Contributing Vertices-Based Minkowski Sum of a Nonconvex–Convex Pair of Polyhedra

HICHEM BARKI, FLORENCE DENIS, and FLORENT DUPONT
Université de Lyon, CNRS
Université Lyon 1, LIRIS UMR5205

The exact Minkowski sum of polyhedra is of particular interest in many applications, ranging from image analysis and processing to computer-aided design and robotics. Its computation and implementation is a difficult and complicated task when nonconvex polyhedra are involved. We present the NCC-CVMS algorithm, an exact and efficient contributing vertices-based Minkowski sum algorithm for the computation of the Minkowski sum of a nonconvex–convex pair of polyhedra, which handles nonmanifold situations and extracts eventual polyhedral holes inside the Minkowski sum outer boundary. Our algorithm does not output boundaries that degenerate into a polyline or a single point. First, we generate a superset of the Minkowski sum facets through the use of the contributing vertices concept and by summing only the features (facets, edges, and vertices) of the input polyhedra which have coincident orientations. Secondly, we compute the 2D arrangements induced by the superset triangles intersections. Finally, we obtain the Minkowski sum through the use of two simple properties of the input polyhedra and the Minkowski sum polyhedron itself, that is, the closeness and the two-manifoldness properties. The NCC-CVMS algorithm is efficient because of the simplifications induced by the use of the contributing vertices concept, the use of 2D arrangements instead of 3D arrangements which are difficult to maintain, and the use of simple properties to recover the Minkowski sum mesh. We implemented our NCC-CVMS algorithm on the base of CGAL and used exact number types. More examples and results of the NCC-CVMS algorithm can be found at: http://liris.cnrs.fr/hichem.barki/mksum/NCC-CVMS

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*; J.6 [**Computer Applications**]: Computer-Aided Engineering—*Computer-aided design (CAD)*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: 2D arrangement computation, 3D intersection, computer-aided design, contributing vertices, Minkowski sum

ACM Reference Format:

Barki, H., Denis, F., and Dupont, F. 2011. Contributing vertices-based Minkowski sum of a nonconvex–convex pair of polyhedra. ACM Trans. Graph. 30, 1, Article 3 (January 2011), 16 pages. DOI = 10.1145/1899404.1899407 http://doi.acm.org/10.1145/1899404.1899407

1. INTRODUCTION

The Minkowski sum or addition of two sets $\mathcal{A}$ and $\mathcal{B}$ in a vector space was defined by the German mathematician Herman Minkowski (1864–1909) as a position vector addition of all elements a and b coming from $\mathcal{A}$ and $\mathcal{B}$, respectively: $\mathcal{A} \oplus \mathcal{B} = \{a + b | a \in \mathcal{A}, b \in \mathcal{B}\}$.

Another definition states that the Minkowski sum of two sets $\mathcal{A}$ and $\mathcal{B}$ is obtained by sweeping all points of $\mathcal{A}$ by $\mathcal{B}$, that is, translating $\mathcal{B}$ so that its origin (the common initial point of all its position vectors) passes through all points of $\mathcal{A}$, and taking the union of all resulting points: $\mathcal{A} \oplus \mathcal{B} = \bigcup_{a \in \mathcal{A}} \mathcal{B}_a$, where $\cup$ denotes the set union operation and $\mathcal{B}_a$ denotes the set $\mathcal{B}$ translated by a position vector a.

Our aim is to compute the Minkowski sum polyhedron $S = A \oplus B$. The polyhedra A and B are the respective boundary representations of the sets $\mathcal{A}$ and $\mathcal{B}$ in $\mathbb{R}^3$. Thus, the aforesaid Minkowski sum definition based on the sweep operation becomes

$$S = A \oplus B = \bigcup_{a \in A} B_a. \tag{1}$$

This is a direct consequence of the boundary representation (polyhedra) we are dealing with, which means that in order to compute the Minkowski sum polyhedron S, it is sufficient to sweep only the boundary points of A by B and to take the boundary of the union of the resulting points (see Figure 1).

This work is partially supported by the French Cluster ISLE of the Rhone-Alpes region within the LIMA Project and also by the ANR (Agence Nationale de la Recherche, France) through MADRAS project (ANR-07-MDCO-015).

Authors' addresses: H. Barki, F. Denis, and F. Dupont, Université de Lyon, CNRS – Université Lyon 1, LIRIS, UMR5205 – 43 Bd. du novembre 1918, F-69622 Villeurbanne, France; email: {hichem.barki, florence.denis, florent.dupont}@liris.cnrs.fr.

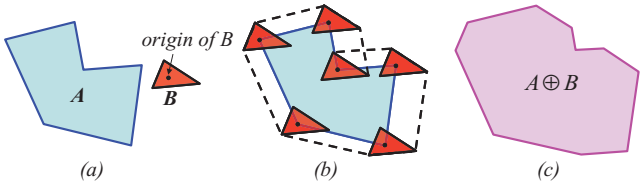

Fig. 1. (a) Two polygons A and B; (b) sweeping all boundary points of A by B; (c) the Minkowski sum polygon $A \oplus B$.

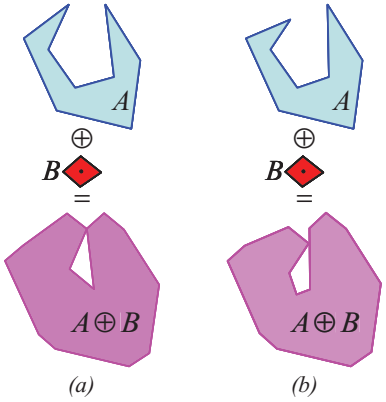

Fig. 2. Nonmanifold Minkowski sums of polygons. (a) Double vertices; (b) a vertex touching an edge tangentially.

By definition, polyhedra are closed and two-manifold meshes. The polyhedra we are working with are handled by the halfedge data structure [Kettner 1999]. Each edge of a polyhedron is represented by two halfedges with opposite orientations. The closeness property implies that a polyhedron does not contain any halfedge which has no incident polyhedral facet (border halfedge). The two-manifoldness property implies that the neighborhood of each point belonging to the polyhedron's surface is homeomorphic to a disk. From these two definitions of closeness and two-manifoldness properties we conclude that each edge (or pair of opposite halfedges) of a polyhedron is incident to exactly two polyhedral facets.

The Minkowski sum polyhedron $S = A \oplus B$ of two polyhedra A and B is a closed polyhedron since it corresponds to the union of translated versions of B, which is a closed and two-manifold polyhedron. These translations are defined by a closed and two-manifold surface which is the boundary of the polyhedron A. However, the Minkowski sum S is not necessarily two-manifold if at least one of the two polyhedra is not convex. Examples of nonmanifold situations are edges which are incident to more than two facets, double vertices, or facets that touch tangentially. Figure 2 illustrates some of these nonmanifold situations in 2D.

1.1 Importance of the Minkowski Sum of Polyhedra

The Minkowski sum of polyhedra is of great interest for many applications, such as computer-aided design and manufacturing [Lee et al. 1998], computer animation and morphing [Kaul and Rossignac 1992], morphological image analysis [Serra 1982, 1988], robot motion planning [Lozano-Pérez 1983], similarity measures for convex polyhedra [Tuzikov et al. 2000], penetration depth estimation and dynamic simulation [Kim et al. 2003], and solid modeling.

For many years, the lack of exact, robust, and efficient algorithms for the computation of the Minkowski sum of polyhedra hindered the development of such domains. As an example, mathematical morphology tools used in the image analysis field can be considered. These tools are based on two dual operators: dilation (based on Minkowski sum) and erosion (based on Minkowski subtraction). The morphological image analysis theory has undergone important changes since its birth in the 60s. This evolution is due in part to the fact that these two basic operators are well defined and implemented for images, because of the regular sampling used for digital image representation. If we want to benefit from achievements of mathematical morphology (dilation, erosion, opening, closing, skeletons, filtering, classification, segmentation, etc.) and to use them in the domain of polyhedra—we speak about polyhedral morphology in this case, we need to define and implement these two basic operators in an exact, robust, and efficient manner. Unfortunately, even one of these two operators, that is, the Minkowski sum is not well defined and implemented for polyhedra. This difficulty mainly comes from the nature of polyhedra, which are boundary representations of compact sets in $\mathbb{R}^3$.

The same discussion applies for other domains where the computation of the Minkowski sum of polyhedra is crucial for tasks such as the computation of collision-free path of translational robots, the penetration depth estimation between polyhedra, the computation of the shortest separation distance between two models, etc.

1.2 Overview of Our Previous and Current Works

In the literature, two main approaches are used for the Minkowski sum computation of polyhedra. The first one is based on the convex decomposition of polyhedra and the union of pairwise convex Minkowski sums. The second one is based on the convolution operation that computes a superset of the Minkowski sum and on the winding number computation for the trimming of the superset. On the one hand, the first approach is relatively easy to implement but suffers from the large size of the decomposition and from the challenging union computation of the large number of convex pairwise Minkowski sums. On the other hand, the second approach is difficult to implement because it is not a trivial task to maintain 3D arrangements and to compute the winding number robustly in $\mathbb{R}^3$. Other efficient approaches approximate the Minkowski sum by keeping the same topology as the exact Minkowski sum and by bounding the approximation error. However, not all applications tolerate the approximated results, such as 3D mesh filtering, computer-aided manufacturing, translational robot motion planning in the presence of tight passages, etc. So, in this work, we are trying to add some contribution to the existing work on Minkowski sum of polyhedra and address at the same time the exactness, efficiency, and robustness issues. More precisely, we propose an exact, efficient, and robust contributing vertices-based algorithm for the computation of the Minkowski sum of a nonconvex–convex pair of polyhedra.

We introduced the concept of contributing vertices for the first time in Barki et al. [2009a]; this concept is related to the sweep-based definition of the Minkowski sum (Eq. (1)). It allowed us to develop the Contributing Vertices-based Minkowski Sum (CVMS) algorithm: an exact, efficient, and degeneracy-free algorithm for the computation of the Minkowski sum of convex polyhedra. The experimentation we performed showed that our CVMS algorithm outperforms the other state-of-the-art algorithms presented in Fogel and Halperin [2007], Hachenberger et al. [2007], Weibel and Wu et al. [2003]. The contributing vertices are defined as follows.

Definition 1. The *contributing vertex* $v_{k,B}$ associated to a particular facet $f_{i,A}$ with an outer normal $n_{i,A}$ is the vertex of B which is at a maximal distance away from the supporting plane of $f_{i,A}$. Formally,

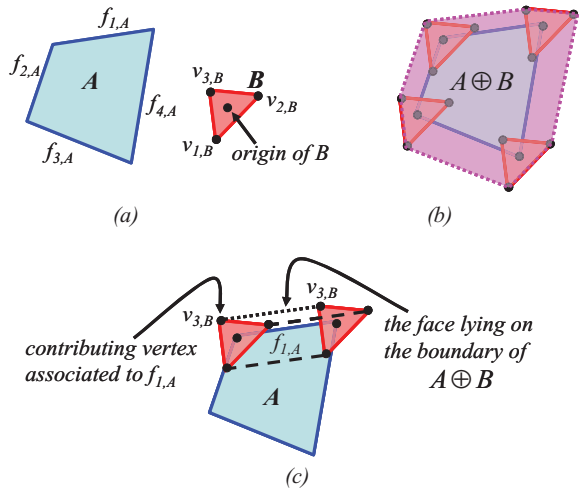

Fig. 3. (a) Two convex polygons A and B; (b) the Minkowski sum $A \oplus B$ depicted as a sweep; (c) the contributing vertex $v_{3,B}$ associated to the face $f_{1,A}$. The dotted line is the face that lies on the boundary of $A \oplus B$. The two other faces (dashed lines) are discarded.

the contributing vertex $v_{k,B}$ of a particular facet $f_{i,A}$ satisfies

$$\langle v_{k,B}, n_{i,A} \rangle = \max_{\mathbb{R}} \langle v_{l,B}, n_{i,A} \rangle \; \forall l = 1, 2, \ldots, v_B, \qquad (2)$$

where $\langle . , . \rangle$ denotes the scalar product. The contributing vertices associated to the facets of B can be computed in a similar manner by interchanging A and B in Eq. (2).

An illustration of the concept of contributing vertices in 2D is shown in Figure 3.

For two convex polyhedra A and B, we classified the Minkowski sum facets into three categories: translated facets, corner facets, and edge facets. The translated facets are computed by means of translations or planar Minkowski sums of the facets of the first polyhedron A. These translations and planar Minkowski sums are defined by the contributing vertices computed for the facets of A. The corner facets of the Minkowski sum polyhedron are computed in a similar manner by considering the facets of the second polyhedron B and their associated contributing vertices. Finally, the edge facets are computed through the use of the visibility and the normal orientation criteria [Barki et al. 2009a].

In Barki et al. [2009b], we took benefit from the concept of contributing vertices and proposed a novel CVMS algorithm for the computation of the Minkowski sum of a nonconvex–convex pair of polyhedra without fold. First, we generated a superset of the Minkowski sum facets by exploiting the concept of contributing vertices and by summing only the features (facets, edges, and vertices) of the input polyhedra that have coincident orientations. The envelope of this superset is the Minkowski sum polyhedron. Secondly, we extracted the exact Minkowski sum boundary from the generated superset by 3D envelope computation. However, this approach is limited to polyhedra whose boundary is completely recoverable from three orthographic projections defined with respect to (w.r.t.) three orthogonal basis vectors in $\mathbb{R}^3$. The limitation of this approach is due principally to the use of 3D envelope computation which assumes that each boundary point, to be recovered, must be seen from outside the superset (or one can shoot a ray from that point to infinity without intersecting any other facet) in at least one of three orthogonal directions—hence the definition of polyhedra without fold. Figure 4 taken from Barki et al. [2009b] depicts a generated Minkowski sum superset for a star-shaped nonconvex polyhedron A and a convex polyhedron B (a geodesic sphere).

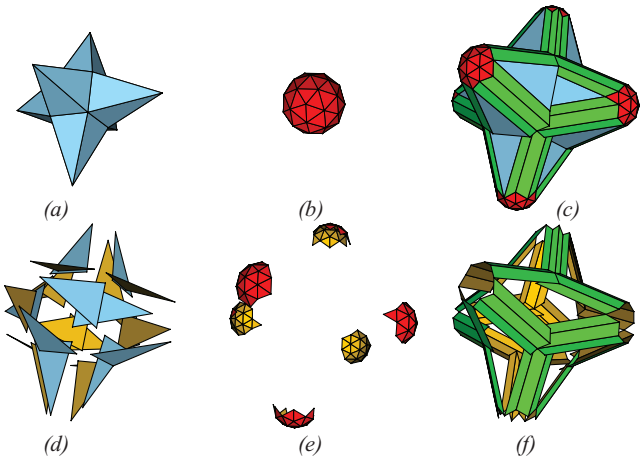

Fig. 4. (a) A nonconvex polyhedron A; (b) a convex polyhedron B; (c) the entire superset of the Minkowski sum facets of $A \oplus B$. (d) The translated facets of $A \oplus B$. (e) The corner facets of $A \oplus B$. (f) The edge facets of $A \oplus B$.

The new algorithm we propose in this work benefits from the superset already generated, but it extracts the Minkowski sum polyhedron in a different manner which will be proven to be more efficient. Moreover, it is not restricted to polyhedra without fold, so it can be applied to any nonconvex–convex pair of polyhedra. It can be seen as belonging to the same class of CVMS algorithms presented in Barki et al. [2009a; 2009b], but is a renewal or an improvement of these two algorithms. For these two reasons, we call our algorithm the NCC-CVMS algorithm. While the CVMS abbreviation stands for "Contributing Vertices-based Minkowski Sum" as for the two previous algorithms, the prefix NCC which stands for "Non-Convex–Convex" distinguishes our algorithm in the current article.

The algorithm proceeds as follows: (1) it computes the superset of the Minkowski sum facets through the use of the contributing vertices concept, (2) it computes the arrangements induced by the intersections among the superset triangles, and (3) it uses the closeness and two-manifoldness properties to extract the Minkowski sum facets and ignore the others (facets lying inside the Minkowski sum mesh). In step (2), we maintain a 2D arrangement for each superset triangle, which eliminates the difficulty arising with 3D arrangements and increases the algorithm's performance. In step (3), we start from a seed facet that is guaranteed to belong to the Minkowski sum polyhedron, then we examine its neighborhood to find the facets that satisfy the two-manifoldness property and belong to the Minkowski sum polyhedron boundary. This process is repeated by considering the recently found neighborhood facets as seed facets until the closeness property is satisfied, that is, until the resulting mesh is closed. By doing so, we avoid the challenging computation of the winding numbers for the arrangements subdivisions and also gain additional performance. In addition to the three mentioned steps, there is some intermediate processing which aims at handling nonmanifold situations and avoiding degeneracies. This intermediate processing will be discussed later in Section 3. Our NCC-CVMS algorithm also

extracts polyhedral holes inside the outer boundary of the Minkowski sum mesh through the use of the closeness and two-manifoldness properties. The handling of nonmanifold situations is detailed in Section 3.3 and the extraction of polyhedral holes is discussed in Section 3.4.

The superset generation step which is the first step of our NCC-CVMS algorithm is not new in this article and will not be discussed. Readers interested in this step can refer to our earlier work [Barki et al. 2009b] were it was proposed for the first time.

1.3 Article Organization

This article is organized as follows: a literature review is presented in Section 2. In Section 3, we describe our algorithm for the extraction of the Minkowski sum from the superset of facets. Finally, we present some implementation details, a performance benchmark, a comparative study, and some results.

2. RELATED WORK

The Minkowski sum of two convex polyhedra A and B is obtained by performing the vector addition of all points of A and B and by computing the convex hull of the resulting points. This process takes $O(mn)$ time for polyhedra with m and n features. For two nonconvex polyhedra, the most common approach is based on convex decomposition of polyhedra. It takes $O(m^3n^3)$ time [Halperin 2002] to compute the sum by decomposing each nonconvex polyhedron into convex pieces [Chazelle 1981], computing the pairwise Minkowski sums of all possible pairs of convex pieces, and performing the union of the pairwise Minkowski sums [Aronov et al. 1997]. The main bottleneck of this approach is the union step which is very time consuming.

Another decomposition method was proposed by Evans et al. [1992]. The authors decomposed the boundary of the input polyhedra into affine cells (instead of convex pieces), computed the pairwise Minkowski sums of transversal affine cells, and performed their union. The decomposition into affine cells yields more pieces than does the decomposition into convex pieces.

Recently, Hachenberger [2007] presented the first exact and robust implementation of a convex decomposition-based Minkowski sum algorithm that performs in $O(m^3n^3)$ time for polyhedra with m and n features through the use of Nef polyhedra.

A Nef polyhedron in $\mathbb{R}^d$ is a point set resulting from a finite number of open halfspaces by set complement and set intersection operations. In $\mathbb{R}^3$, a finite number of planes (defining a finite number of halfspaces) partitions the space into cells of various dimensions (vertices, edges, facets, and volumes). Such a partition augmented with a label for each of its cells is called a Selective Nef Complex (SNC) [Hachenberger et al. 2007]. The selective Nef complex is called a Nef polyhedron in $\mathbb{R}^3$ if the labels are Boolean (in, out). The Boolean labels are called set-selection marks. Nef polyhedra result from a finite number of halfspaces by set complement and intersection operations. The other Boolean set operations (union, difference, and symmetric difference) reduce to set intersection and complement. Moreover, topological operations (boundary, interior, exterior, and closure) that change between open and closed halfspaces are also handled by Nef polyhedra.

In his convex decomposition-based Minkowski sum algorithm, Hachenberger [2007] implemented an optimal convex decomposition algorithm similar to that proposed by Chazelle [1981]. He decomposed nonconvex polyhedra by inserting walls that resolve the reflex edges until all reflex edges are treated. He computed the union of the pairwise Minkowski sums through the use of Nef polyhedra which handle Boolean operations well.

To avoid drawbacks related to convex decomposition-based approaches, Varadhan and Manocha [2006] approximated the union of the pairwise Minkowski sums by using an adaptively subdivided voxel grid. They used Iso-surface extraction and guaranteed the same topology as the exact Minkowski sum. Recently, Lien [2007] used a point-based representation instead of the mesh-based one. He constructed a point set by adding all points from two point sets uniformly sampled from the boundary of two polyhedra. He used three filters (a collision detection, normal, and octree filter) to discard inner points and showed several applications of the point-based representation. In the worst case (when the summands are convex), the point-based approach has a time complexity of $O(mnT_{filter})$, where m and n are the sizes of the point sets sampled from the summands and T_{filter} is the time complexity of filtering a single point.

In his recent work, Lien [2008] proposed a nearly exact algorithm for the computation of the Minkowski sum of polyhedra. First, he computed a brute-force convolution of two input polyhedra in order to generate a superset of the Minkowski sum. Then, he subdivided each facet of this superset into subfacets induced by the facet-facet intersections. After that, he stitched the subfacets into simple regions which are either entirely on the boundary of the Minkowski sum or entirely inside it. Finally, he used a collision detector to remove the regions inside the Minkowski sum and to keep only those lying on its boundary.

Some dual space-based approaches have been investigated for the computation of the Minkowski sum. Ghosh [1993] presented a unified computational framework for the Minkowski sum of polygons and polyhedra. Polyhedra are represented in a dual space: the slope diagram. The sum polyhedron results from the merging of two slope diagrams. The existent implementations of slope diagram-based algorithms work only for convex polyhedra. Other slope diagram variants can be found in Barki et al. [2009a], Bekker and Roerdink [2001], Fogel and Halperin [2007], and Wu et al. [2003].

Guibas et al. [1983, 1987] defined the operation of convolution on planar tracings in 2D. Basch et al. [1996] extended this definition to polyhedral tracings and used it to generate a superset of the Minkowski sum. They used arrangement computations to extract the exact boundary of the Minkowski sum. Their algorithm computes the convolution Q of two polyhedral tracings of size m and n in an output-sensitive time $O(k\alpha(k)log^3k)$, where k is the size of the convolution Q and $\alpha(k)$ is the familiar inverse Ackermann function. Nevertheless, at the best of our knowledge, the convolution-based approach for polyhedra has not been implemented yet.

At first glance, our NCC-CVMS algorithm appears to be similar to the convolution-based approach (since it relies on arrangement computations). However, the two approaches differ for the following reasons: (1) our NCC-CVMS algorithm generates the Minkowski sum facets superset more efficiently than the convolution does. This is due to the use of the contributing vertices concept. (2) the convolution-based approach deals with 3D arrangement while the NCC-CVMS algorithm deals only with 2D arrangements, and (3) the convolution-based approach computes the winding number of all 3D arrangement subdivisions in order to perform trimming and to compute the Minkowski sum while the NCC-CVMS algorithm uses only the closeness and the two-manifoldness simple properties for the same task.

A comparative study of our NCC-CVMS algorithm, the topologically accurate approximation of Varadhan and Manocha [2006], and the nearly exact method of Lien [2008] can be found in Section 4.3.

3. EXTRACTION OF THE MINKOWSKI SUM MESH FROM THE SUPERSET

In the rest of this article, we will consider a nonconvex polyhedron A and a convex polyhedron B. We denote by $F = \{f_1, f_2, \ldots, f_n\}$ the facets superset and by $S = A \oplus B$ the Minkowski sum polyhedron. We will sometimes replace the term "Minkowski sum" by its abbreviation "MS."

The intersections among the superset facets (which are 3D polygons) are maintained by the use of 2D arrangements. So, we will first explain the correspondence between 3D features (3D facets, 3D segments, and 3D points) and the equivalent features in 2D (2D arrangement faces, 2D arrangement segments, and 2D arrangement points or vertices). After that, we will explain the MS boundary extraction algorithm.

3.1 The Correspondence between the 3D MS Superset Facets and the 2D Arrangements

Let us consider an arbitrary facet f_i from the superset F, denote its outer normal by $n_i(x_{n_i}, y_{n_i}, z_{n_i})$, and denote its supporting plane by P_i. To represent the 3D facet f_i by a 2D arrangement $arr(f_i)$, we need a bijection (or a one-to-one correspondence) that associates a 2D point in $arr(f_i)$ to each 3D point lying on f_i. Such a bijective function must be degeneracy-free, simple to implement, and lightweight in terms of computation time.

A trivial solution consists in comparing the absolute values of the coordinates x_{n_i}, y_{n_i}, and z_{n_i} of the outer normal n_i and ignoring the coordinate that has the largest absolute value. Therefore, given a 3D point $p_{f_i}(x, y, z)$ lying on f_i, the corresponding 2D arrangement projected point $p_{arr(f_i)}$ is computed as follows. We have

$$x_{arr} = y \text{ and } y_{arr} = z, \quad |x_{n_i}| = max(|x_{n_i}|, |y_{n_i}|, |z_{n_i}|), \tag{3}$$

$$x_{arr} = z \text{ and } y_{arr} = x, \quad |y_{n_i}| = max(|x_{n_i}|, |y_{n_i}|, |z_{n_i}|), \tag{4}$$

$$x_{arr} = x \text{ and } y_{arr} = y, \quad |z_{n_i}| = max(|x_{n_i}|, |y_{n_i}|, |z_{n_i}|), \tag{5}$$

where x_{arr} and y_{arr} are the respective x and y coordinates of $p_{arr(f_i)}$ in the 2D arrangement $arr(f_i)$ coordinate system.

The inverse bijective projection can be simply determined by using the supporting plane equation and the two retained coordinates. Suppose that P_i is defined by $a_i x + b_i y + c_i z + d_i = 0$, the ignored coordinate value of the facet point can be computed by substituting the values of the two known coordinates in the plane equation.

Throughout the subsequent sections, whenever we talk about superset facets and the associated arrangements, the bijective projection presented in the current subsection is implicitly referred to.

3.2 The MS Polyhedron Extraction Algorithm in Depth

As stated before, our NCC-CVMS algorithm starts by the generation of the superset of the Minkowski sum facets as done in Barki et al. [2009b]. The 3D envelope of the superset is the Minkowski sum polyhedron itself. However, some parts of the superset facets lie inside the Minkowski sum polyhedron. Therefore, a trimming is necessary to remove these "inside" parts of facets, which implies to handle the intersections among the superset facets.

Let us start by giving an outline of our NCC-CVMS approach in Algorithm 1. The implementation details of each step are deferred to Section 4.1.

Given a nonconvex–convex pair of polyhedra A and B, we denote by F the corresponding Minkowski sum facets superset. *MS_stack* denotes the stack used to manipulate the Minkowski sum facets, which contains 2D arrangement faces. *MS_top* denotes a variable of type "2D arrangement face" storing the top of the stack. The NCC-CVMS algorithm proceeds as follows.

Algorithm 1: An outline of the NCC-CVMS algorithm for a nonconvex–convex pair of polyhedra

Input: A non-convex–convex pair of polyhedra A and B
Output: The Minkowski sum polyhedron $S = A \oplus B$

1 Generate the Minkowski sum facets superset F for A and B;
2 Merge the "coplanar conjoint" superset facets and compute the symmetric difference of the "opposite conjoint" superset facets;
3 Triangulate the MS superset facets;
4 Compute the Iso-oriented bounding boxes for the superset triangles and find the boxes' intersections;
5 Compute the intersection of the superset triangles having intersecting bounding boxes;
6 Construct the 2D arrangements (subdivisions of the superset triangles into sub-facets) corresponding to each superset triangle from the intersection segments;
7 Compute the seed sub-facet by finding the lexicographically smallest or greatest point of the superset;
8 Push the arrangement face associated to the seed sub-facet into *MS_stack*;
9 **repeat**
10 *MS_top* ← *MS_stack.top*;
11 *MS_stack.pop* (pop *MS_stack*);
12 Find the neighbors of *MS_top* (2D arrangement faces associated to the superset sub-facets that are incident to the edges of the sub-facet associated to the arrangement face *MS_top*);
13 Among the neighboring sub-facets, determine those belonging to the boundary of the MS polyhedron by using the closeness and two-manifoldness properties;
14 Tag the MS sub-facets appropriately;
15 Update the *MS_stack* (push the *MS_top* neighboring MS sub-facets into it);
16 **until** *(MS_stack is empty)*;
17 Get the Minkowski sum polyhedron facets from the 2D arrangement faces;
18 Triangulate the Minkowski sum polyhedron facets (optional);

Step (1) Generate the Minkowski sum facets superset F for A and B. This step aims at computing a superset of the Minkowski sum facets. Further details can be found in Barki et al. [2009b].

Step (2) Merge the "coplanar conjoint" superset facets and compute the symmetric difference of the "opposite conjoint" superset facets. Two facets lying on the same supporting plane are said "coplanar conjoint" if the intersection of their interior is not empty. The interior of a facet is defined by the set of points belonging to this facet except those belonging to its incident edges and vertices. Two facets lying on two opposite supporting planes are said "opposite conjoint" if there exists at least a common point in their interior.

The principle of this step is the following.

(1) Partition the superset facets $F = \{f_1, f_2, \ldots, f_n\}$ into several clusters $c_i | i = 1, 2, \ldots, m$ $(m \leq n)$ so that each cluster c_i is identified by a unique supporting plane P_i.

(2) Compute the union of the "coplanar conjoint" facets of each cluster and replace these "coplanar conjoint" facets by their union in the superset F (Figure 5(a)).

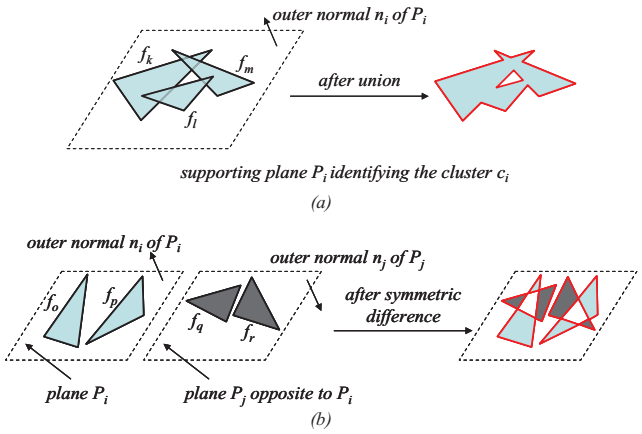

Fig. 5. (a) The coplanar conjoint facets f_k, f_l, and f_m of the cluster c_i are merged to give a unique facet; (b) the symmetric difference between the facets f_o and f_p lying on the plane P_i and the facets f_q and f_r lying on the plane P_j opposite to P_i is computed and replaces these facets in the superset F.

(3) Find all pairs of opposite supporting planes among those identifying the clusters c_i.

(4) Compute the symmetric difference of the facets lying on each pair (c_i, c_j) identified by opposite planes P_i and P_j and replace these "opposite conjoint" facets by their symmetric difference in the superset F (Figure 5(b)).

Merging "coplanar conjoint" facets avoids some degeneracies and simplifies the steps (5) and (13) as detailed shortly. It aims at removing the superimposed parts of the superset facets (common intersection regions for "coplanar conjoint" facets) and replacing them by a unique region. It does not affect the correctness of the subsequent steps since it preserves the closeness property. Computing the symmetric difference of the "opposite conjoint" facets allows to handle some nonmanifold situations (Section 3.3). It does not introduce holes and does not affect the correctness of our algorithm because it removes parts of the superset that are necessarily inside the volume of the Minkowski sum polyhedron.

Step (3) Triangulate the MS superset facets. In order to perform the trimming operation on the MS superset, we must handle intersections between the facets $f_1, f_2, \ldots, f_n$ after the previous step has been accomplished. The naive computation of the intersections between these geometric primitives is expensive and difficult at the same time because some facets contain holes (as shown in Figure 5(a)), others are convex, and others are nonconvex. To ease the solution of this problem, we triangulate all the facets so that the problem reduces to the computation of the intersections between 3D triangles.

After this step, the superset F will be composed of n_T triangles, that is, $F = \{t_1, t_2, \ldots, t_{n_T}\}$, $(n_T \geq n)$ where n is the number of polygonal facets in F and n_T is the number of triangles in F.

Step (4) Compute the Iso-oriented bounding boxes for the superset triangles and find the boxes' intersections. The aim of this step is to speed up the computation of the intersections among the superset triangles by using bounding boxes. The exact intersection among two triangles is computed only when the corresponding bounding boxes intersect. We chose the algorithm of Zomorodian and Edelsbrunner [2002] which operates on Iso-oriented bounding boxes aligned with the coordinate axes. Two Iso-oriented boxes intersect if and only if they intersect in every dimension independently. So the algorithm of Zomorodian and Edelsbrunner [2002] reduces the 3D intersection problem into 1D interval intersection problem which can be solved in a simple and efficient manner.

Step (5) Compute the intersection of the superset triangles having intersecting bounding boxes. For that purpose, we use a slightly modified version of the "interval overlap method" presented in Möller [1997]. The principle of this method is to: (1) reject pairs of nonintersecting triangles (pairs of triangles whose vertices are entirely on one side of the other triangle's supporting plane), (2) compute the intersection line of the supporting planes of the two triangles, and (3) find the overlap of the two segments at which this line is clipped by the two triangles. If the segments overlap, the triangles intersect.

For each triangle t_i of the superset F, we maintain a list *intersection*(t_i). Each element in the list is a structure object composed of a reference to a triangle t_j intersecting t_i along with the segment resulting from this intersection.

The merging process of "coplanar conjoint" facets exhibits its importance when computing intersections since two "coplanar conjoint" triangles intersection results in a 3D polygon and not a 3D segment. This polygon is not suitable for the construction of the planar arrangements whose features are induced by the insertion of (intersection) line segments into them and its computation is time consuming. Therefore, a simple solution is to prevent such coplanar conjoint triangles from passing through the triangle-triangle intersection step by a previous merging process.

Step (6) Construct the 2D arrangements (subdivisions of the superset triangles into subfacets) corresponding to each superset triangle from the intersection segments. This step aims at constructing planar (or 2D) arrangements for the n_T triangles of the superset F. A planar arrangement is the subdivision of the plane into 0D, 1D, and 2D cells, denoted vertices, edges, and faces of the arrangement, respectively [Wein et al. 2008; Agarwal and Sharir 1998]. This subdivision is induced by a set of curves which may intersect each other. In our case, these curves are line segments.

The 2D arrangement $arr(t_i)$ corresponding to a superset triangle t_i, $i = 1, 2, \ldots, n_T$ of F results from the intersection of other superset triangles with the triangle t_i. It is constructed by projecting the intersection segments stored in the list *intersection*(t_i) into $arr(t_i)$.

Figure 6 depicts the construction of the planar arrangement $arr(t_i)$ associated to a triangle t_i according to the intersection segments $e_{1,i}$, $e_{2,i}$, $\ldots$, $e_{7,i}$. We distinguish two kinds of intersection configurations or intersection segments in the MS superset. The "edge-edge" intersection configuration results from two superset triangles sharing a common edge, that is, two superset triangles intersecting only in their boundaries. The intersections between the triangles t_i and t_l (edges $e_{1,i}$ and $e_{2,i}$), between the triangles t_i and t_m (edge $e_{3,i}$), and between the triangles t_i and t_n (edge $e_{4,i}$ and $e_{5,i}$) are examples of the "edge-edge" intersection configuration. The "interior" intersection configuration results from the intersection of a superset triangle which crosses another superset triangle in its interior. The intersections between the triangles t_i and t_j (edge $e_{6,i}$) and between the triangles t_i and t_k (edge $e_{7,i}$) depicted in Figure 6 are examples of the "interior" intersection configuration. Because we have previously merged the "coplanar conjoint" facets, there will be no other intersection configuration such as two coplanar triangles having common interior region. The case of an edge intersecting the interior of another triangle is handled as an "interior" intersection configuration.

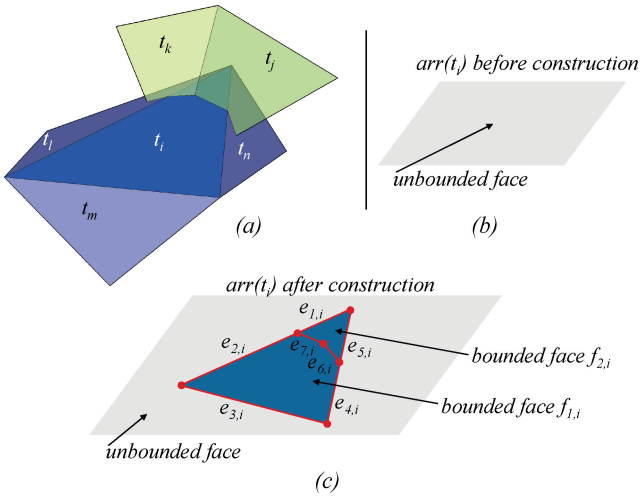

Fig. 6. (a) A list of superset triangles intersecting the triangle t_i; (b) the planar arrangement $arr(t_i)$ before the insertion of intersection segments. It comprises an unbounded face; (c) the arrangement $arr(t_i)$ after the insertion of the intersection segments $e_{1,i}, e_{2,i}, \ldots, e_{7,i}$.

Each arrangement $arr(t_i)$ comprises vertices, bounded edges, and bounded faces that constitute a hole inside a unique unbounded face (see Figure 6(b) and 6(c)). The arrangements we are maintaining in our algorithm are planar or 2D arrangements embedded on the x-y, y-z, or z-x planes through the use of the bijective projection previously described. This approach reduces the problem dimensionality, is an easier task, and is much more efficient than working directly with 2D arrangements embedded on the triangles' supporting planes.

The planar arrangement $arr(t_i)$, when projected back onto the supporting plane of t_i, is a subdivision of the triangle t_i into bounded subfacets. In order to simplify our discussion, we will not mention the bijective projection mechanism and simply say that the planar arrangement $arr(t_i)$ is a subdivision of the triangle t_i into bounded subfacets.

Let us demonstrate that each bounded subfacet of the subdivision of t_i has a unique membership state w.r.t. the Minkowski sum polyhedron.

PROPOSITION 1. *Let us consider a triangle t_i, the list of its intersection segments intersection(t_i) with the other triangles of the MS superset F, and the associated planar arrangement $arr(t_i)$, which defines the subdivision of t_i into n_i bounded subfacets $f_{1,i}, f_{2,i}, \ldots, f_{n_i,i}$. A bounded subfacet $f_{m,i}$, $1 \leq m \leq n_i$ either lies completely on the boundary of the Minkowski sum polyhedron or lies completely inside this boundary (or inside the Minkowski sum polyhedron).*

PROOF. Let us do it by contradiction. Consider an arbitrary subfacet $f_{m,i}$ and assume that it lies partly on the boundary of the Minkowski sum and partly inside the Minkowski sum boundary. This assumption implies that $f_{m,i}$ crosses the Minkowski sum boundary in order to have a part of it on the MS boundary and the other part of it inside the MS boundary. But this crossing is impossible since the arrangement $arr(t_i)$ was induced by all the possible intersections of the triangle t_i and the other triangles of the superset, so there is no intersection or crossing inside the subfacet $f_{m,i}$. We conclude that $f_{m,i}$ either lies completely on the MS boundary or lies completely inside it ($f_{m,i}$ cannot lie outside the MS boundary because the superset is the result of the addition of the features of A and B). □

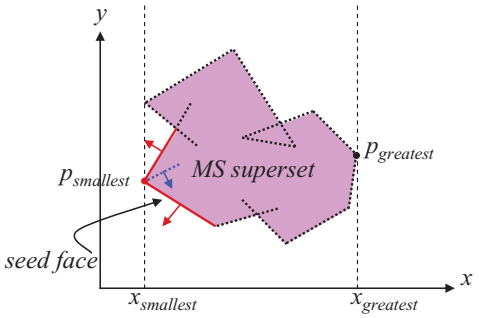

Fig. 7. The selection of the seed face (edge) of an MS superset in 2D. The seed face can be any one of the two faces (edges) drawn as solid lines.

As a result of this step, we have a planar arrangement $arr(t_i)$ associated to each superset triangle t_i, $1 \leq i \leq n_T$. These planar arrangements define subdivisions of the superset triangles into subfacets, each subfacet has a unique membership state (either it lies completely on the MS boundary or it lies completely inside it).

Step (7) Compute the seed subfacet by finding the lexicographically smallest or greatest point of the superset. We already said that the MS polyhedron S is the envelope of the generated MS superset F. This implies that all the subfacets of the superset that are visible from outside it belong to the MS boundary. Thus, knowing that the visibility state does not change inside a subfacet, it is sufficient to find only one visible point inside a subfacet to conclude that this subfacet lies on the MS boundary. We will demonstrate that a seed subfacet can be chosen by taking a subfacet incident to the lexicographically smallest (or greatest) point among the superset vertices so that the other subfacets incident to the lexicographically smallest (or greatest) point lie on the negative side of its supporting plane.

PROPOSITION 2. *A subfacet $f_{m,i}$ of the planar arrangement $arr(t_i)$ incident to the lexicographically smallest point $p_{smallest}$ of the MS superset so that all the other subfacets incident to the lexicographically smallest point $p_{smallest}$ are on the negative side of its supporting plane is a facet of the boundary of the MS polyhedron.*

PROOF. Since $p_{smallest}$ is the lexicographically smallest point of the superset F, its x coordinate value is the smallest among all x coordinate values of the superset. So, we can trace a ray parallel to the x-axis starting from $p_{smallest}$ towards the negative infinite values of x without intersecting any MS superset feature. Therefore, $p_{smallest}$ is visible from outside the MS superset and at least one of the subfacets incident to it is also visible from outside the superset. Suppose that the list of subfacets incident to $p_{smallest}$ has been determined and suppose that $f_{m,i}$ is a subfacet of this list so that all other subfacets of this list lie on the negative side of the supporting plane of $f_{m,i}$. This means that $f_{m,i}$ is not occluded by the other subfacets (because they lie on the negative side of its supporting plane). Therefore, $f_{m,i}$ is visible from outside the MS superset. In conclusion, since $f_{m,i}$ is visible from outside the MS superset, it is an MS subfacet. □

The proof remains the same if we choose the lexicographically greatest point. Only the direction of the traced ray must be reversed (towards the positive infinite x values).

Figure 7 depicts an MS superset in 2D, the corresponding lexicographically smallest point $p_{smallest}$, and the lexicographically

greatest point $p_{greatest}$. In 2D we deal with edges instead of facets. So the edges or 1D-faces incident to $p_{smallest}$ are determined first. After that, the seed edge or 1D-face is chosen so that the other edges incident to $p_{smallest}$ lie on the negative side of its supporting line. In Figure 7, the seed edges are drawn as solid lines. The same logic applies if we have chosen $p_{greatest}$.

Once the seed subfacet has been found, the computation of the other Minkowski sum facets by neighborhood traversal can be started.

Step (8) Push the arrangement face associated to the seed subfacet into MS_stack. The aim of this step is to initialize *MS_stack* with the arrangement face corresponding to the seed subfacet found in the previous step.

Steps (9) to (16) Traversal of the seed subfacet neighborhood and determination of the MS boundary subfacets. In step (8), *MS_stack* is initialized with the arrangement face corresponding to the seed subfacet determined in step (7). This arrangement face is tagged as "MS face" because it lies on the boundary of the MS polyhedron and also as "already stacked" because it is pushed into *MS_stack*. The tag "already stacked" prevents the arrangement faces from being pushed into *MS_stack* more than once, thus it avoids processing an arrangement face and the corresponding superset subfacet more than once.

The processing of *MS_stack* or the "repeat-until" loop iterations (lines 9 to 16) are intended to: (1) store the arrangement face at the top of *MS_stack* in *MS_top*, a variable used to backup the value at the top of *MS_stack* before popping it, (2) pop *MS_stack*, (3) find the subfacets in the neighborhood of the subfacet f_{MS_top} associated to the arrangement face *MS_top*, that is, find the subfacets that are incident to the edges of f_{MS_top}, (4) among these neighboring subfacets, determine those that satisfy the closeness and two-manifoldness properties, tag them as "MS face" because they necessarily lie on the boundary of the MS polyhedron, and (5) push the subfacets recently tagged as "MS face" into *MS_stack* and tag them as "already stacked."

At the beginning, *MS_top* is the arrangement face associated to the seed subfacet. At subsequent iterations, *MS_top* will be one of the arrangement faces associated to one of the MS subfacets neighboring the subfacet f_{MS_top} of the previous iteration.

Let us consider the arrangement face *MS_top* at an arbitrary iteration and its associated 3D subfacet f_{MS_top}, which lies in the subdivision of a superset triangle t_i (see Figure 8(a) and 8(b)). It is clear that f_{MS_top} is a subfacet of the Minkowski sum polyhedron since the stacked faces are arrangement faces tagged as "MS-face" and associated to subfacets lying on the boundary of the Minkowski sum polyhedron. Furthermore, let us consider an arbitrary 3D segment or edge e_j of the subfacet f_{MS_top} and let us denote by $e_{j,i}$ the arrangement edge of $arr(t_i)$ associated to e_j. We will search for the subfacets sharing e_j.

To simplify the discussion, we will suppose first that each arrangement edge $e_{j,i}$ belonging to the arrangement $arr(t_i)$ results from the intersection of exactly two superset triangles, that is, there are no three or more superset triangles intersecting at the same 3D segment e_j. The general case will be discussed later.

To determine the intersection configuration of the 3D segment or edge e_j of the subfacet f_{MS_top}, we examine the planar arrangement $arr(t_i)$ of the triangle t_i. More precisely, we examine the arrangement edge $e_{j,i}$ which is the projection of e_j in the planar arrangement $arr(t_i)$. $e_{j,i}$ is incident to the arrangement face *MS_top* of $arr(t_i)$ associated to f_{MS_top} and to another arrangement face $f_{n,i}$. If the second face $f_{n,i}$ incident to $e_{j,i}$ is the unbounded face of the arrangement $arr(t_i)$, we conclude that the intersection configuration

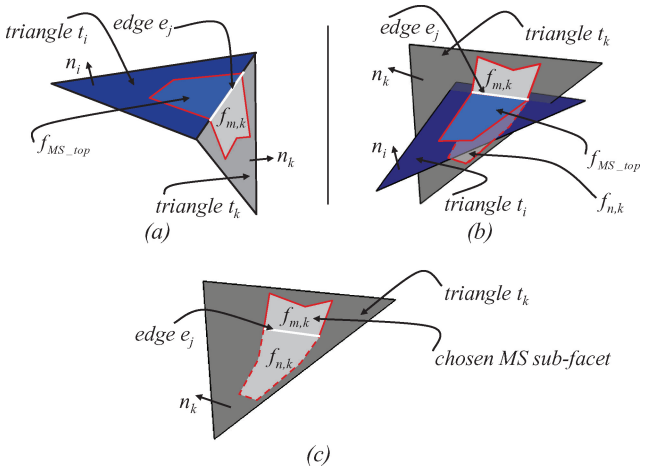

Fig. 8. Determination of the MS subfacet $f_{m,k}$ incident to the edge e_j. (a) The edge e_j has an "edge-edge" intersection configuration; (b) the edge e_j has an "interior" intersection configuration, two subfacets $f_{m,k}$ and $f_{n,k}$ are incident to it; (c) the subfacet $f_{m,k}$ is the MS subfacet.

corresponding to e_j is an "edge-edge" one because if e_j resulted from the intersection of another triangle crossing t_i in its interior, $f_{n,i}$ would be a bounded face corresponding to a subfacet of the triangle t_i. Otherwise, if $f_{n,i}$ is a bounded face of the arrangement $arr(t_i)$, we conclude that the intersection configuration corresponding to the edge e_j is an "interior" one.

The two-manifoldness and closeness properties imply that each edge of the MS polyhedron is incident to exactly two subfacets. The 3D segment e_j is incident to the subfacet f_{MS_top} and to another subfacet of the MS polyhedron that must be determined according to the intersection configuration of e_j.

If the intersection configuration is of type "edge-edge" (see Figure 8(a)), it exists only one subfacet $f_{m,k}$ incident to e_j and adjacent to f_{MS_top}. This subfacet belongs to the subdivision of another superset triangle t_k which is the unique triangle intersecting t_i at the edge e_j as previously assumed. The subfacet $f_{m,k}$ can be found by examining the list of intersection segments $intersection(t_i)$ that gives a reference to the superset triangle t_k intersecting t_i at e_j. Therefore, the subfacet $f_{m,k}$ is the MS subfacet we are searching for because e_j must be incident to two subfacets (f_{MS_top} and $f_{m,k}$). The subfacet $f_{m,k}$ is tagged as "MS face," pushed into *MS_stack*, and tagged as "already stacked" subfacet.

If the intersection configuration is of "interior" type (see Figure 8(b) and 8(c)), only two subfacets $f_{m,k}$ and $f_{n,k}$ incident to e_j and adjacent to f_{MS_top} exist. These two subfacets belong to the subdivision of another superset triangle t_k which is the unique triangle intersecting t_i at the edge e_j as we previously assumed. $f_{m,k}$ and $f_{n,k}$ can be found by examining the list $intersection(t_i)$, which gives a reference to the superset triangle t_k intersecting t_i at the edge e_j. As e_j must be incident to exactly two MS subfacets, one of which is f_{MS_top}, we must choose only one subfacet among $f_{m,k}$ and $f_{n,k}$ to be the second one.

The closeness property is verified by the choice of $f_{m,k}$ or $f_{n,k}$, but only one of these two subfacets satisfies the two-manifoldness property. Now, consider the case presented in Figure 8(b) and 8(c). If we choose $f_{n,k}$ to be the MS subfacet, the second subfacet $f_{m,k}$ incident to e_j lies outside the boundary of the MS polyhedron. Thus $f_{n,k}$ cannot be chosen as an MS subfacet because there is no subfacet lying outside the MS polyhedron. If we choose $f_{m,k}$ to be the MS

subfacet, the second subfacet $f_{n,k}$ incident to e_j lies inside the MS superset, thus $f_{m,k}$ satisfies the two-manifoldness property of the Minkowski sum polyhedron. From Figure 8(b), we conclude that the subfacet to be considered as the MS subfacet incident to e_j is the subfacet lying on the positive side of the supporting plane of t_i (in our example, $f_{m,k}$ is the MS subfacet). Therefore, $f_{m,k}$ is tagged as "MS face," pushed into *MS_stack*, and tagged as "already stacked" subfacet.

The configuration where the subfacets $f_{m,k}$ and $f_{n,k}$ lie on the same supporting plane of t_i will never happen owing to the merging process of "coplanar conjoint" parts. We do not need to worry about such a configuration.

We have determined the subfacet of the MS superset that is incident to the edge e_j of the subfacet f_{MS_top}. The process can be repeated for the other edges incident to f_{MS_top}. At the end of this process, all MS subfacets adjacent to f_{MS_top} are determined. These MS subfacets are correctly tagged and pushed into *MS_stack*.

The "repeat-until" loop stops when *MS_stack* is empty, that is, when all subfacets of the Minkowski sum boundary have been determined so that the final result is the closed Minkowski sum polyhedron.

It remains to handle the case where more than two triangles intersect at the same segment e_j. As stated before, let us consider the arrangement face *MS_top*, the associated 3D subfacet f_{MS_top} lying on the triangle t_i at an arbitrary iteration and consider an edge e_j of f_{MS_top}. Suppose that e_j results from the intersection of t_i with three superset triangles t_1, t_2, and t_3 (see Figure 9(a)). The discussion remains similar if there were more than three triangles intersecting t_i at e_j.

First, we consider only the superset triangle t_1 intersecting t_i at the edge e_j and determine the subfacet $f_{m,1}$ (lying on t_1) that satisfies the closeness and two-manifoldness properties. This will be performed as before by detecting the intersection configuration of e_j, where we supposed that each edge e_j results from the intersection of exactly two superset triangles. After that, we consider each of the remaining triangles t_2 and t_3 apart and also determine the subfacets $f_{n,2}$ and $f_{p,3}$ satisfying the closeness and two-manifoldness properties as done before for the triangle t_1. In other words, we take each pair of triangles (t_i, t_1), (t_i, t_2), and (t_i, t_3) independently and determine the subfacets that are supposed to be MS facets if the other pairs do not exist. We obtain three subfacets $f_{m,1}$, $f_{n,2}$, and $f_{p,3}$ lying on the triangles t_1, t_2, and t_3 respectively. We must choose one subfacet among these three subfacets to be the MS subfacet incident to e_j and adjacent to f_{MS_top}. As before, any chosen subfacet satisfies the closeness property but only one of them satisfies the two-manifoldness property. The subfacet to be chosen is the one lying on a supporting plane that forms the minimal angle w.r.t. the supporting plane of t_i (see Figure 9(b)). This plane angle criterion guarantees that the other nonchosen subfacets lie inside the MS polyhedron. For the example depicted in Figure 9, the subfacet to be chosen is $f_{n,2}$ lying on the triangle t_2.

As a conclusion to this step, the boundary of the MS polyhedron is determined by finding, for each stacked MS subfacet, the adjacent subfacets satisfying the closeness and two-manifoldness properties.

Step (17) Get the Minkowski sum polyhedron facets from the 2D arrangement faces. At this point, all the MS superset subfacets lying on the boundary of the Minkowski sum polyhedron are properly tagged. Therefore, in order to obtain the Minkowski sum polyhedron facets, we iterate through all 2D arrangements, find the 2D arrangement faces which have been tagged as "MS face," and project them on the supporting planes of the triangles associated to their arrangements.

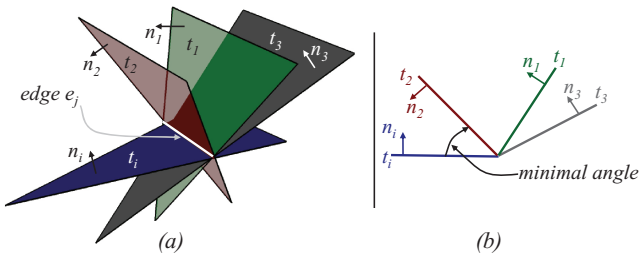

Fig. 9. The plane angle criterion for the determination of the MS subfacet among several others incident to the edge e_j. (a) The triangles t_1, t_2, and t_3 intersect the triangle t_i in the same edge e_j; (b) the minimal angle is the one between the supporting plane of t_i and the supporting plane of t_2. The second MS subfacet incident to e_j is a subfacet lying on t_2.

Step (18) Triangulate the Minkowski sum polyhedron facets (optional). This optional step aims at triangulating the Minkowski sum facets for more convenience to avoid the presence of nonconvex facets.

3.3 Handling of Nonmanifold Situations

As previously mentioned, our NCC-CVMS algorithm handles nonmanifold Minkowski sums. In this subsection, we list all nonmanifold situations and show how to handle them.

Six nonmanifold situations may occur in the superset F: (1) double vertices, (2) a vertex and an edge touching tangentially, (3) a vertex and a subfacet touching tangentially, (4) an edge incident to more than two subfacets, (5) an edge and a subfacet touching tangentially, and (6) two "opposite conjoint" facets.

Because the traversal of the MS boundary subfacets is based on the edges' neighborhood examination through the use of the two-manifoldness and closeness properties, the nonmanifold cases induced by the presence of vertices touching tangentially other primitives (nonmanifold situations (1), (2), and (3) mentioned before) are automatically handled. Indeed, the vertices' neighborhood is not considered during the traversal of the MS boundary.

The nonmanifold situation (4) where an edge is incident to more than two subfacets is also automatically handled. Given a nonmanifold edge incident to more than two subfacets, when examining the MS subfacet f_{MS_top} incident to such a nonmanifold edge at an arbitrary "repeat-until" loop iteration, the two-manifoldness property allows to choose only one subfacet incident to this edge as the MS subfacet. The other MS subfacets incident to this nonmanifold edge are automatically recovered when examining the neighborhood of the other edges incident to these subfacets because the algorithm stops only when the result is a closed mesh. If one of the subfacets incident to this nonmanifold edge was missed, this means that another subfacet incident to the missed subfacet has a border edge, which is not allowed by our algorithm. Therefore, no Minkowski sum subfacet is missed even in this nonmanifold situation.

The situation (5) where an edge touches a subfacet tangentially resumes to the case (4) because this edge is originally incident to two subfacets plus the other subfacets induced by the splitting of the subfacet which touches it tangentially.

Finally, the nonmanifold situation (6) is handled by the intermediary processing of step (2) of our algorithm. Especially, because we already computed the symmetric difference of the "opposite conjoint" facets, this situation does not occur during the traversal of the MS boundary.

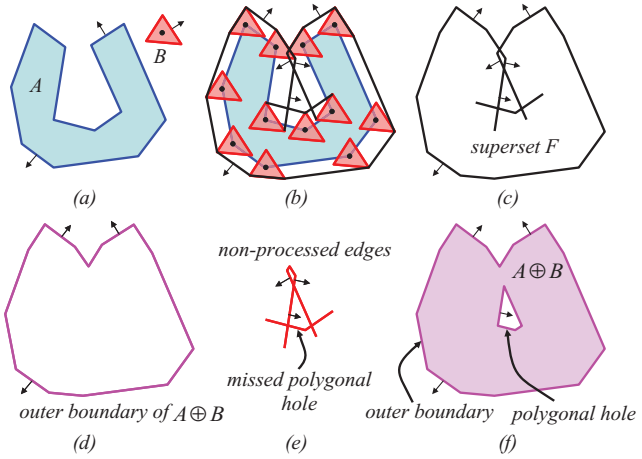

Fig. 10. (a) A nonconvex–convex pair of polygons A and B; (b) and (c) the sweep of A by B and the generation of the superset F; (d) the application of algorithm 1 allows to only compute the outer boundary of the MS; (e) the edges that are not processed by Algorithm 1 and the missed polygonal hole; (f) the correct MS polygon is composed of an outer boundary and a polygonal hole.

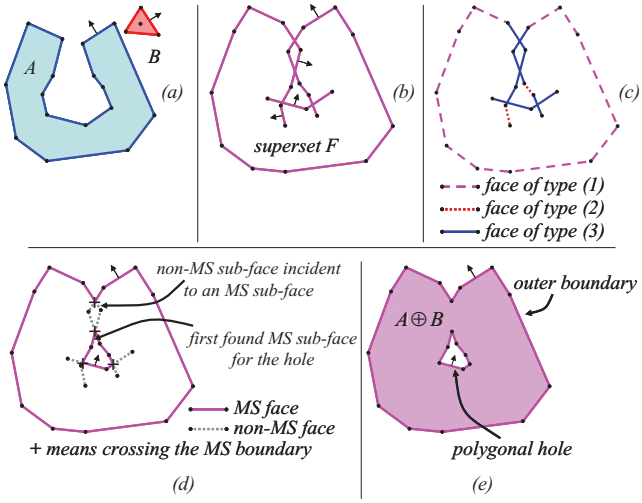

Fig. 11. Illustration of a polygonal hole extraction from a Minkowski sum facets superset in 2D. (a) A nonconvex–convex pair of polygons A and B; (b) their MS faces superset; (c) the different categories of faces of the superset; (d) the crossing of the MS frontier implies to change the membership state at the two sides of the frontier; (e) the MS $A \oplus B$ composed of an outer boundary and a polygonal hole that was successfully extracted.

3.4 Extraction of Polyhedral Holes Inside the Minkowski Sum Outer Boundary

The extraction of the outer boundary of the MS and the handling of nonmanifold situations are not sufficient for some input polyhedra. Consider the example depicted in Figure 10, where the MS polygon is composed of an outer boundary and a polygonal hole which must be recovered (Figure 10(e)). The application of Algorithm 1 on the superset of Figure 10(c) only extracts the outer boundary. The traversal of the MS outer boundary excludes edges lying inside it as shown in Figure 10(e) and the polygonal hole is therefore missed.

In this section, we address the extraction of eventual polyhedral holes inside the MS mesh, which is an additional functionality of our NCC-CVMS algorithm that must be applied after the extraction of the outer boundary, that is, after performing all steps of Algorithm 1. The key idea is to propagate membership states through the neighboring subfacets and to change them for the subfacets encountered after crossing the MS boundary. As detailed in the next paragraphs, the extraction of polyhedral holes relies on the two following properties: (1) the uniqueness of the membership state inside each subfacet and (2) the change of the membership state at the two sides of the crossed MS boundary.

We already demonstrated in Proposition 1 that each subfacet of the superset has a unique membership state w.r.t. the MS mesh. Let us now examine the triangles of the superset. On the one hand, for a triangle lying partially on the boundary of the MS mesh, the edges of the frontier separating its MS subfacets from its non-MS subfacets result from crossing the MS boundary. On the other hand, it is not possible to have a subdivision of a triangle lying completely on the boundary of the MS mesh or completely inside the volume defined by this mesh. This is because such a subdivision implies that some of its subfacets are MS-facets while the others are non-MS subfacets, which is contradictory since the whole triangle has a unique membership state. This allows the classification of the superset triangles into three categories as illustrated in 2D by Figure 11(c).

(1) *Triangles lying completely on the boundary of the MS mesh.* These are nonsubdivided triangles since there is no change of the membership state inside them.

(2) *Triangles lying completely inside the volume defined by the MS mesh.* These triangles are also nonsubdivided ones for the same reason as before.

(3) *Triangles lying partially on the boundary of the MS mesh and partially inside it.* These are triangles subdivided into subfacets, some of which lie completely on the boundary of the MS mesh while the others lie inside the volume defined by this mesh.

This equivalence between the crossing of the boundary of the MS mesh and the change of membership state of the subfacets at the two sides of the frontier, together with the impossibility of having a subdivision of a triangle with a unique membership state (categories (1) and (2) of triangles) gives us a mean to compute eventual polyhedral holes inside the MS outer boundary, by examining the intersections between the remaining triangles and changing the membership state of the subfacets of the subdivided triangles. Figure 11 illustrates the extraction of a polygonal hole from a Minkowski sum facets superset in 2D.

Therefore, after computing the outer boundary of the Minkowski sum mesh, the extraction of the eventual polyhedral holes of the MS mesh reduces to the following.

(1) Find a bounded non-MS subfacet of a triangle subdivision that is incident to an MS subfacet of the same triangle's subdivision. In Figure 11(d), such a bounded non-MS subface is shown.

(2) Traverse nonvisited neighboring triangles until arriving to a subdivided triangle which necessarily crosses the MS mesh boundary. The crossing of the MS boundary is located by the + sign in Figure 11(d).

(3) Set the bounded subfacet at which the triangle is entered as a non-MS subfacet and its adjacent subfacets lying on the subdivision of this triangle as MS subfacets of a polyhedral hole

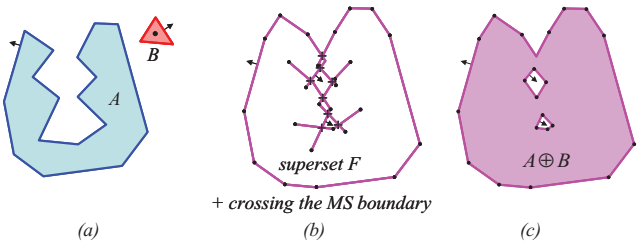

Fig. 12. Extraction of two polygonal holes from a Minkowski sum facets superset in 2D. (a) A nonconvex–convex pair of polygons A and B; (b) their MS faces superset. The + sign indicates where the MS boundary is crossed and where the membership state must be changed; (c) the MS polygon containing two holes.

because they are encountered after crossing the MS boundary. In Figure 11(d), the first MS subface found after crossing the MS boundary is indicated.

(4) Consider the recently found MS subfacet as a seed subfacet and use steps ranging from 8 to 18 of Algorithm 1 to recover the polyhedral hole. The faces drawn as solid lines in Figure 11(d) that are located inside the MS outer boundary are faces of a polygonal hole.

(5) Repeat steps ranging from (1) to (4) until all the subfacets of the superset F are visited.

The previous procedure is able to extract any number of polyhedral holes as illustrated in Figure 12. Some examples of nonmanifold MS meshes and another one containing a polyhedral hole computed by our algorithm are depicted in Figures 15 and 16 of Section 4.

4. IMPLEMENTATION, PERFORMANCE BENCHMARK, AND COMPARATIVE STUDY

In this section, we give some implementation details, provide a comparison with other approaches, and present some results.

4.1 Implementation

The NCC-CVMS algorithm has been implemented using C++ and CGAL [CGA], the Computational Geometry Algorithms Library. The polyhedra are handled by the CGAL `Polyhedron_3` data structure, which is in turn based on the halfedge data structure [Kettner 1999]. The computation of the planar Minkowski sums involved in the NCC-CVMS algorithm is done by the function `minkowski_sum_2` provided in the 2D Minkowski sum package of CGAL. This function implements the convolution operation [Guibas et al. 1983].

As stated previously, we followed the exact computation paradigm to guarantee the exactness of the obtained results. For our implementation, we used the exact number types provided by the GNU Multi Precision (GMP) library [GMP] under CGAL. By using exact number types, we penalize running time performance w.r.t. to the built-in floating point number types. To overcome this problem, we used the lazy kernel adapter [Fabri and Pion 2006; CGA] which speeds up exact computations and reduces the overhead in comparison with the floating-point-based computations.

The merging of "coplanar conjoint" superset facets is done by first projecting the 3D polygonal facets to obtain the corresponding 2D polygons (see Section 3.1), computing the union of the resulting 2D polygons through the use of the 2D regularized Boolean set operations package [Fogel et al. 2008] of CGAL, and projecting back the union result on the supporting plane of the merged facets (see again Section 3.1). The computation of the symmetric difference of "opposite conjoint" superset facets is done in a similar manner.

The triangulation of the superset facets is also done in 2D through the 3D-to-2D and 2D-to-3D projections mentioned earlier. If the facet to be triangulated is a convex facet without holes, the triangulation is performed by simple Euler operations. If the facet is a nonconvex facet without holes, we first convex-partition it into convex polygons using the algorithm presented in Hertel and Mehlhorn [1983] and then we use the Euler operations to triangulate the convex polygons. If the facet contains holes, we use the constrained Delaunay triangulation algorithm of Yvinec [2008]. The constrained edges are the edges incident to the holes and the edges of the outer boundary of the facet.

The Iso-oriented bounding boxes intersection is computed by the fast algorithm presented in Zomorodian and Edelsbrunner [2002].

The 3D triangle intersection computation is achieved by a slightly modified version of the "interval overlap method" presented in Möller [1997].

The 2D arrangements that store the triangles' intersections are handled by the 2D arrangement package of CGAL [Wein et al. 2008].

We shall note the use of the Nef polyhedra implementation [Hachenberger and Kettner 2008] provided by CGAL for the comparison of the running times in Section 4.2.2.

4.2 Performance Benchmark

For our benchmark, we carried out three series of tests. The first one aims at measuring the amount of time consumed for each step of the NCC-CVMS algorithm w.r.t. the overall running time of the algorithm. In the second series, we compared the NCC-CVMS algorithm with the Nef polyhedra-based algorithm which is to the best of our knowledge the unique one handling Minkowski sum of nonconvex polyhedra in an exact way. Another reason for choosing this approach for comparison is the availability of its source-code. The last series of tests concerns nonconvex–convex pairs of polyhedra without fold and aims at comparing our NCC-CVMS algorithm with the CVMS algorithm we presented in Barki et al. [2009b]. All experiments were done on a 4GB RAM, 2.2 GHZ Intel Core 2 Duo personal computer.

4.2.1 *Comparison of the Running Times of the Different Steps of the NCC-CVMS Algorithm.* In the current tests, we will deliberately choose a geodesic sphere and another convex polyhedron to be summed with each nonconvex polyhedron A. The choice of the sphere is motivated by its complexity since it is composed of hundreds of facets with hundreds of different outer normals and thus constitutes a challenge for the superset generation step and for the Minkowski sum polyhedron extraction step. More results of Minkowski sum of several other polyhedra can be downloaded from our Web page at: http://liris.cnrs.fr/hichem.barki/mksum/NCC-CVMS.

The running times for the NCC-CVMS algorithm and the ratio for each step are reported in Table I. The numbers of triangles of A, B, the MS superset, and the MS polyhedron are also given.

From Table I, the MS superset triangles' intersection computation is in average the most time-consuming step of the NCC-CVMS. Another interesting observation is that, for some samples, the merging, symmetric difference computation, and triangulation process consumes more time than the triangles' intersection computation. This is justified by the presence of a large number of "coplanar conjoint" and/or "opposite conjoint" facets in the generated superset.

Table I. Percentage of Running Times for the Different Steps of the NCC-CVMS Algorithm for Several Samples

Operands (triangles number)		Percentage of Running Times (%)									
A	B	SC	MST	BI	TI	AC	BT	TR	Overall time(s)	Sup. size	MS size
Wrench (772)	Rhomb.Tr.Icosa.(62)	10.2	14.3	0.2	41.3	20.7	12.1	1.3	**53.219**	7056	4041
Wrench (772)	Sphere (500)	15.5	31.3	0.1	28.3	15.4	8.6	0.9	**97.532**	9648	5205
Grate1 (540)	Tr.Tetra. (8)	6.5	5.8	0.1	48.7	19.2	16.9	2.9	**15.171**	1634	1915
Grate1 (540)	Sphere (500)	11.2	35.5	0.1	31.4	12.7	8.3	0.7	**107.968**	10311	5524
Bunny (1500)	Cube (12)	5.7	14.5	0.1	36.9	16.5	24	2.4	**21.720**	2606	3386
Bunny (1500)	Sphere (500)	24.9	24.3	0.1	22.5	11.1	15.3	1.7	**136.421**	10217	13281
Knot (992)	Rh.Triacon. (30)	8	12.6	0.1	37.1	17.9	20.7	3.6	**41.656**	4426	6481
Knot (992)	Sphere (500)	13.2	30.7	0.1	26.1	13.1	14.4	2.4	**172.047**	13521	17759
Grate2 (942)	Rh.Dodeca. (12)	5.2	8.2	0.1	40.5	15.5	26.3	4.2	**21.031**	2690	4619
Grate2 (942)	Sphere (500)	8.4	48.3	0.1	18.6	7.8	15.7	1	**280.063**	18663	23760
Dinausor (5000)	Cube (12)	5.3	31.6	0.1	29.9	15	16.1	2	**107.546**	9582	11422
Dinausor (5000)	Sphere (500)	14	58.2	0.1	15.2	7.9	4.1	0.6	**1010.842**	42495	29265
Average running time percentage		**10.7**	**26.3**	**0.1**	**31.4**	**14.4**	**15.2**	**2.0**	**-**	-	-

Rhomb.Tr.Icosa. - RhombiTruncatedIcosahedron, Tr.Tetra. - TruncatedTetrahedron, Rh.Triacon. - RhombicTriacontahedron, Rh.Dodeca. - RhombicDodecahedron
SC - Superset Computation , MST - Merging of "coplanar conjoint" superset facets, Symmetric difference computation of the "opposite conjoint" facets, and Triangulation, BI - Iso-oriented Boxes Intersection computation, TI - superset Triangles' Intersection computation, AC - 2D Arrangements Construction, BT - computation of the seed facet and Boundary Traversal, TR - optional TRiangulation of the MS polyhedron facets, Sup. size - number of triangles of the MS superset after the merging, symmetric difference computation, and triangulation steps, MS size - number of triangles of the MS polyhedron.
Some of the reported models are taken from http://masc.cs.gmu.edu/wiki/SimpleMsum.

The NCC-CVMS algorithm works well with big size polyhedra, having until tens of thousands of facets. As an example, for the Dinausor model to be summed with the sphere model, the NCC-CVMS algorithm handled correctly the superset of 42K triangles and computed the Minkowski sum in less than 17 minutes. For the models reported in Table I, the NCC-CVMS algorithm computed about 6.2K triangles per minute in average. This measure gives a first vision about the efficiency of our algorithm.

The NCC-CVMS algorithm handles well the change of genus. As an example, the Wrench model in Table I has a genus of 4 while the Minkowski sum polyhedra Wrench $\oplus$ Sphere and Wrench $\oplus$ RhombitruncatedIcosahedron have a genus of 0 (see Figure 14(d)). Models with higher genus are also well handled. The models Knot and Knot $\oplus$ Sphere have a genus of 9 (see Figure 14(a)).

For some models such as the Grate2 model, the size of the MS polyhedron is greater than the size of the MS superset. At first glance, this might seem contradictory since the MS polyhedron is the result of the trimming of the MS superset (some subfacets of the superset will not be present in the MS polyhedron). However, this is natural because in some cases, the triangulation of the MS subfacets retained after the trimming operation yields a lot of triangles due to the geometry of these facets.

4.2.2 *Comparison between the NCC-CVMS Algorithm and the Nef Polyhedra-Based Minkowski Sum Approach.* For some of the models reported in Table I, we computed the Minkowski sum by using our NCC-CVMS algorithm and the Nef polyhedra-based convex decomposition method implemented within CGAL. The running times are reported in Table II. We avoided deliberately to report the sizes of the polyhedra A and B which are the same as in Table I.

We shall note that Nef polyhedra-based convex decomposition approach is more powerful than necessary for the Minkowski sum computation. The Nef polyhedra data structure is complex and not restricted to the Minkowski sum computation. It is also used for other needs such as Boolean operations on polyhedra [Hachenberger et al. 2007]. Moreover, the computation of the pairwise Minkowski sums performed by overlaying two Nef polyhedra embedded on the sphere, with lower-dimensional features, unbounded or bounded boundaries, etc., is not optimized for the Minkowski sum.

Table II. Comparison of the Running Times for the Nef Polyhedra-Based Approach and the NCC-CVMS Algorithm

Operands		Nef Polyhedra		NCC-CVMS
A	B	Conv. dec. A	Nef (s)	NCC-CVMS (s)
Grate1	Tr.Tetra.	47	**20.547**	**15.171**
Grate1	Sphere	47	**272.406**	**107.968**
Bunny	Cube	869	**733.266**	**21.720**
Knot	Rh.Triacon.	690	**4262.800**	**41.656**
Grate2	Rh.Dodeca.	230	**114.125**	**21.031**
Grate2	Sphere	230	**2052.890**	**280.063**
Dinausor	Cube	3193	**2842.480**	**107.546**

Rhomb.Tr.Icosa. - RhombiTruncatedIcosahedron, Tr.Tetra. - TruncatedTetrahedron, Rh.Triacon. - RhombicTriacontahedron, Rh.Dodeca. - RhombicDodecahedron
Conv. decomp. of A - The number of convex pieces resulting from the decomposition of the non-convex polyhedron A.

From the running times reported in Table II, it appears that our NCC-CVMS algorithm is more efficient than the Nef polyhedra-based approach. Moreover, our algorithm is even much more efficient than the Nef polyhedra for models yielding more convex parts, such as the the Bunny and the Dinausor models. However, as claimed previously, the Nef polyhedra-based approach deals with nonconvex pairs of polyhedra that can not been handled by our NCC-CVMS algorithm at the moment.

4.2.3 *Comparison between the NCC-CVMS Algorithm and the CVMS Algorithm for Nonconvex–Convex Pairs of Polyhedra without Fold.* In this section, we computed the Minkowski sum of some nonconvex–convex pairs of polyhedra without fold by using our NCC-CVMS algorithm and also by using the CVMS algorithm we previously proposed in Barki et al. [2009b]. The running times are reported in Table III along with the percentages of the superset computation step.

Besides the fact that the NCC-CVMS algorithm handles polyhedra having fold and that the CVMS algorithm cannot handle them, Table III shows that NCC-CVMS performs better than CVMS for nonconvex–convex pairs of polyhedra without fold. The envelope computation in CVMS induces some redundancy, that is, a large number of the MS facets that are computed more than once before

Table III. Comparison of the Running Times for the CVMS Algorithm of Barki et al. [2009b] and Our NCC-CVMS Algorithm

Operands		CVMS Algorithm			NCC-CVMS
A	B	Sup. comp.	Env. comp.	CVMS(s)	NCC-CVMS(s)
L(20)	Tetra.(4)	15.9%	84.1%	**0.390**	**0.266**
L(20)	Sphere(500)	3.3%	96.7%	**16.046**	**5.045**
Path(28)	Sphere(500)	3.3%	96.7%	**20.218**	**6.313**
Star(24)	Sphere(500)	1.5%	98.5%	**42.235**	**9.673**
Average		**2.7%**	**97.3%**	-	-

Tetra. - Tetrahedron.

Sup. comp. - Percentage of the running time of the superset computation step in the CVMS algorithm of [Barki et al. 2009b], Env. comp. - Percentage of the running time of the envelope computation step in the CVMS algorithm of [Barki et al. 2009b].

the union. In contrast, NCC-CVMS deals with each MS facet only once.

The envelope computation in CVMS is the most time-consuming step (97.3% in average). The superset computation step consumes in average 2.7% of the overall time. In NCC-CVMS, the superset computation consumes in average 10.7% (see Table I). The increase of the superset computation ratio in NCC-CVMS is due to the reduction of the running time devoted to the other steps of NCC-CVMS compared to the envelope computation step of the CVMS algorithm.

4.3 Comparative Study with Other Approaches

In this subsection, we give a comparative study of our NCC-CVMS algorithm and two other approaches handling the Minkowski sum of nonconvex polyhedra in an efficient and nearly exact way. The first approach is the topologically accurate approximation of Varadhan and Manocha [2006] and the second one is the nearly exact method of Lien [2008].

Varadhan and Manocha used a convex-decomposition-based approach and approximated the union of the pairwise Minkowski sums by generating a voxel grid, computing signed distance on the grid points, and performing Iso-surface extraction from the distance field.

Lien first computed a superset of the Minkowski sum through a brute-force convolution of the two input polyhedra. Then, he subdivided each facet of this superset into subfacets induced by the facet-facet intersections. After that, he stitched the subfacets into simple regions which are either entirely on the boundary of the Minkowski sum or entirely inside it. Finally, he used a collision detector to decide about the regions lying on the boundary of the Minkowski sum. However, the used collision detector cannot distinguish between two objects which are in contact configurations or which collide. To overcome this problem, the author perturbed each sampled point with an infinitesimally position vector pointing in the direction of the concerned subfacet outer normal. For that reason, the method is said "nearly exact."

We compare our NCC-CVMS algorithm to the mentioned approaches on the base of the following criteria: class of handled input polyhedra, class of handled output, exactness, efficiency, and implementation.

—*Class of handled input polyhedra.* The method of Lien and the approach of Varadhan and Manocha both handle nonconvex polyhedra. Our NCC-CVMS algorithm only handles nonconvex–convex pairs. To be able to handle nonconvex pairs of polyhedra, the superset generation step of our NCC-CVMS algorithm must be updated.

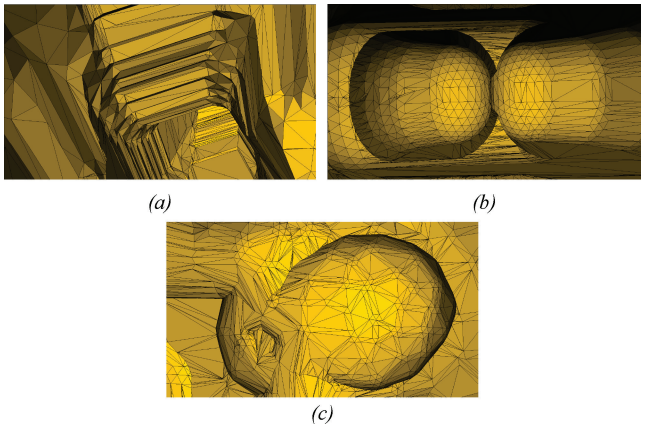

Fig. 13. Internal views for Minkowski sum polyhedra computed by the NCC-CVMS algorithm. (a) For the MS of the Dinausor model and the Cube model; (b) for the MS of the Wrench model and the Sphere model; (c) for the MS of the Bunny model and the Sphere model.

—*Class of handled output.* Concerning the method of Lien, there is no claim about the handling of nonmanifold situations. Moreover, there is no handling of "coplanar conjoint" facets or "opposite conjoint" facets. If some subfacets which are either "coplanar conjoint" or "opposite conjoint" belong to the boundary of the Minkowski sum, they satisfy the collision detector, create non-handled nonmanifold situations, and induce erroneous results. The approach of Varadhan and Manocha does not handle nonmanifold output because the sampling algorithm used to generate the voxel grid does not handle primitives touching tangentially. NCC-CVMS handles nonmanifold output and is able to recover eventual polyhedral holes of the Minkowski sum (see the next subsection for examples of these situations).

—*Exactness.* The method of Lien gives an exact solution in most cases. However, it does not guarantee the exactness of the result for all input models because the perturbation mechanism used in the collision detector for the distinction between non-MS subfacets and MS subfacets is empirical. The approach of Varadhan and Manocha gives an accurate approximation of the result. NCC-CVMS gives an exact result.

—*Efficiency*. Lien's method is the more efficient among the three due to the use of built-in number types, the parallelized implementation, and because it is not based on convex decomposition. The approach of Varadhan and Manocha suffers from the large size of the convex decomposition. Moreover, the decomposition algorithm is not optimal and the convex hull method used for the computation of the pairwise Minkowski sums is not the most efficient for the task at hand (see Barki et al. [2009a] for a discussion of the approaches used for the computation of the Minkowski sum of convex polyhedra). Our algorithm uses exact number types, it is more efficient than the approach of Varadhan and Manocha since it processes about 6.2K triangles per minute in average while the approach of Varadhan and Manocha takes few minutes to compute the Minkowski sum of models composed of hundreds of triangles.

—*Implementation.* The approach of Lien and that of Varadhan and Manocha are both implemented using built-in number types and suffer from round-off errors. NCC-CVMS is implemented using exact number types and does not suffer from such issues.

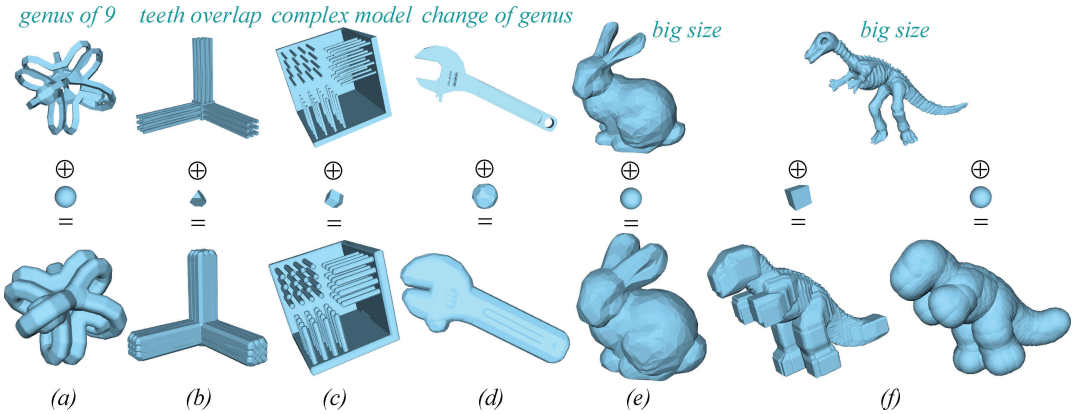

Fig. 14. Minkowski sum polyhedra generated by our NCC-CVMS algorithm. From top to bottom of each subfigure: the polyhedron A, the polyhedron B, and the Minkowski sum polyhedron $S = A \oplus B$. (a) The MS of the Knot model (of high genus) and the Sphere model; (b) the MS of the Grate1 model and the TruncatedTetrahedron model. The teeth overlap is robustly handled; (c) the MS of the Grate2 complex model and the RhombicDodecahedron model; (d) the MS of the Wrench model and the RhombiTruncatedIcosahedron model. The change of genus is correctly handled; (e) the MS of the Bunny model and the Sphere model; (f) the MS of the Dinausor model and the Cube/Sphere models. The NCC-CVMS algorithm works well on big size polyhedra.

4.4 Examples

In this subsection, we present two sets of examples of Minkowski sum meshes we computed with NCC-CVMS. The first one concerns results that are polyhedra while the second gives examples of Minkowski sums exhibiting nonmanifold situations and polyhedral holes.

For the first set of examples, we present two views for the Minkowski sum polyhedron from a view point that is:

—inside the MS polyhedron (Figure 13). This view aims at showing that the trimming is done correctly during the traversal step and that no non-MS subfacet is present in the MS polyhedron.

—outside the MS polyhedron (Figure 14). This view aims at showing that no subfacet is missed during the traversal step and that there is not any hole in the boundary of the MS polyhedron.

In Figure 14, some models reported in Table I are shown from outside view points. The overlap of the rods in the Grate1 model does not cause any problem to our algorithm (Figure 14(b)).

For the second set of examples, we manually generated some models whose Minkowski sums either are nonmanifold meshes or contain a polyhedral hole. Figures 15(a), 15(b), and 15(c) illustrate nonmanifold Minkowski sums computed by NCC-CVMS in the presence of double vertices, a nonmanifold edge, and two facets touching tangentially. Figure 16 shows a Minkowski sum with a polyhedral hole that was successfully extracted.

We shall note here that nonmanifold Minkowski sums and Minkowski sums with polyhedral holes are uncommon in practice. For this reason, we manually generated the input models illustrated in Figures 15 and 16 in order to produce such meshes. For the models depicted in Figure 16, NCC-CVMS computed the whole Minkowski sum (842 triangles) in 8.328 seconds, among which 15.4% was spent in the computation of the polyhedral hole.

We also validated the correctness of our results by comparing the distance between the Minkowski sum polyhedra computed by our NCC-CVMS algorithm and those computed by the Nef polyhedra-based approach through the use of the Metro software which can be found at: http://vcg.isti.cnr.it/activities/surfacegrevis/simplification/metro.html.

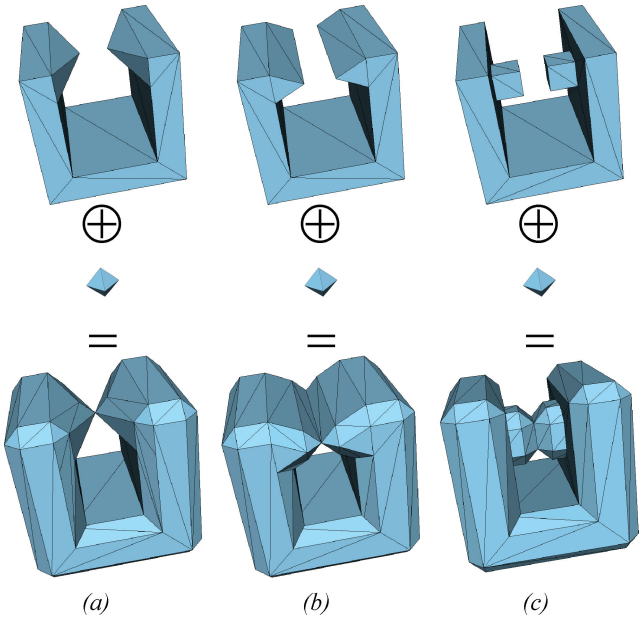

Fig. 15. Non-manifold Minkowski sums computed by the NCC-CVMS algorithm. (a) Double vertices. (b) A non-manifold edge. (c) Facets touching tangentially.

More results of the NCC-CVMS algorithm and some animations showing the traversal of the boundary and the incremental construction of some MS polyhedra can be found in our Web page at: http://liris.cnrs.fr/hichem.barki/mksum/NCC-CVMS.

5. CONCLUSION AND FUTURE WORK

We have presented the NCC-CVMS algorithm, an original contributing vertices-based algorithm for the computation of the Minkowski sum of a nonconvex–convex pair of polyhedra A and B which handles nonmanifold output and extracts eventual polyhedral holes of the Minkowski sum result. First, we computed a superset

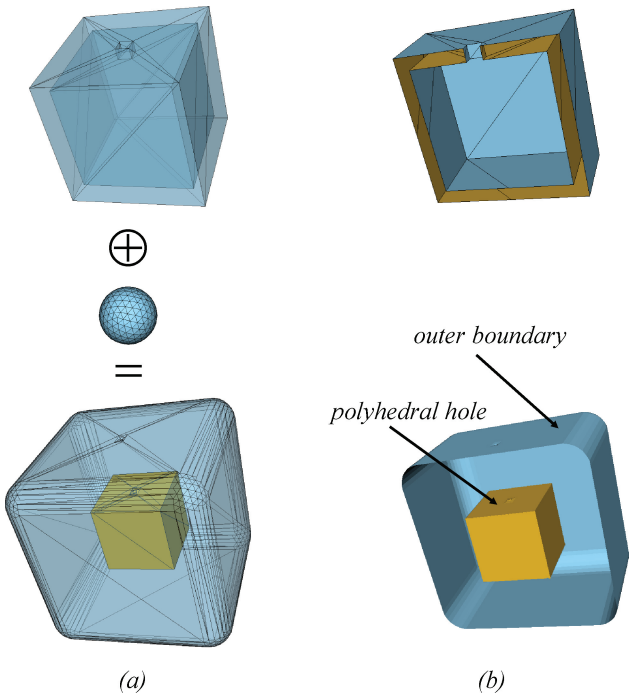

Fig. 16. A Minkowski sum with a polyhedral hole computed by the NCC-CVMS algorithm. (a) The two operands and the whole MS mesh; (b) two cuts better showing the geometry of the nonconvex operand and the polyhedral hole inside the MS outer boundary.

of the Minkowski sum facets of A and B. Second, we computed the intersections among the superset triangles and maintained them by 2D arrangements. Finally, starting from a seed facet that is guaranteed to belong to the boundary of the Minkowski sum, we examined its neighborhood and determined the Minkowski sum facets satisfying the closeness and two-manifoldness properties. This process was repeated until the closeness property of the Minkowski sum polyhedron was satisfied, that is, until the resulting polyhedron was closed.

We followed the exact computation paradigm in the implementation of the NCC-CVMS algorithm. Degenerate cases were well handled and robustness issues related to built-in number types were avoided. Moreover, nonmanifold Minkowski sum meshes were also well handled by our algorithm. Experimental results showed that our NCC-CVMS algorithm outperforms the Nef polyhedra-based convex decomposition method of Hachenberger [2007] which also handles nonconvex polyhedra in an exact fashion. We gave a comparative study of our approach and two other approaches computing Minkowski sum of nonconvex polyhedra but in an inexact way: the nearly exact approach of Lien [2008] and the topologically accurate approximation of Varadhan and Manocha [2006].

The NCC-CVMS algorithm is more general and more efficient than the algorithm we previously proposed in Barki et al. [2009b] for a nonconvex–convex pair of polyhedra without fold.

As a part of our future work, we are working on the generalization of our algorithm to any pair of nonconvex polyhedra. For that purpose, the superset generation step has to be updated accordingly. The Minkowski sum boundary extraction algorithm presented here will then be applied without significant modifications since the closeness and two-manifoldness properties are not altered by the nonconvexity of polyhedra.

REFERENCES

CGAL. Computational geometry algorithms library. http://www.cgal.org.

GNU MP. the GNU MP bignum library. http://gmplib.org.

AGARWAL, P. AND SHARIR, M. 1998. Arrangements and their applications. In *Handbook of Computational Geometry*, Elsevier Science Publishers B.V. North-Holland, 49–119.

ARONOV, B., SHARIR, M., AND TAGANSKY, B. 1997. The union of convex polyhedra in three dimensions. *SIAM J. Comput. 26,* 6, 1670–1688.

BARKI, H., DENIS, F., AND DUPONT, F. 2009a. Contributing vertices-based minkowski sum computation of convex polyhedra. *Comput. Aid. Des. 41,* 7, 525–538.

BARKI, H., DENIS, F., AND DUPONT, F. 2009b. Contributing vertices-based minkowski sum of a non-convex polyhedron without fold and a convex polyhedron. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications (SMI'09)*. IEEE Computer Society, 73–80.

BASCH, J., GUIBAS, L., RAMKUMAR, G., AND RAMSHAW, L. 1996. Polyhedral tracings and their convolution. In *Proceedings of 2nd Workshop on the Algorithmic Foundations of Robotics*. 171–184.

BEKKER, H. AND ROERDINK, J. 2001. An efficient algorithm to calculate the Minkowski sum of convex 3d polyhedra. In *Proceedings of the International Conference on Computational Sciences-Part I (ICCS'01)*. Springer, 619–628.

CHAZELLE, B. 1981. Convex decompositions of polyhedra. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC'81)*. ACM, New York, 70–79.

EVANS, R., OÒCONNOR, M., AND ROSSIGNAC, J. 1992. Construction of Minkowski sums and derivatives morphological combinations of arbitrary polyhedra in CAD/CAM systems. US Patent 5159512.

FABRI, A. AND PION, S. 2006. A generic lazy evaluation scheme for exact geometric computations. In *Proceedings of the 2nd Library-Centric Software Design*.

FOGEL, E. AND HALPERIN, D. 2007. Exact and efficient construction of Minkowski sums of convex polyhedra with applications. *Comput. Aid. Des. 39,* 11, 929–940.

FOGEL, E., WEIN, R., ZUKERMAN, B., AND HALPERIN, D. 2008. 2d regularized boolean set-operations. In *CGAL User and Reference Manual*, 3.4 Ed., C. E. Board, Ed.

GHOSH, P. 1993. A unified computational framework for Minkowski operations. *Comput. Graph. 17,* 4, 357–378.

GUIBAS, L. AND SEIDEL, R. 1987. Computing convolutions by reciprocal search. *Discrete Comput. Geom. 2*, 175–193.

GUIBAS, L. J., RAMSHAW, L., AND STOLFI, L. 1983. A kinetic framework for computational geometry. In *Proceedings of 24th annual IEEE Symposium on the Foundation of Computer Science*. 100–111.

HACHENBERGER, P. 2007. Exact Minkowski sums of polyhedra and exact and efficient decomposition of polyhedra in convex pieces. In *Proceedings 15th Annual European Symposium on Algorithms*. 669–680.

HACHENBERGER, P. AND KETTNER, L. 2008. 3d boolean operations on Nef polyhedra. In *CGAL User and Reference Manual*. 3.4 Ed., C. E. Board, Ed.

HACHENBERGER, P., KETTNER, L., AND MEHLHORN, K. 2007. Boolean operations on 3d selective nef complexes: data structure, algorithms, optimized implementation and experiments. *Comput. Geom. Theory Appl. 38,* 1-2, 64–99.

HALPERIN, D. 2002. Robust geometric computing in motion. *Int. J. Robotics Res. 21,* 3, 219–232.

HERTEL, S. AND MEHLHORN, K. 1983. Fast triangulation of simple polygons. In *Proceedings of the International FCT-Conference on Fundamentals of Computation Theory*. Springer, 207–218.

KAUL, A. AND ROSSIGNAC, J. 1992. Solid-interpolating deformations: construction and animation of PIPs. *Comput. Graph. 16,* 1, 107–115.

KETTNER, L. 1999. Using generic programming for designing a data structure for polyhedral surfaces. *Comput. Geom. Theory Appl. 13,* 1, 65–90.

KIM, Y., OTADUY, M., LIN, M., AND MANOCHA, D. 2003. Fast penetration depth estimation using rasterization hardware and hierarchical refinement. In *Proceedings of the 19th Annual Symposium on Computational Geometry* (*SCG'03*). ACM, New York, 386–387.

LEE, I.-K., KIM, M.-S., AND ELBER, G. 1998. Polynomial/rational approximation of Minkowski sum boundary curves. *Graph. Models Image Process. 60,* 2, 136–165.

LIEN, J.-M. 2007. Point-based Minkowski sum boundary. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications (PG'07)*. IEEE Computer Society, 261–270.

LIEN, J.-M. 2008. A simple method for computing Minkowski sum boundary in 3d using collision detection. In *Proceedings of 8th Workshop on the Algorithmic Foundations of Robotics.*

LOZANO-PÉREZ, T. 1983. Spatial planning: A configuration space approach. *IEEE Trans. Comput. 32,* 2, 108–120.

MÖLLER, T. 1997. A fast triangle-triangle intersection test. *J. Graph. Tools 2,* 2, 25–30.

SERRA, J. 1982. *Image Analysis and Mathematical Morphology*, vol. 1. Academic Press, London.

SERRA, J. 1988. *Image Analysis and Mathematical Morphology*, vol. 2. Academic Press, New York.

TUZIKOV, A., ROERDINK, J., AND HEIJMANS, H. 2000. Similarity measures for convex polyhedra based on Minkowski addition. *Patt. Recogn. 33*, 979–995.

VARADHAN, G. AND MANOCHA, D. 2006. Accurate Minkowski sum approximation of polyhedral models. *Graph. Models 68,* 4, 343–355.

WEIBEL, C. Minkowski sums. http://roso.epfl.ch/cw/poly/public.php.

WEIN, R., FOGEL, E., ZUKERMAN, B., AND HALPERIN, D. 2008. 2d arrangements. In *CGAL User and Reference Manual* 3.4 ed., C. E. Board, Ed. http://www.cgal.org/Manual/lotest/doc.html/cgal_manual/contents.html

WU, Y., SHAH, J., AND DAVIDSON, J. 2003. Improvements to algorithms for computing the Minkowski sum of 3-polytopes. *Comput. Aid. Des. 35,* 13, 1181–1192.

YVINEC, M. 2008. 2d triangulations. In *CGAL User and Reference Manual* 3.4 Ed., C. E. Board, Ed. http://www.cgal.org/Manual/lotest/doc.html/cgal_manual/contents.html

ZOMORODIAN, A. AND EDELSBRUNNER, H. 2002. Fast software for box intersections. *Int. J. Comput. Geom. Appl. 12,* 1-2, 143–172.

Received June 2009; revised August 2010; accepted October 2010