through its closest vertex, the latter planes being further translated to intersect at the midpoint of the closest vertices (see Fig. 3). The set of closest vertices of the particles to each $CP$ candidate is identified, and, among all these,

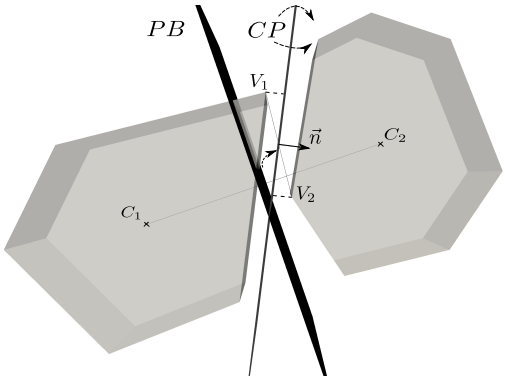



Figure 1: 3D-contact detection using the common plane strategy [16]. The perpendicular bisector plane ($PB$) of the particle centroids $C_1$ and $C_2$ allows to identify the closest vertices $V_1$ and $V_2$. The $PB$ is further translated to the mid-point of the $[V_1; V_2]$ segment and rotated until maximisation of the plane-particle gap ($gap = \overrightarrow{V_1V_2}.\vec{n}$) to achieve the common plane ($CP$).

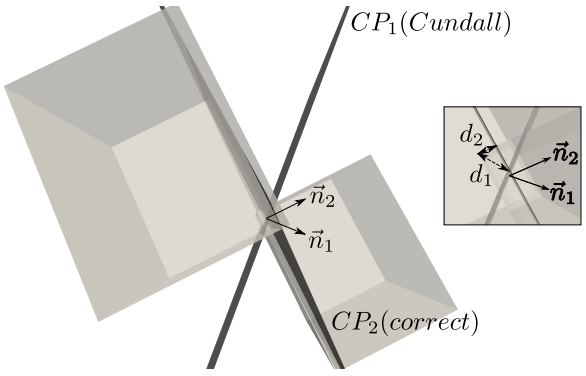



Figure 2: 3D-common plane candidates $CP_1$ (Cundall) and $CP_2$ (correct) between two parallelepiped interpenetrated by an edge. Interpenetration distance $d_1$ to $CP_1$ is significantly larger than interpenetration distance $d_2$ to the correct common plane (see inset). Note that corresponding normals $\vec{n}_1$ and $\vec{n}_2$ are also significantly different.

**Algorithm 1** Revised Cundall algorithm

**Require:** Orthonormed frame ($\vec{n}_{12}$, $\vec{p}_{12}$, $\vec{q}_{12}$) is known, with $\vec{n}_{12}$ and mid-point of particle centroids defining their $PB$, $CP$-candidate initialized to $PB$, $gap_{max}$ and orientation perturbation step range initialized

$gap_{max} \leftarrow -1000.$
$[k_{min}; k_{max}] \leftarrow [\tan(10^{-7}); \tan(5^{\circ})]$
**for** $\vec{n}$ in $\{\vec{n}_{12}, \vec{p}_{12}, \vec{q}_{12}\}$ **do**
  **while** closest vertices to $CP$-candidate change **do**
    Find closest vertex of each particle to $CP$-candidate along $\vec{n}$
    Translate the $CP$-candidate to mid-point of these vertices
    Initialize $\vec{p}_1$ and $\vec{q}_1$ so that ($\vec{n}$, $\vec{p}_1$, $\vec{q}_1$) is an orthonormed frame
    **for** ($\vec{p}$,$\vec{q}$) in $\{(\vec{p}_1,\vec{q}_1);(rot_{\pi/4}(\vec{p}_1),rot_{\pi/4}(\vec{q}_1))\}$ **do**
      $k \leftarrow k_{max}$
      **while** $k \geq k_{min}$ **do**
        $\vec{n}_{1temp} \leftarrow (\vec{n} + k\vec{p})/(1 + k^2)$
        $\vec{n}_{2temp} \leftarrow (\vec{n} - k\vec{p})/(1 + k^2)$
        $\vec{n}_{3temp} \leftarrow (\vec{n} + k\vec{q})/(1 + k^2)$
        $\vec{n}_{4temp} \leftarrow (\vec{n} - k\vec{q})/(1 + k^2)$
        **for** $i = 1$ to $4$ **do**
          Find closest vertex of each particle to $CP$-candidate along $\vec{n}_{itemp}$
          Calculate interparticle $gap_i$ along $\vec{n}_{itemp}$
        **end for**
        **if** $max(gap_i) > gap_{max}$ **then**
          Update closest vertices
          $\vec{n} \leftarrow \vec{n}_{itemp}$
          $gap_{max} \leftarrow max(gap_i)$
        **else**
          $k \leftarrow k/2$
        **end if**
      **end while**
    **end for**
  **end while**
**end for**

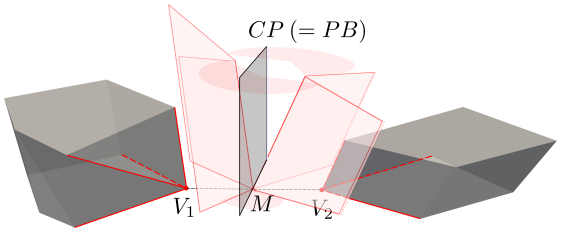



Figure 3: Two polyhedra and some of their common plane candidates ($CP$) according to [17]. $CP$ candidates are the perpendicular bissector plane ($PB$) of the closest vertices $V_1$ and $V_2$ of the polyhedra identified during the previous iteration as well as planes which are parallel to at least one edge (highlighted in bold red/dark color) of a polyhedron that contains either $V_1$ or $V_2$. Note that each $CP$ candidate passes through point $M$, the midpoint of $V_1$ and $V_2$. Here, the actual common plane ($CP$) coincides with the $PB$ of vertices $V_1$ and $V_2$.

the $CP$ candidate achieving the maximum distance between its set of closest vertices is retained. Finally (step 4), the retained set of closest vertices is compared to the previous one. If the two sets of closest vertices are equal, then the $CP$ has been obtained, otherwise the algorithm iterates from step 3 using the last set of closest vertices and their $CP$ candidate. Note that unlike Cundall's algorithm, Nezami's algorithm requires that no particle is in contact with the $CP$ candidate prior to applying step 3. In case such a contact exists, the authors prescribe to translate both particles perpendicular to this plane and away from it to achieve a positive and small gap. Obviously, the opposite translation is applied to the particles once the $CP$ has been identified. According to Nezami and coworkers, their algorithm yields the coordinates of contact points and common plane normal vector as well as overlap distance with a complexity of order smaller than $O(v_A + v_B)$.
However, as shown in Fig. 4, the translation prescribed by Nezami in case of an existing contact between the $CP$ candidate and a particle prior to applying his algorithm may yield an erroneous $CP$. In fact, such an error occurs when the most deeply penetrated vertex of a particle into the other particle changes with the translation direction. In Fig. 4, $V_2$ is the most deeply penetrated vertex into *particle* 1 along direction $\vec{n}_1$, hence it becomes the closest vertex to the $CP$ candidate once the translation along $\vec{n}_1$ is performed and $CP_1$ is found to be the common plane. Nevertheless, $V_2'$ is the most deeply penetrated vertex into *particle* 1 along direction $\vec{n}_2$ and this latter direction minimizes the overlap, hence $CP_2$ is the correct common plane and it should be observed that both $CP$ candidates differ significantly.
Steps 2, 3 and 4 of Nezami's algorithm only require minor modifications to appropriately account for overlapping particles without translating them. In case of a negative gap and as suggested by Cundall [16], the closest vertices simply need to be redefined as the most deeply burried into the plane and the $CP$ as the plane which minimizes the overlap, hence maximizes the algebraic distance between the sets of most deeply burried vertices (or equivalently minimize their absolute distance). The revised Nezami algorithm is summarized as algorithm 2.

Other algorithms are focused on finding the closest features (point, edge or face) between convex polyhedra $A$ and $B$. Among these, the GJK algorithm [20] was designed upon observing that finding the closest points between non-contacting polyhedra $A$ and $B$ is equivalent to finding the point of another convex set, consisting of their Minkowski difference $A - B$ ($A - B = \{x_A - x_B,\ x_A \in A,\ x_A \in B\}$), which minimizes the distance to the coordinates origin $O$. Interestingly, the coordinates origin always lies outside the outer boundary of the Minkowsky difference $A - B$ of non-contacting polyhedra, so that the point minimizing the distance of $A - B$ to the coordinates origin may be redefined as the point $V$ of the outer boundary of $A - B$ located closest to the coordinates origin and $\|\vec{OV}\|$ is the interparticle distance. When the polyhedra are in contact or overlap, the coordinates origin is located respectively on or inside the outer boundary of the Minkowski difference $A - B$, so that the point $V$ of the outer boundary of $A - B$ located closest to the coordinates origin defines both the overlap depth $-\|\vec{OV}\|$ (negative distance) and contact normal $\vec{OV}$ of the overlapping polyhedra (pointing from polyhedron $A$ to polyhedron $B$).

To find the point $V$ whilst avoiding the lengthy calculation of the entire Minkowski difference, the algorithm builds a sequence of simplices whose distance to the coordinates origin decreases monotonically to achieve the intended

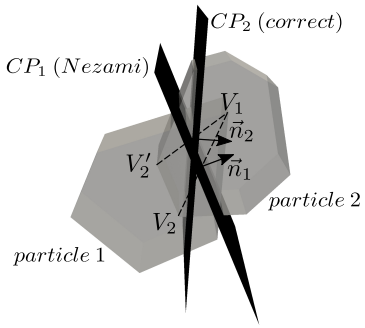



Figure 4: Common plane according to ref. [17] ($CP_1$, normal $\vec{n}_1$) and correct common plane ($CP_2$, normal $\vec{n}_2$) between two interpenetrated polyhedra. $\vec{n}_2$ is the direction of minimum relative translation to remove any interpenetration between the particles. $V_1$ and $V_2$ denote the closest vertices of particles 1 and 2 respectively to $CP_1$, while $V_1$ and $V_2'$ denote the closest vertices of these particles to the correct common plane ($CP_2$).

**Algorithm 2** Revised Nezami algorithm

**Require:** Normal $\vec{n}_{12}$ is known, with $\vec{n}_{12}$ and mid-point of particle centroids defining their $PB$, $CP$-candidate initialized to $PB$

Find vertices $V_1$ and $V_2$ of particles located closest to $CP$-candidate along $\vec{n}_{12}$
Calculate corresponding $gap$
$gap_{max} \leftarrow gap$
Initialize $\vec{n}_{12save}$ so as to enter the loop
**while** $\vec{n}_{12} \neq \vec{n}_{12save}$ **do**
  $\vec{n}_{12save} \leftarrow \vec{n}_{12}$
  Determine mid-point $MP$ of vertices $V_1$ and $V_2$
  $\vec{n}_{12} \leftarrow \overrightarrow{V_1V_2}/\|\overrightarrow{V_1V_2}\|$
  Find all faces $F_1$ and edges $E_1$ of particle 1 comprising vertex $V_1$
  Find all faces $F_2$ and edges $E_2$ of particle 2 comprising vertex $V_2$
  **for all** couples of edges $E_1$ and $E_2$ **do**
    **if** $E_1$ non-parallel to $E_2$ **then**
      Determine plane $P_{12}$ passing through $PM$ and defined by $E_1$ and $E_2$
    **end if**
  **end for**
  **for all** edges $E_1$ **do**
    Determine plane $P_1$ passing through $MP$ and parallel to both $E_1$ and cross product $(E_1; \vec{n}_{12})$
  **end for**
  **for all** edges $E_2$ **do**
    Determine plane $P_2$ passing through $MP$ and parallel to both $E_2$ and cross product $(E_2; \vec{n}_{12})$
  **end for**
  **for all** planes $PB$, $F_1$, $F_2$, $P_{12}$, $P_1$, $P_2$ **do**
    Find vertices $V_1$ and $V_2$ of particles located closest to plane along its normal $\vec{n}_{plane}$
    Calculate interparticle $gap$ along $\vec{n}_{plane}$
    **if** $gap > gap_{max}$ **then**
      Update closest vertices
      $\vec{n}_{12} \leftarrow \vec{n}_{plane}$
      $gap_{max} \leftarrow gap$
    **end if**
  **end for**
**end while**

minimum. Here, a simplex is a convex feature, e.g. a vertex, segment, triangle or tetrahedron, at most comprising $r+1$ affinely independent vertices of the outer boundary of the Minkowski difference in a $r$-dimensional space. Given a simplex $\tau_k$ at the end of the $k$-th iteration, the subsequent simplex $\tau_{k+1}$ is obtained by addition of a new vertex $w_{k+1}$ so that $\tau_{k+1}$ is located closer to the coordinates origin than $\tau_k$, thus building iteration after iteration the closest simplex to the coordinates origin. Note that if $\tau_k$ already contains $r+1$ affinely independent vertices, then the vertex not belonging to the segment or the triangular face located closest to the coordinates origin respectively for $r=2$ and $r=3$ is deleted. Practically, $w_{k+1}$ is calculated as the difference between the outermost vertex of $A$ in the oriented direction $\vec{v}_k$, which minimizes the distance from previous simplex $\tau_k$ to the origin and points towards the origin, and the outermost vertex of $B$ in the opposite direction (Fig. 5). The GJK algorithm stops either when the origin lies inside the new simplex $\tau_{k+1}$ or in the absence of a new vertex $w_{k+1}$ generating a closer simplex to the coordinates origin. Last, the coordinates of the point $V_{k+1}$ of the final simplex $\tau_{k+1}$ located closest to the coordinates origin are calculated using the Johnson algorithm [22]. This algorithm recurses on the subsets of the final simplex, starting with those having the smallest number of vertices (e.g. vertices, then segments, then triangular faces, then tetrahedron) until one contains the point $V_{k+1}$. Upon calling $S_1$ to $S_p$ the $p$ vertices of this subset, $V_{k+1}$ is uniquely defined as the barycentre of vertices $S_1$ to $S_p$ which verifies the following equations:

$$\sum_{i=1}^{p} \lambda_i = 1 \quad \text{and} \quad \forall\, i \in \{1, ...p\} \quad \lambda_i > 0, \tag{1}$$

$$\sum_{i=1}^{p} \lambda_i \vec{OS}_i.(\vec{OS}_j - \vec{OS}_1) = 0 \quad \text{for } j \in \{2, ..., p\}, \tag{2}$$

with $\vec{OV}_{k+1} = \sum_{i=1}^{p} \lambda_i \vec{OS}_i$.

Observe that equations (1) and (2) provide a small system of the form $\boldsymbol{M}\lambda = \vec{L}$ with $p$ equations ($p \leq 4$) and $p$ unknowns $\lambda_i$, which may be inverted to determine the $\lambda_i$ values and calculate both the distance $\|\vec{OV}_{k+1}\|$ of $V_{k+1}$ to the origin and the orientation of $\vec{OV}_{k+1}$. Furthermore, note that if the particles are not in contact, $V_{k+1}$ is also the closest point of the Minkowsky difference $A-B$ to the origin and $\|\vec{OV}_{k+1}\|$ the distance between the two polyhedra. Conversely, when the polyhedra are in contact or overlap, $\|\vec{OV}_{k+1}\|$ denotes the overlap depth and $\vec{OV}_{k+1}$ defines the contact normal. According to Gilbert and coworkers [20], their algorithm has a complexity of order $O(v_A + v_B)$.

Despite its widespread use due to its low computational costs and high versatility, the GJK algorithm was reported to suffer numerical unstabilities resulting from rounding errors [24, 23]. These rounding errors would either cause the simplex generation algorithm to loop forever [24] or the Johnson algorithm to be numerically unstable as the combination of dot products and differences may yield ill-conditionned system of equations (1) and (2). As an example, ill-conditioning would occur when the simplex generation algorithm converges towards a degenerate simplex, in other words a simplex comprising vertices which are nearly affinely dependent such as an elongated triangular face having two vertices lying very close to each other. This situation may be encountered when a large flat face of a polyhedron interacts with another polyhedron smaller by several orders of magnitude [24], a situation that may be somewhat frequent in DEM simulations of convex polyhedra assemblies.

Solutions to circumvent GJK issues have been suggested and tested with reasonable success by several authors [24, 25, 26, 23], starting with Gilbert et al themselves [20] who designed a *backup procedure* to cache the above-mentioned dot products which would otherwise be calculated several times by the Johnson algorithm with the risk of rounding errors being introduced. This *backup procedure* was further improved by Van den Bergen [24] who suggested a caching algorithm which minimizes the caching overhead. This author also sped up the GJK algorithm by designing an early termination procedure to detect non-contacting particles, focused on detecting the existence of a separating axis between them. More recently, Montanari et al [23] have suggested the *signed volume method* to circumvent the numerical instabilities of the Johnson algorithm. The authors' idea is to disembed the determination of the simplex closest to the coordinates origin from the calculation of the closest point coordinates. Once the GJK algorithm has converged to the simplex located closest to the coordinates origin, their method consists in (step 1) identifying the smallest set of vertices of this simplex which contains the closest point to the coordinates origin and (step

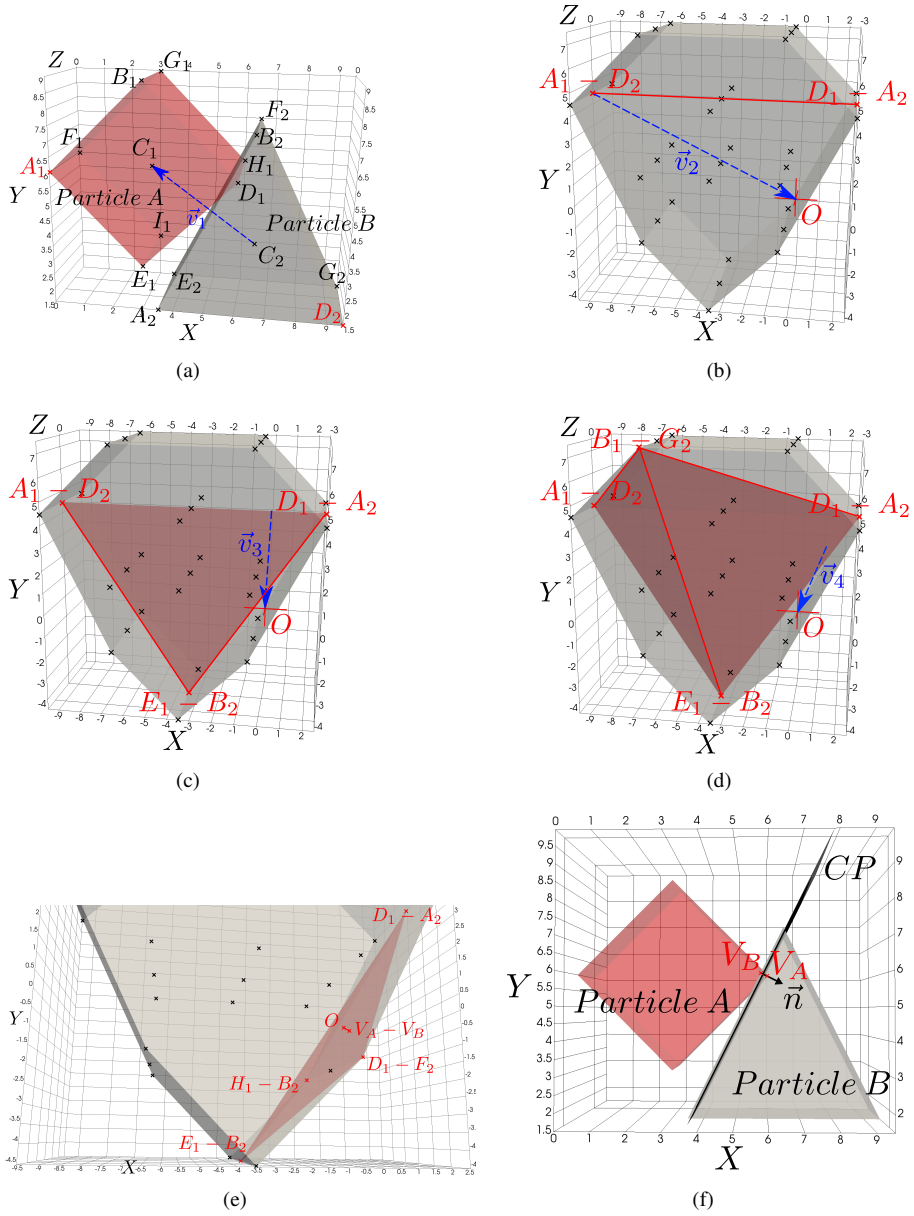



Figure 5: Principle of the GJK algorithm [20] demonstrated using **(a)** Overlapping parallelepiped and prism generalized from 2D Fig. 7*a* of ref. [23] (resp. particle $A$ with centroid $C_1$ and eight cross-marked vertices $A_1$ to $I_1$, and particle $B$ with centroid $C_2$ and six cross-marked vertices $A_2$ to $G_2$), with $A_1$ the outermost vertex of particle A in the oriented direction $\vec{v}_1$ and $D_2$ the outermost vertex of particle B along $-\vec{v}_1$. **(b)** The difference of preceeding vertices $A_1 - D_2$ belongs to the convex hull of the Minkowsky difference of particles $A$ and $B$ and defines initial simplex $\tau_1$ as well as new oriented direction $\vec{v}_2$ pointing from $A_1 - D_2$ to coordinates origin $O$, along which outermost points of particles $A$ and $B$ define new point $D_1 - A_2$ of the convex hull (fully represented here in light-grey together with crosses representing its vertices). **(c)** $\vec{v}_3$ is the oriented direction minimizing the distance between new simplex $\tau_2$ containing two points $A_1 - D_2$ and $D_1 - A_2$ and coordinates origin, along which outermost points of particles $A$ and $B$ define new point $E_1 - B_2$ of the convex hull. **(d)** Following the same principle, $\vec{v}_4$ is the oriented direction minimizing the distance between new simplex $\tau_3$ containing three points $A_1 - D_2$, $D_1 - A_2$ and $E_1 - B_2$, and coordinates origin, and $B_1 - G_2$ is the new point identified on the convex hull of the Minkowski difference. **(e)** Following the same principle, the gjk algorithm has converged to the tetrahedra $(D_1 - A_2, E_1 - B_2, H_1 - B_2, D_1 - F_2)$ which encloses the coordinates origin and $V_{k+1} = V_A - V_B$ the closest point of the Minkowsky difference $A - B$ to the origin. **(f)** Overlapping polyhedra and corresponding common plane (CP) and normal $\vec{n}$ calculated using the GJK algorithm, with $V_A$ and $V_B$ the closest points identified between particles $A$ and $B$ ($V_B V_A$ and $\vec{n}$ are parallel).

2) determining the barycentric coordinates of this closest point by solving a small system of equations comprising equation (1) and a revised form of equation (2) which writes:

$$\sum_{i=1}^{p} \lambda_i \vec{OS}_i = \vec{P}_0, \tag{3}$$

where $\vec{P}_0$ contains the coordinates of the projection of O on the affine-hull of the vertices $S_i$ followed by 1. Again, equations (1) and (3) provide a small system of the form $\boldsymbol{M}\lambda=\vec{P}_0$, though this time the $i$-th column of $\boldsymbol{M}$ no more comprises scalar products likely to cause numerical instabilities when their values tend to zero but simply the coordinates of vector $\vec{OS}_i$ followed by 1. When the coordinates origin is inside the simplex, meaning that the particles overlap, the authors state that $\lambda_i$ values may be determined by the following equation:

$$\lambda_i = \boldsymbol{C}_{4,i}/det(\boldsymbol{M}), \tag{4}$$

where *det* denotes the determinant of $\boldsymbol{M}$ and $\boldsymbol{C}_{4,i}$ its $i$-th cofactor. Observe that equation (4) is erroneous since, in accordance with Cramer's rules, $\mathbf{C}_{4,i}$ cannot be the minor of $\boldsymbol{M}$ in absolute value but should comprise the coordinates of vector $\vec{P}_0$ as its $i$-th column to read:

$$\lambda_i = det[\vec{M}_1 ... \vec{M}_{i-1} \vec{P}_0 \vec{M}_{i+1} ... \vec{M}_p]/det(M), \tag{5}$$

where $\vec{M}_j$ denotes the $j$-th column of matrix $\boldsymbol{M}$.

The Lin-Canny algorithm [19] uses another strategy to find the closest features, which may be summarized as follows: define criteria (step 1) to partition the space surrounding each convex polyhedron into regions, each region being closer from a given feature than from any other feature making up the surface of the polyhedron. Then (step 2), select two features, one on each polyhedron, and determine their closest points. Next (step 3), check whether the closest point on one feature belongs to the region of the other feature and conversely. If either check fails, then (step 4) by identifying which criteria failed, select a new set of features on the polyhedra which more likely yields successful checks and iterate from step 3. According to the authors, this algorithm terminates in order $O(v_A + v_B)$ iterations and yields the distance and closest features between two polyhedra. Unfortunately, this algorithm gets stuck in a loop when the two polyhedra are in contact and forcing its termination relies on arbitrary iteration threshold value yet does not allow determining an interpenetration depth if any.

More recently, algorithms based on convex optimization have been designed [18, 27]. Basically, such an algorithm consists in (step 1) writing linear inequalities which define the interior of each polyhedron, then (step 2) establishing whether there exists at least one point verifying simultaneously all the previous inequalities. Next (step 3), in case such a point exists, meaning that both polyhedra intersect, the contact point - where the interparticle contact forces apply - is determined as the *analytical center* of the intersection area using an adapted version of the $log - barrier$ method [28]. This method expresses the burrial depth - with reference to each plane defining the intersection area - of any point located into this area and returns the point maximizing the sum of burrial depth log-function. Last (step 4), the contact normal is calculated as the weighed average of the gradients of two polynomial *inner potentiels*, one for each polyhedron, each potential being defined inside one of the particles as a positive and decreasing function of the distance to the particle surface.

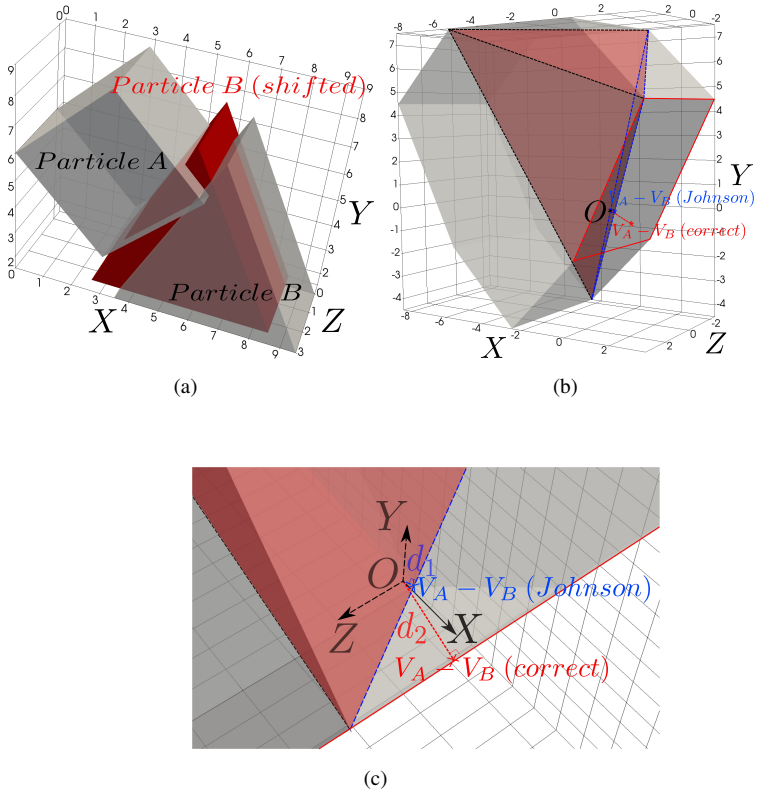
Since GJK appears to be the most versatile contact detection algorithms - unlike others, it may easily handle polyhedron/spheroid contact situations -, the next section sheds a new light on its numerical instabilities and introduces a more robust GJK algorithm which fixes them.

## 3. New light on GJK drawbacks and revised algorithm

### *3.1. New light on GJK drawbacks*

The contact situation depicted in Fig. 5 is of particular interest to shed a new light on the numerical instabilities of GJK algorithm. In this case, the GJK algorithm converges to a simplex whose face located closest to the coordinates origin belongs to the convex hull of the Minkowsky difference of particles $A$ and $B$ (see Fig. 5e). However, if

*Particle B* is slightly translated along the $-X$ and $-Z$ axes with reference to *Particle A* as shown on Fig. 6a, then the GJK algorithm will converge to the simplex depicted in Fig. 6b. Observe that this tetraedral-shaped simplex has two interesting particularities: first, unlike its vertices, none of its triangular faces is located on the convex hull of the Minkowsky difference; second, the coordinates origin is located strictly inside this tetrahedron. As a consequence, applying the Johnson algorithm to determine the closest point of this simplex to the coordinates origin would lead to point $V_A - V_B$ (*Johnson*) (see zoomed Fig. 6c). Unfortunately, this point is burried into the Minkowsky difference of particles $A$ and $B$ and not located on its convex hull, hence the vector linking the coordinates origin to this point cannot be representative of the overlap direction or distance between the two particles. The true closest point $V_A - V_B$ (*correct*) of the Minkowsky difference of particles $A$ and $B$ to the coordinates origin, also depicted in Fig. 6b and c, differs significantly from the previous one. Fig. 7 reflects this difference in terms of common plane orientation and overlap distance between particles $A$ and $B$.




(a) (b)

(c)

Figure 6: Particles shift (a) and new simplex resulting from the GJK algorithm (b). (c) is a magnified view of (b). The shift applied is a simple translation of particle $B$ along the $-X$ and $-Z$ axes. (b) and (c) display the closest point $V_A - V_B$ (*Johnson*) of the tetrahedral-shaped simplex to the coordinates origin as calculated by the Johnson algorithm, the true closest point of the Minkowsky difference of particles $A$ and $B$ to the coordinates origin $V_A - V_B$ (*correct*), as well as the corresponding overlap distances $d_1$ and $d_2$ respectively.

In fact, it is not unusual that the GJK algorithm converges to a simplex such as the one depicted in Fig. 6b, with no face located on the convex hull of the Minkowsky difference of the two particles. Furthermore, the response returned by the GJK algorithm may be even more unsatisfactory. Indeed, upon slightly translating particle $B$ of Fig. 5a along $-X$ as depicted in Fig. 8a, the GJK algorithm will converge to the new tetrahedral-shaped simplex depicted in Fig. 8b. Further to having none of its triangular faces located on the convex hull of the Minkowsky difference of the particles, the particularity of this simplex is that the coordinates origin lays on one of its faces. As a consequence, the closest point of this simplex to the coordinates origin as determined by the Johnson algorithm will be the origin itself, hence the distance between the particles will erroneously be found equal to zero and no overlap will be detected.

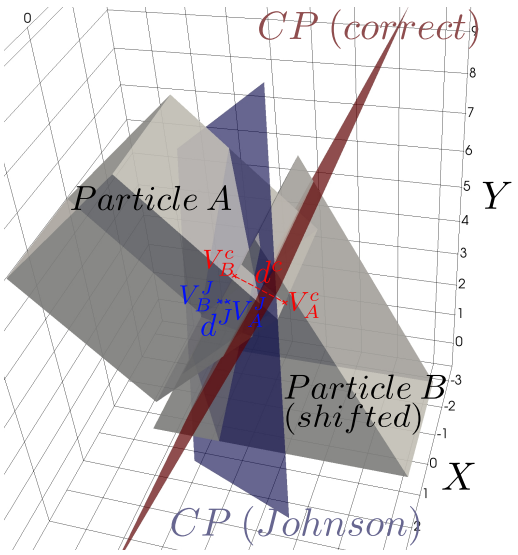
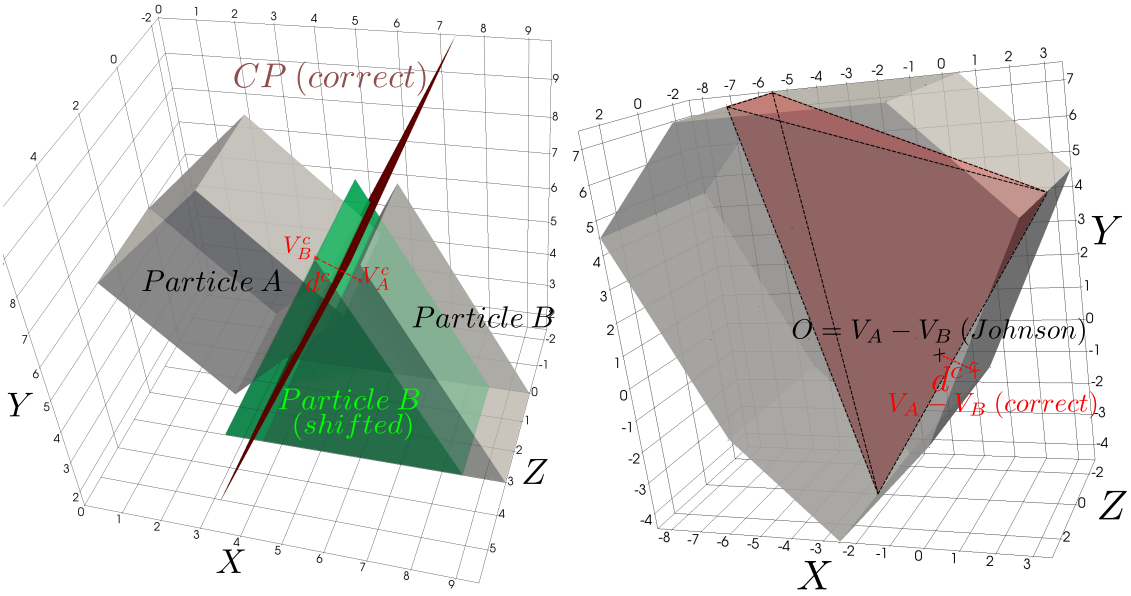



Figure 7: Common plane orientations ($CP$), closest points $V_\alpha^\beta$ and overlap distances $d^\beta$ between particles $A$ and $B$, superscript $J$ refers to Johnson's algorithm and c to true values.




(a) (b)

Figure 8: Shift of *Particle B* with reference to *Particle A*, $V_A$ and $V_B$ being their true closest points (a). New simplex resulting from the GJK algorithm (b). In (b), observe that points $O$ and $V_A - V_B$ ($Johnson$) coincide, so that Johnson's algorithm finds no overlap. The true closest point of the Minkowsky difference of *particle A* and *particle B* (*shifted*) to the coordinates origin is $V_A - V_B$ (*correct*), and the corresponding overlap distance is $d^c$.

### *3.2. A more robust algorithm: GJK-TD*

Observe that, whether polyhedra $A$ and $B$ overlap or not, the point $V$ located closest to the coordinates origin always lies on the outer boundary of their Minkowsky difference $A - B$ (see section 2). Furthermore, for overlapping polyhedra, GJK always converges to a tetrahedral-shaped simplex and the point of this simplex located closest to the coordinates origin always lies on one of its triangular faces (e.g. see points $V_A - V_B$ ($Johnson$) in Fig. 5e, 6c and 8b). A straightforward consequence is that, in order to circumvent the drawbacks of the GJK algorithm evidenced in sec-

tion 3.1, additional steps are needed to ensure that, at the end of the GJK algorithm, the simplex triangular face located closest to the coordinates origin lies on the outer boundary of the Minkowsky difference $A-B$ of the polyhedra. These steps are depicted in the next paragraph using the notations of section 3.1.

Consider the contact situation depicted in Fig. 8a between *Particle A* and *Particle B* (*shifted*), for which applying the GJK algorithm yields the tetrahedral-shaped simplex of Fig. 8b. The basic idea of the steps taken to relocate the closest triangular face to the coordinates origin on the outer boundary of the Minkowsky difference $A-B$ of the polyhedra may be summarized as follows : 1) determine the distance of each triangular face of the tetrahedral-shaped simplex to the coordinates origin, then 2) expand the face located closest to the coordinates origin along its normal towards the outer boundary of the Minkowsky difference $A-B$ of the polyhedra, and 3) repeat this expansion with the subsequent closest triangular face among the resulting set of triangular faces until the condition that the current closest triangular face is located on the outer boundary of the Minkowsky difference $A-B$ of the polyhedra is fulfilled. Both the expansion principle and face location condition may simply be explained using the contact situation depicted in Fig. 8.

To describe the expansion principle, consider the tetrahedral-shaped simplex $(A_1, B_1, C_1, D_1)$ of Fig. 9a, which is the same simplex as in Fig. 8b but observed from another perspective. Once the triangular face $(A_1, B_1, C_1)$ minimizing the distance $d_1$ to the coordinates origin has been identified, its normal $\vec{n}_1$ pointing outside the initial simplex (or, equivalently, in the direction opposite to the coordinates origin) is calculated. Next, vertex $E_1$ is determined as the Minkowsky difference between the outermost vertex of *particle A* in the oriented direction $\vec{n}_1$ and the outermost vertex of *particle B* in the opposite direction (see Fig. 9b). Last, triangular faces $(A_1, B_1, E_1)$, $(B_1, C_1, E_1)$ and $(A_1, C_1, E_1)$ are substituted for triangular face $(A_1, B_1, C_1)$ in the set of faces in which the closest face to the coordinates origin is sought. Fig. 9c to e depict the application of the expansion principle to the subsequent face located closest to the coordinates origin and so on. It should be emphasized that, until the closest triangular face to the coordinates origin is found on the Minkowsky outer boundary of the polyhedra (see triangular face $(B_1, E_4, E_3)$ in Fig. 9e), each triangular face is deleted right after being visited (e.g. face $(A_1, B_1, C_1)$). Furthermore, records are kept to ensure that, during the subsequent iterations of the expansion process, deleted faces are never rebuilt, thus ensuring that the algorithm will not get trapped in a non-ending loop.

The face location condition used to ensure that the current triangular face is located on the outer boundary of the Minkowsky difference $A-B$ of the polyhedra simply consists in 1) determining the outermost vertex of *particle A* in the oriented direction $\vec{n}$ normal to the current face (see Fig. 9e) and the outermost vertex of *particle B* in the opposite direction, and 2) checking that the resulting point $E_4$, which is a vertex of the outer boundary of the Minkowsky difference $A-B$ of the polyhedra, belongs to the current triangular face.

Once the triangular face located closest to the coordinates origin $(B_1, E_4, E_3)$ has been determined on the Minkowsky outer boundary of the polyhedra (see Fig. 9e), the closest point $V$ of this face to the coordinates origin $O$ is calculated as the barycentre of vertices $(B_1, E_4, E_3)$ for which vector $\vec{OV}$ is normal to the face:

$$\sum_{i=1}^{3} \lambda_i = 1 \quad \text{and} \quad \forall\, i \in \{1, ..., 3\} \quad \lambda_i \geq 0, \tag{6}$$

$$\sum_{i=1}^{3} \lambda_i \vec{OS}_i = 0\vec{V}, \tag{7}$$

with $S_i$ in $\{B_1, E_4, E_3\}$ and $0\vec{V} = (\vec{OS}_1.\vec{n})\vec{n}$.

Equations (6) and (7) provide a small system of the form $\boldsymbol{M}\lambda = \vec{OV}$, whose unknowns $\lambda_i$ are easily determined by the following equations:

$$\lambda_1 = det[\vec{OV}\vec{OS}_2\vec{OS}_3]/det[\vec{OS}_1\vec{OS}_2\vec{OS}_3], \tag{8}$$

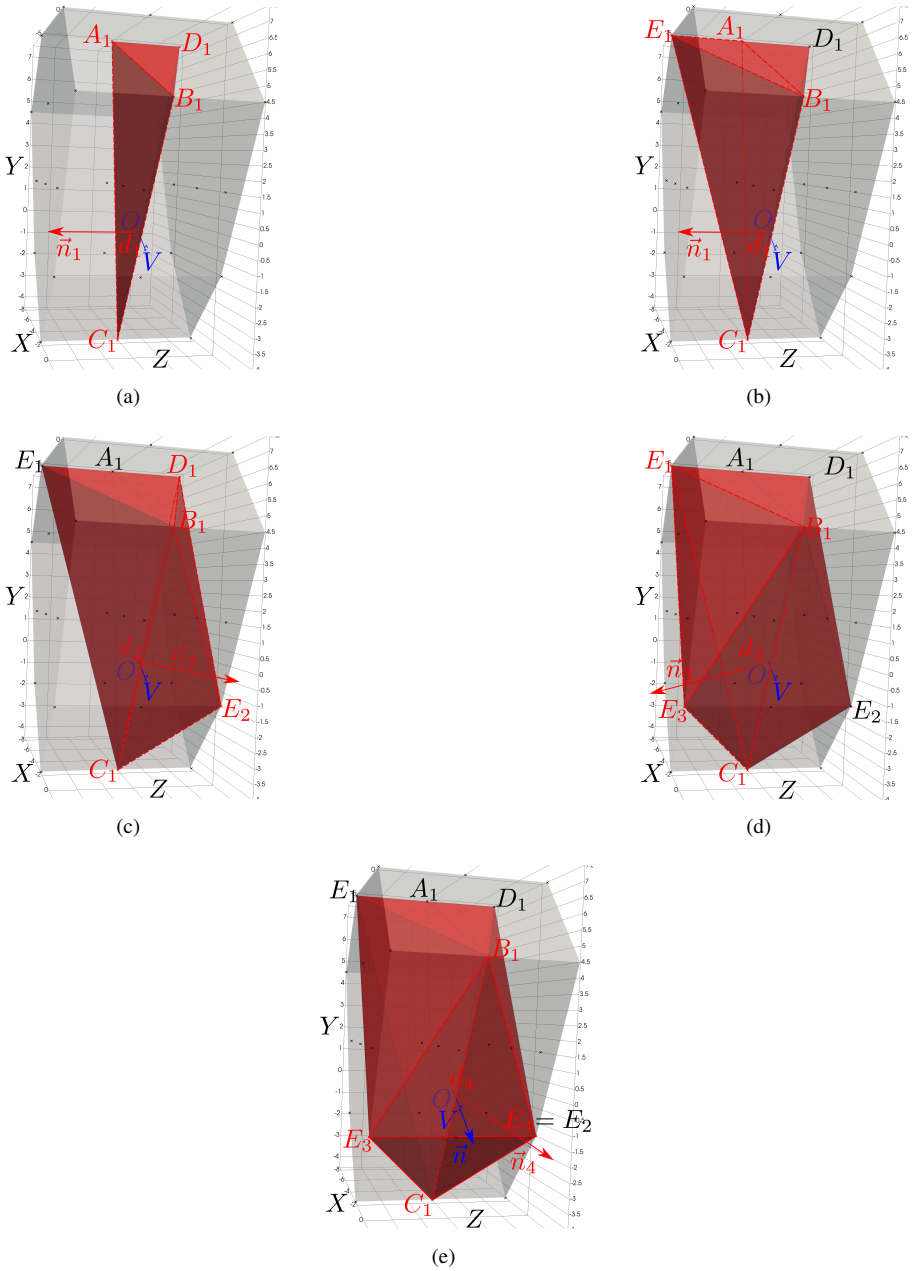



Figure 9: Tetrahedron expansion principle. Starting from tetrahedral-shaped simplex $(A_1, B_1, C_1, D_1)$ (a), the closest triangular face $(A_1, B_1, C_1)$ to the coordinates origin is identified and its outward normal $\vec{n}_1$ (oriented opposite to the coordinates origin) is calculated. Then, the outermost vertex $E_1$ along $\vec{n}_1$ is identified (b), triangular face $(A_1, B_1, C_1)$ is deleted and replaced by triangular faces $(A_1, B_1, E_1)$, $(B_1, C_1, E_1)$, $(A_1, C_1, E_1)$ so that the former tetrahedral-shaped simplex becomes the new polyhedron $(A_1, D_1, B_1, C_1, E_1)$. Similarly, the closest triangular face $(D_1, B_1, C_1)$ to the coordinates origin of the new polyhedron is identified (c) and its outward normal $\vec{n}_2$ is calculated, in order to determine vertex $E_2$ and substitute triangular faces $(B_1, C_1, E_2)$, $(C_1, D_1, E_2)$ and $(D_1, B_1, E_2)$ for triangular face $(B_1, C_1, D_1)$ to achieve a new polyhedron. Following the same principle, the closest triangular face $(E_1, B_1, C_1)$ to the coordinates origin of the new polyhedron is identified (d) and its outward normal $\vec{n}_3$ is calculated, in order to determine vertex $E_3$ and substitute triangular faces $(B_1, C_1, E_3)$, $(C_1, E_1, E_3)$ and $(E_1, B_1, E_3)$ for triangular face $(B_1, C_1, E_1)$. Similarly, the last step (e) identifies $(B_1, C_1, E_3)$ as the closest triangular face to the coordinates origin, with $\vec{n}_4$ its outward normal allowing to identify outermost vertex $E_4 = E_2$. This new vertex yields triangular face $(B_1, E_4, E_3)$ which is both located closest to the coordinates origin and on the convex hull of the Minkowsky difference (represented in light-grey), hence the expansion is stopped.

$$\lambda_2 = det[\vec{OS}_1\vec{OV}\vec{OS}_3]/det[\vec{OS}_1\vec{OS}_2\vec{OS}_3], \tag{9}$$

$$\lambda_3 = det[\vec{OS}_1\vec{OS}_2\vec{OV}]/det[\vec{OS}_1\vec{OS}_2\vec{OS}_3]. \tag{10}$$

Last, these barycentric coordinates $\lambda_i$ are used to determine the closest points $V_A^c$ and $V_B^c$ of polyhedra $A$ and $B$ as the barycentres of the particle vertices forming the Minkowski vertices $\{B_1, E_4, E_3\}$ (see Fig. 8a). As already mentioned, $-\|\vec{OV}\|$ (negative distance) defines the overlap depth of the overlapping particles and $\vec{OV}$ their contact normal (pointing from particle $A$ to particle $B$).

The new algorithm that has just been presented is summarized as Algorithm 3. It was called $GJK-TD$ to emphasize the face expansion steps ($TD$ stand for $Tetrahedron\ Distortion$) performed at the end of a basic application of the GJK algorithm. Several comments shall be made about this new algorithm when applied to two overlapping convex polyhedra made of triangular faces.
First, like the Signed Volumes method [23], the matrix $M$ of the small system defined by equations (6) and (7) involves no scalar product likely to cause ill-conditioning of the system. Hence, $\lambda_i$ values defined by equations (8), (9), (10) may always be calculated.
Second, the expansion algorithm always converges to a triangular face located on the outer boundary of the Minkowsky difference between the polyhedra, and the convergence occurs in less than $3(v_A + v_B) - 8$ steps. Indeed, the algorithm

**Algorithm 3** GJK-TD algorithm

**Require:** GJK applied to convex polyhedra A and B has converged to a tetrahedral-shaped simplex
$\sum = \emptyset$
**for** face $p$ in simplex **do**
  Calculate $d_p$, $\vec{n}_p$
  $\sum \leftarrow \vec{n}_p$, $d_p$, face connectivity
**end for**
$q = 0$
$\sum_{visited} = \emptyset$
**loop**
  Search $\sum$ for triangular face $(S_i, S_j, S_k)$ with smallest $d_p$
  **if** Triangular face location condition = TRUE **then**
    Calculate $\vec{OV} = (\vec{OS}_i.\vec{n}_p)\vec{n}_p$
    Calculate $\lambda_i$, $\lambda_j$ and $\lambda_k$
    Calculate $V_A^c$ and $V_B^c$ using $\lambda_x$
    Exit loop
  **else**
    $q \leftarrow q + 1$
    Calculate outermost vertex of polyhedron A along $\vec{n}_p$
    Calculate outermost vertex of polyhedron B along $-\vec{n}_p$
    Calculate corresponding Minkowsky vertex $E_q$
    **for** each new triangle made of $E_q$ and two vertices $(S_{r1}, S_{r2})$ among $(S_i, S_j, S_k)$ **do**
      **if** Triangular face $(S_{r1}, S_{r2}, E_q)$ not in $(\sum_{visited}$ U $\sum)$ **then**
        Calculate normal $\vec{n}_{q+4}$ and distance $d_{q+4}$
        $\sum \leftarrow \vec{n}_{q+4}$, $d_{q+4}$ and face connectivity
        $q \leftarrow q + 1$
      **end if**
    **end for**
    $\sum_{visited} \leftarrow \vec{n}_p$, $d_p$ and face p connectivity
    Discard $\vec{n}_p$, $d_p$ and face p connectivity from $\sum$
  **end if**
**end loop**

consists in finding a sequence of triangular faces whose distance to the coordinates origin is strictly increasing. Furthermore, the vertices of these triangular faces belong to the Minkowsky difference of the polyhedra. Hence, the distances to the coordinates origin of the sequence of triangular faces are bounded by that of the closest triangular face of the outer boundary of the Minkowsky difference. As a consequence, like for any strictly increasing and bounded mathematical sequence in a convex set, the convergence is always achieved. Next, each convex polyhedron consisting of triangular faces, the outer boundary of their Minkowsky difference is also made of triangular faces. Besides, the triangular face to which the expansion algorithm converges is always located on the outer boundary of the Minkowsky difference of the polyhedra as a consequence of the face location condition. Finally, the complexity of the algorithm may be assessed as follows:

- According to the Descartes-Euler theorem, the total number of faces $f$, edges $e$ and vertices $v$ of any convex polyhedron relate by the formula $f - e + v = 2$;
- Now, the outer boundary of the Minkowsky difference of the polyhedra is made exclusively of triangular faces, each comprised of three edges shared by exactly two faces, hence the total number of edges and faces relate according to $e = \frac{3f}{2}$;
- The two preceeding relations yield $f = 2v - 4$;
- Besides, the expansion algorithm may visit *internal* triangular faces, that is faces embedded into the Minkowsky difference of the polyhedra and not located on its outer boundary (the latter being *external* faces), whose total number is $f - 4$ (this may easily be shown by recurrence);
- Last, the outer boundary of the Minkowsky difference between two polyhedra made of $v_A$ and $v_B$ vertices respectively contains exactly $v_A + v_B$ vertices. As a consequence, the expansion algorithm will visit at most all $2(v_A + v_B) - 4$ external and $v_A + v_B - 4$ internal triangular faces, and these faces will be visited at most once since any visited face is deleted and never restored until convergence.

## 4. Efficiency of the GJK-TD algorithm

In order to examine the efficiency of the GJK-TD algorithm, one shall address both its calculation accuracy and speed. This is achieved upon considering nine contact situations with various overlap, which are used as a benchmark to compare GJK-TD with the revised versions of Cundall's and Nezami's algorithms as depicted in section 2. Given that these three algorithms implement significantly different strategies to determine a contact situation, it seems reasonable to consider that the level of agreement between the results returned by these methods reflects their accuracy. Furthermore, upon selecting more or less overlapped contact situations, which are representative of all common contact types, namely vertex-face, edge-face, edge-edge, or face-face contacts, comparing the calculation speed between these three algorithms should yield representative orders of magnitude. The nine contact situations used are depicted in Fig. 2, 4, 5f, 7, 8a, 10a, 10b, 10c, 10d respectively.

All calculations are performed on a double precision Ubuntu 16.04 – *LTS* machine equipped with an intel® Core™i7 – 6700HQ CPU@2.60GHz processor having 8 GB RAM. The calculation routines, GJK-TD as well as the revised versions of Cundall's and Nezami's algorithms, are written in Fortran90 and compiled using GNU Fortran 5.4.0.

### *4.1. Computing accuracy*

For each of the nine contact situations, both the normal components and overlap depth are calculated to the highest possible precision level. All calculation routines use double precision values, hence results are returned with 15 significant digits (machine precision). For each contact situation, Table 1 summarizes the results returned by the GJK-TD algorithm and Fig. 11 displays the accuracy with which the revised versions of Cundall's and Nezami's algorithms agree with the results returned by GJK-TD. Note that two separate charts are displayed, Fig. 11a depicting accuracy in terms of mean absolute difference between the normal components calculated by any of the revised algorithm and GJK-TD, and Fig. 11b depicting accuracy in terms of overlap depths absolute difference with reference to GJK-TD.

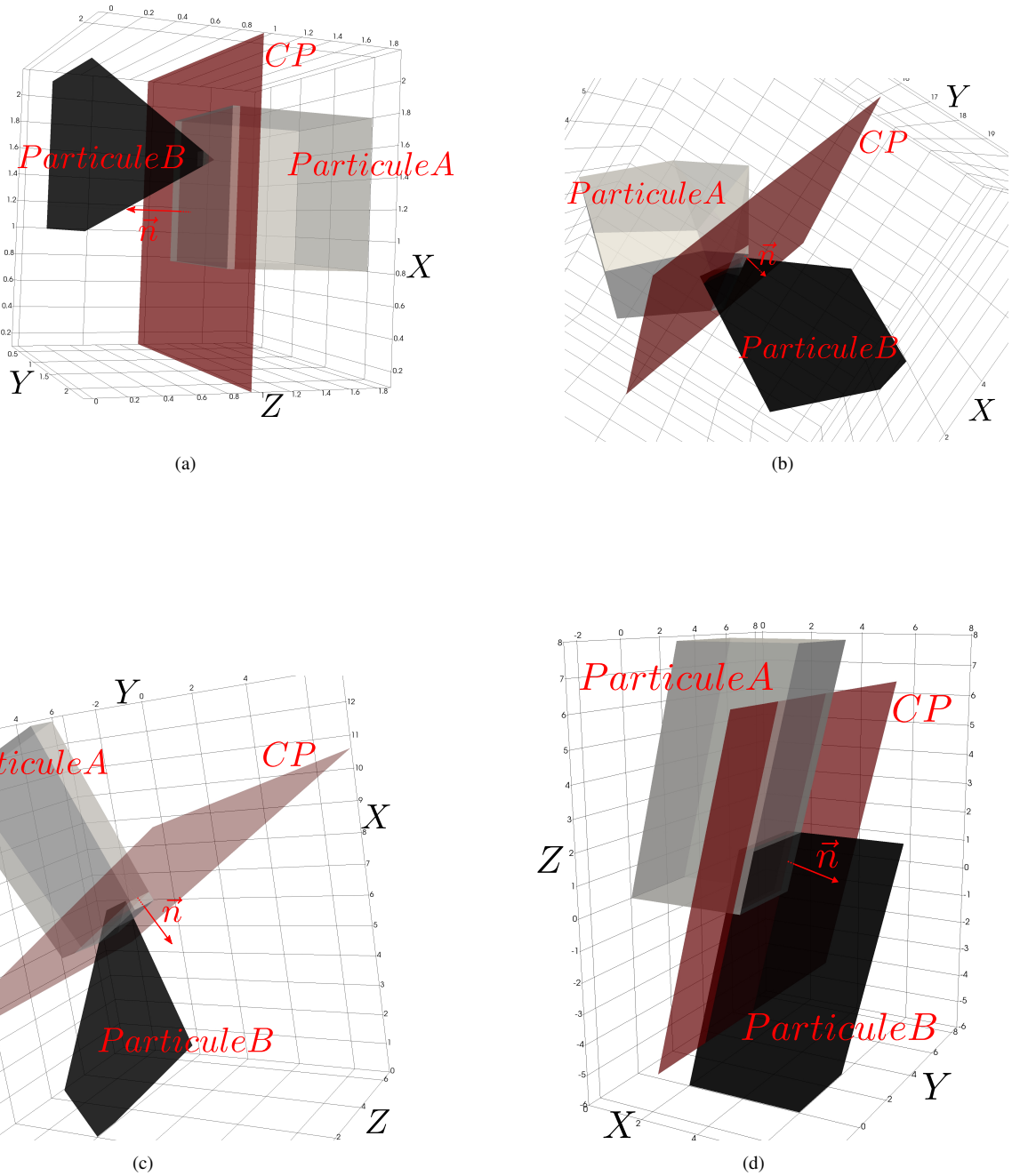



Figure 10: Various 3D contact situations used to benchmark GJK-TD with the revised versions of Cundall's and Nezami's algorithms, (a) vertex-edge contact, (b) edge-edge contact, (c) edge-face contact and (d) face-face contact. $\vec{n}$ is the normal vector of the common plane (*CP*), pointing from *Particle A* to *Particle B*.

Table 1: Normal components and overlap calculated to machine precision ($10^{-15}$) by GJK-TD for nine contact situations.

| Test-case | $n_x$ | $n_y$ | $n_z$ | overlap |
|---|---|---|---|---|
| Fig. 2 | 0.000000000000000 | 1.000000000000000 | 0.000000000000000 | −0.250000000000000 |
| Fig. 4 | 0.992277876713668 | −0.124034734589208 | 0.000000000000000 | −4.026012719933860 |
| Fig. 5f | 0.894427190999916 | −0.447213595499958 | 0.000000000000000 | −0.223606797749979 |
| Fig. 7 | 0.894427190999916 | −0.447213595499958 | 0.000000000000000 | −1.118033988749900 |
| Fig. 8a | 0.894427190999916 | −0.447213595499958 | 0.000000000000000 | −1.118033988749900 |
| Fig. 10a | 0.000000000000000 | 0.000000000000000 | −1.000000000000000 | −0.100000024000000 |
| Fig. 10b | −0.168309951548722 | 0.950616325200044 | −0.260768791217108 | −0.572082006256242 |
| Fig. 10c | 0.832050294337844 | −0.554700196225229 | 0.000000000000000 | −0.554700196225229 |
| Fig. 10d | 0.970142500145332 | 0.000000000000000 | −0.242535625036333 | −0.485071250072666 |

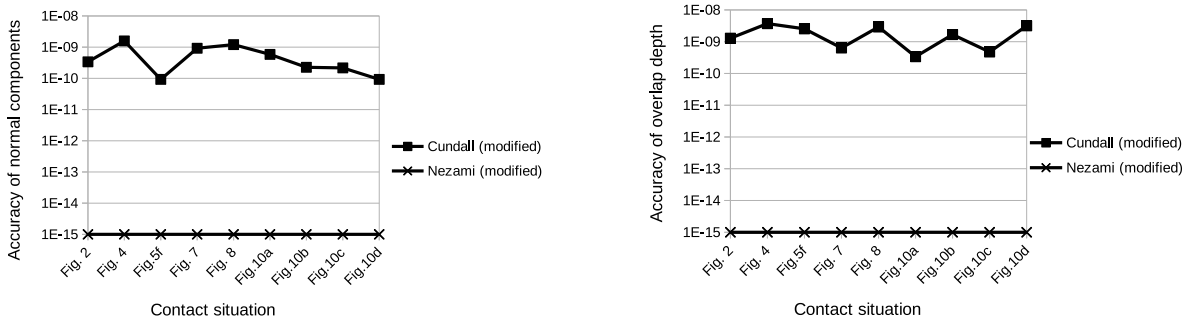



Figure 11: Comparison of calculation accuracy against GJK-TD for algorithms Cundall (revised) and Nezami (revised), for nine contact situations, (a) normal mean components, (b) overlap, (c) edge-face contact and (d) face-face contact.

Observe that, whatever the contact situation, GJK-TD and the revised version of Nezami's algorithm return results which agree with eachother to machine precision. This is consistent with the finite and limited number of possible common plane candidates when two convex polyhedra overlap. By contrast, the revised version of Cundall's algorithm returns results which agree with those of the other two algorithms at best to $10^{-10}$. Note that a better agreement could be achieved upon setting the Cundall's minimum orientational perturbation step to a value below the requested $10^{-7}\pi/180$ radians, but this would strongly affect the calculation speed.

Next, the level of accuracy of these results is compared with that of the results reported by Montanari et al. [23] on gear teeth with various number of vertices (see their figure 15). Montanari's results appear to be at least one order of magnitude less accurate than the results reported in the present paper.

### *4.2. Computing speed*

For each of the nine contact situations depicted in Fig. 2, 4, 5f, 7, 8a, 10a, 10b, 10c, 10d, Table 2 summarizes the calculation speed of all three contact detection algorithms both in terms of absolute and relative to GJK-TD values. These calculation speed figures are obtained using the *SYSTEM_CLOCK* Fortran function, which allows a level of precision up to 1 nanosecond, and the reported values are averaged over one million repetitions of each algorithm. According to this table, GJK-TD is able to return the contact normal and overlap depth within 6 to 95 microseconds, depending on the contact situation, whereas the revised versions of Cundall's and Nezami's algorithms are respectively 6 and 65 times more computationally intense on average, and up to 12 and 135 times slower respectively. Furthermore, it should be pointed out that, due to the extra loop encompassing lines 4 to 28 of Algorithm 1, the revised Cundall

Table 2: Comparison of calculation speed between algorithms Cundall (revised), Nezami (revised) and GJK-TD, for nine contact situations.

| Test case | GJK-TD (x$10^{-6}$ s) | Nezami (revised) (x$10^{-6}$ s) | Nezami (revised)/ GJK-TD | Cundall (revised) (x$10^{-6}$ s) | Cundall (revised)/ GJK-TD |
|---|---|---|---|---|---|
| Fig. 2 | 10.896 | 83.920 | 7.7 | 540.656 | 49.6 |
| Fig. 4 | 94.688 | 121.456 | 1.3 | 1225.261 | 12.9 |
| Fig. 5f | 6.425 | 77.768 | 12.1 | 870.992 | 135.6 |
| Fig. 7 | 19.095 | 91.640 | 4.8 | 938.476 | 49.1 |
| Fig. 8a | 18.975 | 77.156 | 4.1 | 671.476 | 35.4 |
| Fig. 10a | 10.697 | 67.356 | 6.3 | 1288.066 | 120.4 |
| Fig. 10b | 27.189 | 85.672 | 3.2 | 2582.582 | 95.0 |
| Fig. 10c | 22.239 | 161.700 | 7.3 | 1056.096 | 47.5 |
| Fig. 10d | 19.208 | 85.068 | 4.4 | 719.848 | 37.5 |
| mean | 25.490 | 94.637 | 5.7 | 1099.273 | 64.8 |

algorithm has a complexity which is three times that of the classical Cundall algorithm. In other words, the computational cost of the former is three times that of the latter. Yet, as mentioned in section 2, in some configurations, the classical Cundall algorithm fails to return correct results for the contact plane.

A first attempt was made to confront these figures with those reported by Montanari et al. [23] on overlapping polygonal spheres with various number of vertices tested over one million cycles (see their figure 17c). Their figure 17c suggests that the speed of the tested algorithms does not significantly vary with the number of vertices, hence confronting their results in terms of algorithm velocity to those of the present paper obtained with different polyhedral-shaped particles makes sense. However, their figure raises questions about the CPU time unit (ns): indeed, repeating a simple assignment to 1 of a unique integer variable for one million cycles already takes 6.153152 $10^{-3}$ s with the machine used in the present paper, hence about 6 ns per assignement, and about 2.5 ns per assignment on an Intel® Xeon® CPU E5 – 2630 v4 2.20GHz processor having 28 GB RAM. As a consequence, it seems doubtfull that the run of a full algorithm such as $JB$, $BK$ or $SV$ as reported by Montanari et al. only takes a few nanoseconds. Then we confronted our contact detection speed figures to those reported by Wachs et al. [15] on overlapping cubes or tetrahedra. These authors have implemented a tailored version of the GJK algorithm in which overlapping particles are first reduced through a homothety, so that they are no more in contact, prior to determining the set of closest points. The contact detection speed reported by these authors on an Intel4® Pentium CPU 3GHz processor is 74 $\mu s$ and 106 $\mu s$ for overlapping cubes and tetrahedra respectively. These figures are consistent with those of the present paper.

## 5. Conclusion

Algorithms dedicated to initial contact detection between overlapping convex polyhedra have been reviewed, focusing on three of the most prominent ones [16, 17, 20]. Drawbacks affecting the use of these three algorithms into DEM simulations have been evidenced and solutions to these drawbacks have been suggested. In particular, new light was shed on GJK drawbacks and a new algorithm supplementing the original one was introduced. Finally, revised versions of these three algorithms implementing the suggested solutions have been benchmarked over nine contact situations. These contact situations involve two overlapping polyhedra which are representative of all common contact types, namely vertex-face, edge-face, edge-edge and face-face contacts. The benchmarking results show that GJK-TD and Nezami (revised) return identical results to machine precision, whereas Cundall (revised) returns identical results to the tenth decimal place. Furthermore, these results show the GJK-TD returns both the normal components and overlap depth within a few tens of microseconds, whereas Nezami (revised) and Cundall (revised) are respectively 6 and 65 times more computationally intense on average. It is believed that the robustness and efficiency of GJK-TD will boost its use into DEM simulations, all the more that this versatile algorithm may easily be customized to detect

contact between convex polyhedra and spheroid particles.

## 6. References

## References


[1] L. E. Silbert, D. Ertas, G. S. Grest, T. Halsey, D. Levine, S. J. Plimpton, Granular flow down an inclined plane, Phys. Rev. E 64 (2001) 385–403.

[2] E. Azéma, Y. Descantes, N. Roquet, J. N. Roux, F. Chevoir, Discrete simulation of dense flows of polyhedral grains down a rough inclined plane, Phys. Rev. E 86 (2012) 031303.

[3] L. E. Silbert, D. Ertas, G. S. Grest, T. C. Halsey, D. Levine, Geometry in Frictionless and Friction Sphere Packings, Phys. Rev. E 65 (2002) 031304.

[4] R. S. Farr, R. D. Groot, Close packing density of polydisperse hard spheres, J. Chem. Phys. 131 (2010) 244104.

[5] P. Cundall, O. Strack, A discrete numerical model for granular assemblies, Geotechnique 29 (1) (1979) 47–65.

[6] H. A. Makse, D. L. Johnson, L. M. Schwartz, Packing of compressible granular materials, Physical Review Letters 84 (2000) 4160–4163.

[7] S. Torquato, T. M. Truskett, P. G. Debenedetti, Is Random Close Packing of Spheres Well Defined ?, Physical Review Letters 84 (2000) 2064–2067.

[8] A. Donev, R. Connelly, F. H. Stillinger, S. Torquato, Underconstrained Jammed Packings of Nonspherical Hard Particles : Ellipses and Ellipsoids, Phys. Rev. E 75 (2007) 051304.

[9] S. R. Williams, A. P. Philipse, Random packings of spheres and spherocylinders simulated by mechanical contraction, Phys. Rev. E 67 (2003) 051301.

[10] J. Zhao, S. Li, W. Jin, X. Zhou, Shape Effects on the Random-Packing Density of Tetrahedral Particles, Phys. Rev. E 86 (2012) 031307.

[11] S. Torquato, Y. Jiao, Dense Packings of Polyhedra: Platonic and Archimedean Solids, Phys. Rev. E 80 (2009) 041104.

[12] E. Azéma, F. Radjai, G. Saussine, Quasistatic rheology, force transmission and fabric properties of a packing of irregular polyhedral particles, Mechanics of Materials 41 (6) (2009) 729 – 741, advances in the Dynamics of Granular Materials.

[13] J.-F. Camenen, Y. Descantes, P. Richard, Effect of confinement on dense packings of rigid frictionless spheres and polyhedra, Phys. Rev. E 86 (2012) 061317.

[14] J. F. Camenen, Y. Descantes, Geometrical properties of rigid frictionless granular packings as a function of particle size and shape, Phys. Rev. E 96 (2017) 012904.

[15] A. Wachs, L. Girolami, G. Vinay, G. Ferrer, Grains3d, a flexible dem approach for particles of arbitrary convex shape part i: Numerical model and validations, Powder Technology 224 (2012) 374 – 389.

[16] P. Cundall, Formulation of a three-dimensional distinct element modelpart i. a scheme to detect and represent contacts in a system composed of many polyhedral blocks, International Journal of Rock Mechanics and Mining Sciences and Geomechanics Abstracts 25 (3) (1988) 107 – 116.

[17] E. G. Nezami, Y. M. Hashash, D. Zhao, J. Ghaboussi, A fast contact detection algorithm for 3-d discrete element method, Computers and Geotechnics 31 (7) (2004) 575 – 587.

[18] C. Boon, G. Houlsby, S. Utili, A new algorithm for contact detection between convex polygonal and polyhedral particles in the discrete element method, Computers and Geotechnics 44 (Complete) (2012) 73–82.

[19] M. C. Lin, J. F. Canny, A fast algorithm for incremental distance calculation, in: Proceedings of the 1991 IEEE International Conference on Robotics and Automation, 1991, pp. 1008–1014.

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

[21] B. Mirtich, V-clip: Fast and robust polyhedral collision detection, ACM Trans. Graph. 17 (3) (1998) 177–208.

[22] D. W. Johnson, The optimization of robot motion in the presence of obstacles, Ph.D. thesis, Univ. of Michigan (1987).

[23] M. Montanari, N. Petrinic, E. Barbieri, Improving the gjk algorithm for faster and more reliable distance queries between convex objects, ACM Trans. Graph. 36 (3) (2017) 30:1–17.

[24] G. Van den Bergen, A fast and robust gjk implementation for collision detection of convex objects, Journal of Graphics Tools 4 (2) (1999) 7–25.

[25] C. Ericson, Real-Time Collision Detection, Morgan Kaufmann, San Francisco, 2004.

[26] V. Tereshchenko, S. Chevokin, A. Fisunenko, Algorithm for finding the domain intersection of a set of polytopes, Procedia Computer Science 18 (2013) 459 – 464, 2013 International Conference on Computational Science.

[27] C. Boon, G. Houlsby, S. Utili, A new contact detection algorithm for three-dimensional non-spherical particles, Powder Technology 248 (2013) 94 – 102, discrete Element Modelling.

[28] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.