Algorithmica (2009) 55: 329–345
DOI 10.1007/s00453-008-9219-6

Exact Minkowksi Sums of Polyhedra and Exact and Efficient Decomposition of Polyhedra into Convex Pieces

Peter Hachenberger

Received: 13 November 2007 / Accepted: 18 July 2008 / Published online: 15 August 2008

Abstract We present the first exact and robust implementation of the 3D Minkowski sum of two non-convex polyhedra. Our implementation decomposes the two polyhedra into convex pieces, performs pairwise Minkowski sums on the convex pieces, and constructs their union. We achieve exactness and the handling of all degeneracies by building upon 3D Nef polyhedra as provided by CGAL. The implementation also supports open and closed polyhedra. This allows the handling of degenerate scenarios like the tight passage problem in robot motion planning.

The bottleneck of our approach is the union step. We address efficiency by optimizing this step by two means: we implement an efficient decomposition that yields a small number of convex pieces, and develop, test and optimize multiple strategies for uniting the partial sums by consecutive binary union operations.

The decomposition that we implemented as part of the Minkowski sum is interesting in its own right. It is the first robust implementation of a decomposition of polyhedra into convex pieces that yields at most $O(r^2)$ pieces, where r is the number of edges whose adjacent facets comprise an angle of more than 180 degrees with respect to the interior of the polyhedron.

Keywords Minkowski sum · Decomposition of polyhedra into convex pieces · Nef polyhedra · Tight passage · Exact arithmetic

This work was partially supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

P. Hachenberger (✉)
Department of Computing Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
e-mail: phachenb@win.tue.nl

1 Introduction

The Minkowski sum of two point sets P and Q in $\mathbb{R}^d$, denoted by $P \oplus Q$, is defined as the set $\{p + q : p \in P, q \in Q\}$. Minkowski sums are used in a wide range of applications such as robot motion planning [19], computer-aided design and manufacturing [11], penetration depth computation [18, 20], offset computation [23], morphing [17], and mathematical morphological operations [24].

In several applications (e.g. in GIS or imaging) one deals with Minkowski sums of two-dimensional objects, and several implementations exist. When the two objects are non-convex polygons, two approaches are commonly used. The first approach computes the convolution of the boundary of two polygons [14]. The other approach decomposes both polygons into convex pieces, computes the pairwise Minkowski sums of the pieces, and merges the pairwise sums. Both approaches have also been studied and implemented in combination with exact geometric computation [1, 27]. The *Library for Efficient Data Types and Algorithms* (LEDA)[1] offers an exact implementation based upon the first method. The *Computational Geometry Algorithm Library* (CGAL)[2] offers exact implementations of both methods.

There are also many applications, however, that require the computation of the Minkowski sums of three-dimensional objects. Examples can be found in CAD/CAM, assembly planning, and motion planning. Implementations of the 3D Minkowski sum exist, but they are not robust. The most efficient such implementation is probably by Varadhan and Manocha [26]. It is based upon the convex decomposition approach as described above for the two-dimensional case. The Minkowski sum is approximated by generating a signed distance field. If needed, an isosurface can be extracted from the field. They guarantee the correct topology of the result, but are limited to manifold boundaries. Although many input objects are commonly two-manifolds, this limitation seems to be a major robustness issue, because the primitives of the union step, i.e., the Minkowski sum of two convex pieces, are not allowed to touch tangentially. Evans et al. [12] presented a third method, which is similar to the approach that decomposes the polyhedra into convex pieces. They compute the Minkowski sum by decomposing the boundaries of the two polyhedra into affine cells, computing pairwise Minkowski sums between pairs of transversal affine cells, and computing their union with translated versions of the original polyhedra. In comparison to the decomposition into convex cells, the decomposition into affine cells produces more cells. Evans et al. implemented their method based on IBM's polyhedron modeler Geometric Design Processor (GDP) [28], which is not marketed any more. When the Minkowski sum was implemented, GDP did not support exact arithmetic, and it is unclear whether the implementation was ever tested together with exact arithmetic. Experimental results are not available [22]. At the moment there is no implementation available that is robust (that is, can deal with all possible degenerate cases), nor is there an implementation that is exact. This is the goal of our work: to provide a solution for the computation of Minkowski sums of 3D polyhedra that can handle all degenerate cases and is exact.

[1] http://www.algorithmic-solutions.com.

[2] http://www.cgal.org.

We present the first exact implementation of the Minkowski sum of two non-convex polyhedra. Our implementation is based on the convex decomposition approach. For the union step we use 3D Nef polyhedra [15] as provided by CGAL, which provide exact and efficient Boolean operations and handle all degeneracies. Our solution handles regularized solids with open or closed boundary. As a consequence, it can also be applied to degenerate scenarios like the tight passage problem in robot motion planning. The exactness of our implementation is provided by the kernels of CGAL [5] and is not discussed in this paper.

In addition to exactness, we also emphasize efficiency. We mostly concentrate on optimizing the time needed by the union of the pairwise Minkowski sums of convex pieces, which is the bottleneck of the used approach. For reducing the running time of the union it is essential to have a decomposition that yields a low number of convex pieces. The decomposition into a minimum number of convex pieces is known to be NP-hard [21]. More than 20 years ago Chazelle proposed a decomposition method, which generates $O(r^2)$ convex pieces in $O(nr^3)$ time and $O(nr^2)$ space, where n is the complexity of the polyhedron and r is the number of *reflex edges*, i.e., edges whose adjacent facets form an angle larger than 180 degrees with respect to the interior of the polyhedron. However, no robust implementation of this algorithm is known. Most of the practical methods perform surface decomposition or tetrahedral volumetric decomposition [7, 10, 16]. These methods generate $O(n)$ convex pieces of constant complexity.

As a part of our Minkowski sum implementation, we present the first robust implementation of a decomposition of general non-convex polyhedra into $O(r^2)$ convex pieces. We use Chazelle's main idea of inserting facets that resolve reflex edges. On the other hand, we construct the facets by a completely different method. Apart from the technical differences, we keep the actual number of convex pieces low by scheduling the construction of the facets in an opportune way.

To optimize the union step itself, we develop different union strategies. The union of multiple polyhedra is done by consecutive binary union operations. Here, the order of the union operations has a significant impact on the running time of the algorithm. Our union strategies use different heuristics to minimize the complexity of the intermediate results and to reduce the total amount of memory usage. We compare the efficiency of the heuristics experimentally.

The paper is organized as follows. In Sect. 2 we discuss how to efficiently decompose a non-convex polyhedron into convex pieces. The Minkowski sum of convex 3D polyhedra is discussed in Sect. 3. Section 4 compares multiple union strategies and refines the most promising. Also, we perform one large experiment to get an idea of the performance. In Sect. 5, we briefly discuss the handling of tight passage problems. Finally, a conclusion is given in Sect. 6.

2 Decomposing a Polyhedron Into Convex Pieces

The problem of partitioning a polyhedron into convex pieces is more complex than its two-dimensional counterpart. In general it is not possible to decompose a polyhedron into simplices, i.e., into tetrahedra, without introducing Steiner points [21]. The

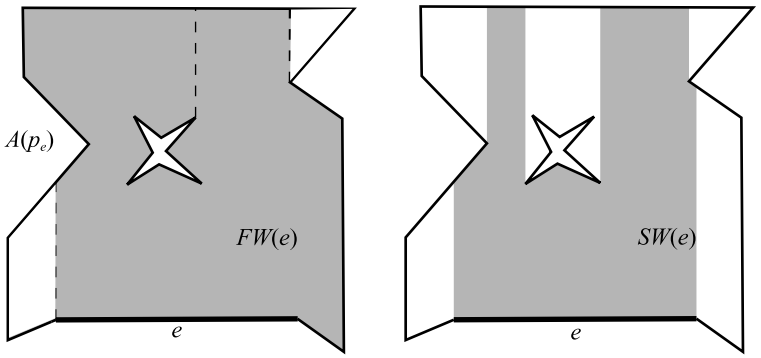

Fig. 1 *Left*: The flood wall $FW(e)$ consists of all facets adjacent to e in the planar arrangement $A(p_e)$ of the intersection of the polyhedron and the vertical plane p_e through e. The arrangement $A(p_e)$ may also contain previously inserted walls (*dashed lines*). *Right*: A sight wall $SW(e)$ covers all points that can be connected to e by a vertical edge without intersections

decomposition of a polyhedron into a minimum number of convex pieces is known to be NP-hard [21].

A basic decomposition method was introduced and analyzed by Chazelle [6]. The idea is to remove each *reflex edge*, i.e., each edge whose adjacent facets have an angle larger than 180 degree with respect to the interior of the polyhedron, by inserting an additional facet that cuts the angle into two parts smaller than 180 degrees. Chazelle showed that a polyhedron with input complexity n and r reflex edges, can be decomposed into $O(r^2)$ convex pieces in $O(nr^3)$ time and $O(nr^2)$ space. He also provided an example for which the bound of $O(r^2)$ convex sub-polyhedra is tight. For simple polyhedra Chazelle and Palios presented an algorithm that yields $O(n+r^2)$ pieces in $O(nr+r^2\log r)$ time [8]. Bajaj and Dey gave a decomposition of polyhedra with arbitrary genus that may also include voids and certain non-manifold situations. Their algorithm yields $O(r^2)$ convex pieces in $O(nr^2+r^{7/2})$ time using $O(nr+r^{5/2})$ space [3].

In this section we combine known techniques to a new decomposition algorithm, which decomposes general polyhedra into convex pieces. New ideas appear in Sect. 2.2 on how to implement the insertion of the facets that decompose the polyhedron. Our main goal is to achieve a decomposition into $O(r^2)$ convex pieces and additionally keep the number of convex pieces small.

We follow the common decomposition approach of inserting only vertical facets usually denoted as walls. A *wall* $W(e)$ of some non-vertical edge e is a connected subset of the vertical plane p_e that supports e. Walls were first defined by Aronov and Sharir [2]. Because their definition was given for a decomposition of the three-dimensional space with respect to a set of triangles we adapt their definition to our problem as follows: Let $A(p_e)$ be the planar arrangement of the intersection of the polyhedron (including previously erected walls) with the vertical plane p_e through e. Then, the wall of $W(e)$ consists of all faces of $A(p_e)$ that are incident to e and inside the polyhedron. The left graphic of Fig. 1 illustrates the planar arrangement $A(p_e)$ and the wall $W(e)$.

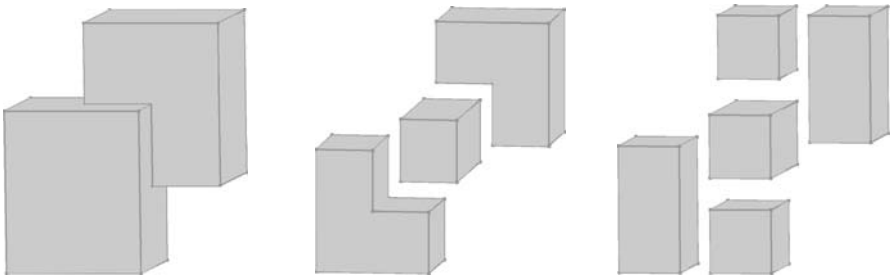

Fig. 2 Vertical decomposition (viewed from the *top*). *Left*: Non-convex polyhedron. *Middle*: Non-vertical reflex edges have been resolved by insertion of walls. *Right*: Vertical reflex edges have been resolved by insertion of y-vertical walls. The polyhedron has been decomposed into convex sub-polyhedra

The convex decomposition by walls, also denoted as *vertical decomposition*, works in two steps. In the first step, walls are erected for all non-vertical reflex edges. In the second step, walls parallel to the yz-plane, further on denoted as y-vertical walls, resolve the vertical reflex edges. Figure 2 illustrates the two steps of the vertical decomposition.

Vertical decompositions have yet only been used in the decomposition of the three-dimensional space with respect to a set of triangles. The decomposition of polyhedra is a special case of this scenario and can therefore be solved with the same methods. On the other hand, these methods are inefficient for polyhedra. Starting with a set of triangles all the edges are only incident to one triangle. An edge with only one adjacent facet surely is an reflex edge and needs to be resolved. The early decomposition methods inserted a wall for every triangle edge, every intersection edge of two triangles, and for every vertical reflex edge left after the first two steps [2]. Later Shaul and Halperin argued that a coarser decomposition is helpful for many applications and introduced the partial (vertical) decomposition [25]. They also insert a wall for every triangle edge, but then only insert further walls where it is necessary—walls for the intersection edges of two triangles, for instance, are unnecessary. In a polyhedron the number of reflex edges can be small compared to the total number of edges. Our approach is similar to the one of Shaul and Halperin. We also use the vertical decomposition as a basis, and insert only necessary walls.

To keep the number of convex cells small, we do not use walls as defined by Aronov and Sharir, but walls as later given by de Berg, Guibas, and Halperin. They define a vertical wall of e as the set of all points that can be connected to e via a vertical segment that does not intersect a face, edge, or vertex [9]. Since they also considered decompositions of the three-dimensional space with respect to triangles, we adapt their definition to our setting. For us, a vertical wall is the set of points that can be connected to e via a vertical segment that completely lies within the polyhedron. The right graphic of Fig. 1 illustrates the definition. To distinguish the two wall types, we further-on refer to the walls as defined by Aronov and Sharir [2] as flood walls and to the walls as defined by de Berg, Guibas and Halperin [9] as sight walls.

2.1 Decomposition Algorithm

Like a vertical decomposition, our algorithm performs two steps:

1. Create a sight wall for every non-vertical reflex edge.
2. For every vertical reflex edge r_v find a plane p that resolves r_v and create a new facet that is the intersection of p with the cell that contains the reflex angle.

Our decomposition improves on the vertical decomposition in three points: It inserts only walls for reflex edges, it uses sight walls instead of flood walls in the first step, and the walls in the second step need not be y-vertical walls.

We do not give a proof that a decomposition using sight walls never yields more convex pieces than a decomposition using flood walls. For the asymptotic complexity such a proof is irrelevant, because both wall types allow a decomposition into $O(r^2)$ pieces. Instead, we first give an intuition, why sight walls usually yield better results, and prove later that our decomposition yields at most $O(r^2)$ convex parts.

Consider the construction of two flood walls of reflex edges e and e'. Let us assume that $FW(e)$, if built immediately, seals a pocket in the boundary of the polyhedron. But if the flood wall $FW(e')$ is built before, it may intersect e and therefore split the pocket into two halves. Both halves of the pocket will later be sealed separately by the walls constructed for the two halves of e (see Fig. 3). Thus, depending on the building order of the flood walls, such a pocket can be decomposed into multiple pieces, although fewer pieces are sufficient. Using sight walls instead, no sight wall $SW(e')$ will split the pocket into two halves; vertical segments cannot intersect such a pocket.

The distinction of the two wall types is not necessary in the second step. Apart from the fact that the definition of sight walls does not apply to vertical edges, creating a wall by flooding the intersection of a plane and a cell of the decomposition does not imply the same disadvantages as in the first step. Because all reflex edges are

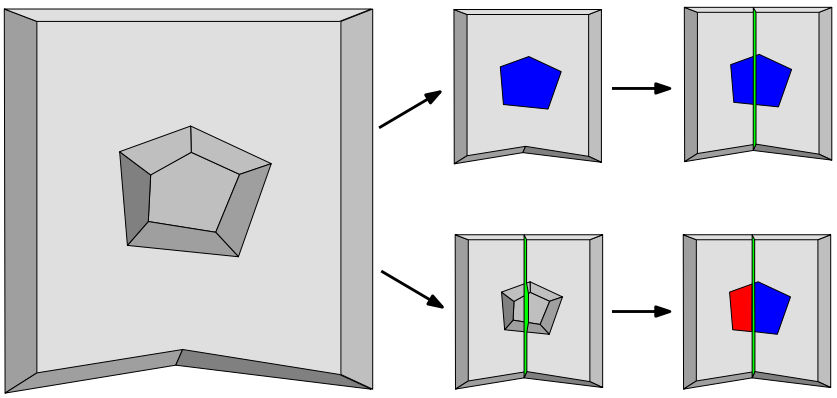

Fig. 3 Dependent upon the order of construction, flood walls decompose the polyhedron on the *left* into three or four pieces. In the *top row*, a flood wall first seals the pocket in the back and then another flood wall resolves the reflex edge on the bottom of the polyhedron (three pieces). In the *bottom row*, the reflex edge on the bottom of the polyhedron is resolved first. The respective flood wall also splits the pocket in two halves. Then the two pocket halves are sealed separately (four pieces)

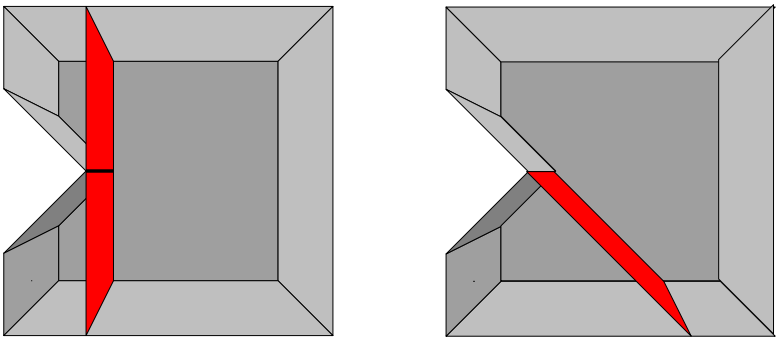

Fig. 4 The upper and lower part of a wall can each separate a cell into two pieces (*right*). If walls are created as an extension of a facet adjacent to a reflex edge, the wall will always divide the intersected cell into two pieces (*right*)

vertical, no wall can divide a reflex edge into two parts. Thus, the order of the wall creations does not influence the number of convex parts. However, in the second step the vertical plane through a reflex edge is not unique and the choice of the plane can change the number of convex parts. A y-vertical flood wall divides a cell into two or three parts, while a flood wall along one of the two facets adjacent to the reflex edge exactly splits one cell into two parts (see Fig. 4). It would be desirable that many walls actually resolve two reflex edges at once. This situation equals the problem of decomposing a polygon into a minimum number of convex parts, but it is unclear how to apply the respective algorithms to our scenario. Clearly, the vertical reflex edges match with reflex vertices in the polygon decomposition, but we don't know how the shape of the decomposed cell can be seen as a polygon.

It remains to prove the correctness of our algorithm and the complexity of the resulting decomposition. To make the phrasing of the proof simpler, we now define the notion of upper and lower walls. The wall $W(e)$ of a reflex edge e may consist of points below and points above e as illustrated by the left graphic of Fig. 4. We refer to these two parts as the *lower wall* $W^-(e)$ and *upper wall* $W^+(e)$ of e.

Lemma 1 *The sight wall $SW(e)$ of a non-vertical reflex edge e decomposes at most one cell into at most three pieces.*

Proof Every reflex edge e can form only one reflex angle, which lies within some cell c. According to our definition of sight walls, c is the only cell that might be decomposed by the sight wall $SW(e)$. If the two facets that span the reflex angle at e are on the same side s_0 of the vertical plane p_e supporting e, then $SW(e)$ consists of an upper and a lower part and the construction of $SW(e)$ decomposes c into up to three parts. On side s_0, there can be two sub-cells—one sub-cell above the upper adjacent facet of e and one sub-cell below the lower adjacent facet of e. On the other side of p_e, there can only be one sub-cell (see left graphic of Fig. 4). Note that the sight wall $SW(e)$ often only is a subset of the intersection of p_e and c. In this case c is not decomposed by $SW(e)$.

If the facets spanning the reflex angel at e are on the same side of p_e (or one of them is on p_e), then $SW(e)$ consists only of one part, which is completely above or below e. The construction of $SW(e)$ decomposes c into at most two sub-cells—one sub-cell on either side of p_e. Again, $SW(e)$ might also cause no decomposition at all.

Most important for the correctness of this proof is the observation, that no other boundary parts may intersect or touch the interior of $SW(e)$. Such boundary parts might cause additional sub-cells. Because of the definition of sight walls, $SW(e)$ cannot extent beyond any such intersection. Therefore such an intersection can never be in the interior of $SW(e)$. □

We can now prove the following theorem.

Theorem 1 *Our algorithm decomposes a general polyhedron into at most* $O(r^2)$ *convex pieces.*

Proof To prove the correctness of a convex decomposition we need to argue that all reflex edges get resolved. To evaluate an upper bound on the number of pieces resulting from the decomposition, it is sufficient to argue that any wall decomposes a cell into a constant number of pieces and then count the maximum number of inserted walls. From Lemma 1 we already know that the insertion of every sight wall causes the decomposition of at most one cell into at most three pieces. To prove the upper bound on the decomposition complexity, it is left to show that we only need to insert $O(r^2)$ vertical walls to resolve all reflex edges.

In the first step of our decomposition, we resolve all non-vertical reflex edges. The boundary edges of a sight wall $SW(e)$ inserted in this phase consists of e, intersections with the boundary of the polyhedron, and vertical edges starting from the endpoints of e. Only the last type of boundary edges can be new reflex edges, but they are vertical and therefore handled in the second step. Also, they are the intersection edges of the walls. Since the vertical supporting facet of a non-vertical edge is unique, the construction of the vertical walls for all non-vertical reflex edges can only cause $O(r^2)$ intersections. Summing up, the first step inserts $O(r^2)$ walls. Afterwards, there are no non-vertical reflex edges and at most $O(r^2)$ vertical reflex edges. The decomposition consists of at most $O(r^2)$ pieces.

In the second step, there are no non-vertical reflex edges. Thus, a new wall cannot cut a reflex edge into two parts. Because we do not create sight walls, but flood the whole intersection of the supporting plane and one decomposition cell, no new reflex edges are introduced; all new edges are intersections with the boundary of the decomposed cell. At most $O(r^2)$ walls are inserted in this step. Because of the way we choose the supporting planes of the walls, each wall decomposes one cell into exactly two parts. Thus, after step two, there will be no more reflex edges and we have a decomposition of at most $O(r^2)$ pieces. □

2.2 Implementation of the Decomposition

We construct walls by walking along the future boundary of that wall and extending the incidence structure of all the encountered boundary items. In the second step of

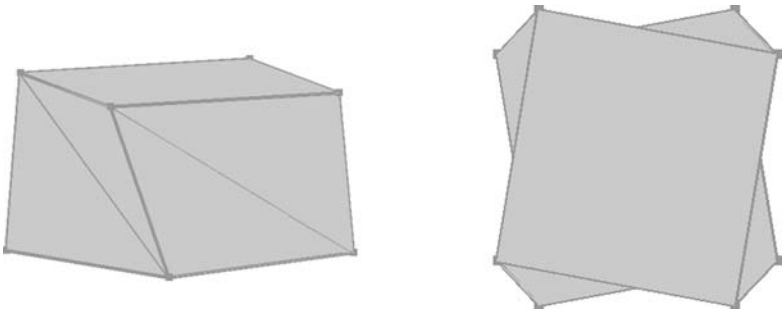

Fig. 5 Variation of Schönhardt's polyhedron with a quadrangular base viewed from the side and from the top. The diagonals of the sides are reflex edges, which are circularly dependent on one another

the decomposition, applying such a walk is simple. Given a reflex edge e, let p_e be the plane supporting the wall $W(e)$, and let c be the cell that will be decomposed by it. Then $W(e)$ can be created by walking along the intersection of c with p_e and adding the necessary incidences for the new facet to each encountered item.

In the first step of the decomposition, the boundary of a wall $W(e)$ is more complex. The boundary of $W(e)$ may not only consist of e and intersections with c, but also of intersections with other walls. If the walls are built in random order it is not guaranteed that those intersections exist when constructing a wall. Therefore, we cannot just walk along the boundary. To allow using the walk, we schedule the construction of the walls in such a way that all boundary parts of a wall exist in the moment of its construction. To do this, we must resolve mutually and cyclic dependencies.

Often, a lower wall $W^-(e)$ is part of the boundary of an upper wall $W^+(e')$, or vice versa. To avoid these dependencies, the lower and the upper walls are created in two separate sessions. Starting with the lower parts, we want to sort the reflex edges, and thereby schedule their construction, from bottom to top. In general, the edges of a polyhedron—and the same holds for the reflex edges of a polyhedron—cannot be sorted along a given direction; there can always be cyclic dependencies as illustrated by Fig. 5. But, as we will see in the following, it is possible to resolve the dependencies.

We sort the reflex edges by their lower endpoints. As a result, every pair of reflex edges has one of three relations. If (a) they do not overlap vertically, the sorting schedules the edges well. The same holds, if they overlap vertically, but their projection onto the xy-plane does not intersect. If (b) they overlap vertically and their projections onto the xy-plane intersect internally, the lower wall of one of them may be part of the other's boundary. Let's assume such an edge pair e and e', where $W^-(e)$ is part of the boundary of $W^-(e')$. If e has the smaller lower endpoint the schedule works nicely; otherwise the walk cannot proceed along $W^-(e')$'s boundary because $W^-(e)$ is missing. If (c) they share a common endpoint, the lower walls share a boundary edge. Their construction is mutually dependent.

To solve the described dependency problems, we cut some reflex edges into two or more parts and insert vertical edges from the endpoints of all reflex edges to the bottom of the polyhedron. The inserted vertical edges exactly resemble the common boundary parts of walls with common endpoints and therefore resolve the problem

described in (c). Before inserting the vertical edges, we search the sorted list of reflex edges for situation (b). If we find an edge e part of the boundary of $W(e')$, but e' is scheduled before e, then we cut e at the intersection point v_i with the vertical plane $p_{e'}$ supporting e' and put the two edge halves at their proper positions into the sorted set. Later a vertical edge will also be inserted between v_i and the bottom of the polyhedron. This vertical edge exactly resembles the intersection between $W^-(e)$ and $W^+(e')$. Note that the split of a reflex edge performed to resolve (b) can be unnecessary if e cannot be seen from e'. However, such a split does not introduce an additional convex piece. The upper walls can be handled in the same way as the lower walls.

Note that we also need to consider the degenerate case of two (or more) reflex edges that lie in the same vertical plane. The lower wall of the upper reflex edge of such a pair and the upper wall of the lower reflex edge have a two-dimensional intersection. Our implementation does not need to handle this situation explicitly. The vertical edges, which are inserted starting from the endpoints of the reflex edges, split the degenerate reflex edges. Afterwards the endpoints of each remaining degenerate edge pair lie exactly above one another, i.e., the vertical projection of the two edges is identical. Since our algorithm starts by constructing all lower walls, the construction of the upper wall of the lower edge of such an edge pair is redundant and therefore not performed.

In the second step of the decomposition, no wall intersects a reflex edge or introduces new reflex edges. Also there are no mutual and cyclic dependencies. Since we use multiple directions, the supporting planes usually intersect and therefore the decomposition is not unique. But the number of sub-polyhedra among all orders of wall creation is usually constant. Only if a wall randomly resolves several reflex edges at once, the number of sub-polyhedra decreases.

For a better overview, we sum up the implementation of our decomposition:

1. Create sight walls of non-vertical reflex edges.
   (a) Create lower sight walls.
      – Put all non-vertical reflex edges into list E.
      – Sort E by lower endpoints of the edges.
      – Cut cyclically dependent edges in E and insert the halves properly into E.
      – Add vertical edges from both endpoints of every edges in E to bottom of the polyhedron.
      – For every edge e in E create a facet by applying the walk starting at e along the boundary of $SW(e)$.
   (b) Create upper sight walls.
      – Analogous to Step 1(a).
2. Create flood walls of vertical reflex edges.

The worst case running time of our implementation is not impressive since we use kd-tree based ray shooting for the insertion of the vertical edges and for the walk along the boundary. With a worst-case running time of $O(n\sqrt[3]{n}\log n)$ per ray-shooting query [15], the worst case running time for the decomposition is

$O(n^2r^4\sqrt[3]{nr^2}\log nr)$ while using $O(nr^2)$ space. Because of the unspectacular nature of these bounds, we omit to prove them.

Our implementation is the first exact implementation of a decomposition of general polyhedra into $O(r^2)$ convex pieces. It is a powerful tool that can be used for many applications, but it is unclear whether it is the most efficient decomposition for computing Minkowski sums. Surface decompositions and volumetric decompositions do not give a decomposition of a polyhedron into convex pieces, but might be applicable even for non-approximate Minkowski sums. Varadhan and Manocha successfully used surface decomposition for their approximate implementation [26]. Unfortunately, their paper does not reveal whether the decomposition is applicable also for exact solutions. For both alternative methods there are no solutions that are sensitive to the number of reflex edges. Polyhedra often have many reflex edges and a decomposition into $O(n)$ pieces can be more effective than a decomposition into $O(r^2)$ pieces. It would also be interesting to implement a convex decomposition using a space sweep as done by Shaul and Halperin and compare it with our walk-based implementation.

3 The Minkowski Sum of Convex Polyhedra

The Minkowski sum of two convex polyhedra is also a convex polyhedron. Furthermore, it is well known that each vertex $v_{P\oplus Q}$ of the Minkowski sum $P \oplus Q$ is the vector sum of vertices v_P in P and v_Q in Q [19]. Hence, a trivial solution for the Minkowski sum of two convex polyhedra P and Q computes the convex hull of all vector sums of vertex pairs of P and Q. This algorithm performs a convex hull computation on pq vertices, where p and q are the number of vertices in P and Q. Using CGAL's `convex_hull_3` function this algorithm runs in $O(pq\log(pq))$ time.

A more efficient solution can be obtained by using normal diagrams. Each convex polyhedron P has a unique dual representation N_P called the *Gaussian diagram* or *normal diagram*. It is a subdivision of the sphere into vertices, edges and faces, such that the outward-directed normal directions of all planes supporting some item of P constitute an item of N_P. For a facet of P there is exactly one plane supporting it. Thus, its dual item is the single point on the sphere with the same normal direction as the supporting plane. The normal directions of the planes supporting an edge e_P of P form a great arc on the sphere. The endpoints of the great arc are dual items of the facets incident to e_P. A face f_n on N_P is the dual item of a vertex v_p of P. f_n is bound by a convex cycle of edges and vertices, which are the dual items of the edges and facets incident to v_p. The order of the edges and vertices around f_n coincides with the order of dual items around v_p.

The faces of $N_{P\oplus Q}$ are intersections of faces of N_P and N_Q. Moreover, the dual face of $v_{P\oplus Q}$ is the intersection of the dual faces of v_P and v_Q with $v_P + v_Q = v_{P\oplus Q}$. As a consequence, the overlay of N_P and N_Q is the normal diagram of the Minkowski sum $P \oplus Q$. Also, the exact point set of $P \oplus Q$ can easily be obtained by storing the primal vertices with their respective dual face and computing the vector sums for each face in the overlay. Thus, using the overlay of normal diagrams improves on the trivial algorithm in two points. First, the construction of $P \oplus Q$ operates on the

exact set of vertices, which might be far smaller than pq. However, in the worst case, $P \oplus Q$ still has $O(pq)$ vertices. And second, the incidence structure of $N_{P\oplus Q}$ allows us to construct $P \oplus Q$ from it in time linear to $P \oplus Q$.

With Nef polyhedra embedded on the sphere [15] as provided by CGAL, we already have a tool that can be used to realize normal diagrams, and for which we also have an overlay algorithm that can be reused for the Minkowski sum. The overlay algorithm also allows to store arbitrary data with each vertex, edge, and face, and to propagate this data properly during the overlay. The missing operations are the two conversions between a convex three-dimensional polyhedron and its normal diagram. Both functions are easy to implement.

Spherical Nef polyhedra are not the most efficient solution for the overlay of normal diagrams. Asymptotically, there is no essentially superior solution, but the *Cubical Gaussian Map* of Fogel and Halperin is clearly faster [13]. The binary operations on spherical Nef polyhedra can handle more complex overlays than those of normal diagrams, which are always convex arrangements; they never include nested faces or lower dimensional features. Therefore, our overlay algorithm is obviously more costly than needed. Apart from that, the spherical predicates are too expensive.

For the experiments of this paper, the spherical Nef polyhedra are sufficient, since the running time of computing the Minkowski sum of the convex parts is always much smaller than the union of these partial solutions. For a future release of our implementation of non-convex Minkowski sum computations, we plan to exploit other efficient implementations of algorithms that compute Minkowski sums of convex polyhedra such as the one based on Cubical Gaussian Maps [13], or the one based on Arrangement on Surfaces [4].

4 Merging a Set of Polyhedra

The union of the Minkowski sums of the convex sub-polyhedra is done by consecutive binary union operations of 3D Nef polyhedra. Since the complexity of the binary union operation depends on the complexities of both input and the result polyhedron in equal shares, it is essential not to perform the binary operations in arbitrary order. The trivial method for instance maintains one Nef polyhedron holding the current intermediate result. It starts with an empty polyhedron and adds the polyhedra one by one. This method performs very badly, since most of the union operations involve at least one big polyhedron, namely the intermediate result. Experiments showed that examples that can be computed in less than 10 minutes with efficient methods, run for more than a day with the trivial approach. Clever methods merge small polyhedra first, and try to keep intermediate results as small as possible. Since we cannot foresee the optimal order, we develop and test different strategies.

Our first method maintains the set of all unhandled primitives and intermediate results, and merges the two smallest polyhedra in each step. This can be realized by a priority queue. The priority of a polyhedron is its size measured by the number of its vertices. The priority queue is initialized with all pairwise Minkowski sums of the convex pieces. Then, repeatedly the two smallest polyhedra are extracted from the queue, and their union is inserted into the queue. The method terminates with the result left as the remaining element in the queue.

Fig. 6 Example Minkowski sums. *Top*: cup and cup ⊕ ball. *Bottom*: spoon, star, and spoon ⊕ star

Our second strategy tries to merges neighboring polyhedra to reduce the size of the intermediate results. For this purpose, we put the primitives into a queue and sort the queue by the lexicographically smallest vertex of the primitives. In order to merge polyhedra of the same complexity, we proceed similar to the priority queue approach. We extract and merge the first two polyhedra, and append their result to the end of the queue. The neighboring relation used for the sorting is maintained throughout the whole union process. Sorting the polyhedra by their lexicographically smallest vertex does not specify how we compare two such vertices in the sorting function. We test three comparison types: lexicographical comparison of the coordinates, comparison of the L1 distance from a point in a corner of the scenery, and comparison of the L2 distance from the origin.

Experimenting with larger examples, it becomes obvious that memory is a major issue in either of the above strategies. The queue-based approach can be adapted, such that no more than $\frac{\log p}{2}$ primitives need to be stored, where p is the number of primitives. Instead of a queue we maintain a stack. The primitives are computed and inserted to the stack one by one. After pushing the ith primitive onto the stack, we $\lfloor \log i \rfloor$ times pop and merge the respective top two items and push the result back onto the stack. Note that every binary union (besides the ones after the insertion of the final primitive) combines two polyhedra that are unions of the same number of primitives. Although we construct primitives just before they are pushed on the stack, we also want to sort the set of primitives in advance. For this purpose, we additionally store all normal diagrams and a sorted list of all ordered pairs of pointers to the normal diagrams that schedules the creation of the primitives. The list is sorted by the sum of the smallest vertices of the respective sub-polyhedra. Again we sort with the same three comparison types.

Table 2 summarizes the tests of the strategies. They were performed with CGAL 3.3 on a machine with a 2.4 GHz AMD Opteron processor and 4 GB RAM. The code was compiled with g++ 3.2 and compiler option -O2. The `Nef_polyhedron_3` template class was instantiated with the geometric kernel `Homogeneous<leda_integer>`. Floating-point filtering was not used in the experiments, because floating-point filtering is not efficient in combination with the `Nef_polyhedron_3` class, yet. Two experiments are illustrated in Fig. 6.

Table 1 Models used in the experiments

	cube	ball1	ball2	star	spoon	mushroom	cup
facets	6	128	1000	24	336	448	1000
parts	1	1	1	5	186	255	774

Table 2 Performance of the different union strategies, the decomposition and the Minkowski sum of the convex pieces. For each model the number of facets and the number of convex sub-polyhedra are listed in Table 1. The running times are given in seconds

model1	model2	priority queue	queue			stack			convex decomp	convex sum
			lexi	L1	L2	lexi	L1	L2		
mushroom	cube	170	151	151	152	147	147	149	43	14
mushroom	ball1	608	529	534	545	408	415	423	43	102
spoon	star	963	930	921	925	755	726	732	43	84
cup	ball2					8497			415	939

The stack strategy proved to be superior to the others. This becomes clearer the larger the examples get. The difference between the comparison types is very small. Lexicographical comparisons show the best result. The last line of the table shows the running time of a much bigger example.

Our experiments are far from comprehensive. There are many techniques that might improve the running time of the union. Amongst others, we are interested in using space-filling curves for the sorting of the primitives. These curves sort the points of a d-dimensional space along a curve, such that points that are close in the d-dimensional space are also close to one another on the curve. Space-filling curves might be more suitable than the sorting functions we used so far to reflect the proximity relation of the primitives.

5 Tight Passages

Computing the Minkowski sum of a tight passage scenario requires the handling of open polyhedra. A Nef polyhedron stores set selection marks for every vertex, edge, facet, and volume, which indicate whether the respective item is part of the polyhedron. In case of an open polyhedron the marks of the boundary items are unselected. These selection marks can additionally be stored in the normal diagram. During the overlay operation the marks are transferred in the following way. Given normal diagrams N_P and N_Q, we consider intersecting items i_P and i_Q. Their intersection forms item $i_{P\oplus Q}$ in the overlay $N_P \oplus N_Q$. Then, the selection mark stored with $i_{P\oplus Q}$ can be computed as $i_P \wedge i_Q$. This means for instance, that a vertex $v_{P\oplus Q} = v_P + v_Q$ of $P \oplus Q$ is selected iff v_P and v_Q are selected.

If we allow arbitrary selection marks for the input, i.e., each boundary part may have a different mark, the computation of the Minkowski sum becomes more complicated. The reason is, that each side of a convex sub-polyhedron may consist of

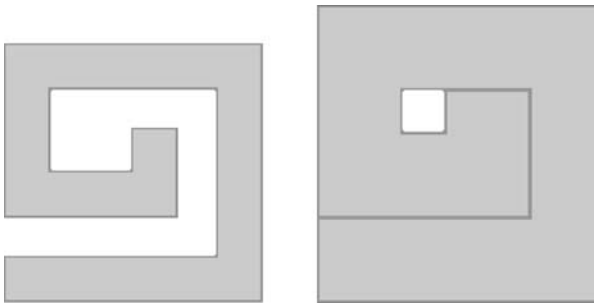

Fig. 7 A maze whose corridors have unit width, and the Minkowski sum of a unit cube and the maze

several facets, some of which are boundary parts of the input and some of which are walls inserted by the decomposition. The selection marks of these facets can differ. As a consequence, the selection marks cannot be handled as described above.

Fortunately, we don't need arbitrary selection marks to handle tight passages. It is only necessary to allow polyhedra that are either open or closed, i.e., all selection marks of boundary items are either selected or unselected. In this situation it suffices to ignore the selection marks of the walls. If the side of a sub-polyhedron consists of original facets and walls, we use the uniform mark of the original facets; if there are only walls, we can assign an arbitrary mark. Naturally, the walls must be selected because they represent a point set inside a polyhedron. Thus, not selecting a wall yields an unselected facet in a convex Minkowski sum pr_1 that should be selected. On the other hand, this wall is adjacent to another sub-polyhedron. Because of the adjacency and because of the convexity of primitives, the convex Minkowski sums computed on this other sub-polyhedron must generate a primitive pr_2 that overlaps pr_1 such that the wrongly unselected facet is in the interior of pr_2. The final result of the Minkowski sum operation will not contain any of these facets.

Figure 7 shows a tight passage scenario solved with our implementation.

6 Conclusion

We have presented an exact implementation of the Minkowski sum of non-convex polyhedra. Our implementation also supports open and closed polyhedra, which allows to solve extreme scenarios like tight passage problems.

As part of our Minkowski sum, we have also presented the first robust decomposition of non-convex polyhedra into $O(r^2)$ convex pieces, where r is the number of reflex edges.

As part of future work, we want to further improve the efficiency of our implementation and release it as part of CGAL. We plan two major points of improvement. First, we want to improve the second step of the decomposition by adapting polygon decomposition methods for the decomposition of the cylindrical cells. Those methods would sometimes resolve two vertical reflex edges with one wall and therefore generate fewer convex pieces. Furthermore, we plan to replace our solution for the convex Minkowski sums by a faster approach.

Acknowledgements The author likes to thank Lutz Kettner for stimulating the work of this paper. I thank Efi Fogel for helping me repeat the comparison of theirs and out implementation of the Minkowski sum on convex polyhedra. I thank Liangjun Zhang and Dinesh Manocha for providing 3D models, such that I could repeat two of the experiments in [26]. Furthermore, I woud like to thank Mark de Berg for valuable discussions on the presented topic. Finally, I thank the anonymous reviewers of this paper, who helped me improve the paper considerably.

References

1. Agarwal, P.K., Flato, E., Halperin, D.: Polygon decomposition for efficient construction of Minkowski sums. Comput. Geom.: Theory Appl. **21**, 39–61 (2002)
2. Aronov, B., Sharir, M.: Triangles in space or building (and analyzing) castles in the air. Combinatorica **10**(2), 137–173 (1990)
3. Bajaj, C.L., Dey, T.K.: Convex decomposition of polyhedra and robustness. SIAM J. Comput. **21**(2), 339–364 (1992)
4. Berberich, E., Fogel, E., Halperin, D., Mehlhorn, K., Wein, R.: Sweeping and maintaining two-dimensional arrangements on surfaces: A first step. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) 15th European Symposium on Algorithms, ESA 07, Eilat, Israel, October 2007. Lecture Notes in Computer Science, vol. 4698, pp. 645–656. Springer, Berlin (2007)
5. The CGAL Homepage. http://www.cgal.org/
6. Chazelle, B.: Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. SIAM J. Comput. **13**(3), 488–507 (1984)
7. Chazelle, B., Dobkin, D., Shouraboura, N., Tal, A.: Strategies for polyhedral surface decomposition: an experimental study. Comput. Geom.: Theory Appl. **7**(5–6), 327–342 (1997)
8. Chazelle, B., Palios, L.: Triangulating a nonconvex polytope. Discrete Comput. Geom. **5**, 505–526 (1990)
9. de Berg, M., Guibas, L.J., Halperin, D.: Vertical decompositions for triangles in 3-space. Discrete Comput. Geom. **15**(1), 35–61 (1996)
10. Ehmann, S.A., Lin, M.C.: Accurate and fast proximity queries between polyhedra using convex surface decomposition. In: Computer Graphics Forum, Proc. Eurographics 2001, vol. 20(3), pp. 500–510 (2001)
11. Elber, G., Kim, M.-S. (eds.): Special Issue of Computer Aided Design: Offsets, Sweeps and Minkowski Sums, vol. 31 (1999)
12. Evans, R.C., O'Connor, M.A., Rossignac, J.R.: Construction of Minkowski sums and derivatives morphological combinations of arbitrary polyhedra in CAD/CAM systems. US Patent 5159512 (1992)
13. Fogel, E., Halperin, D.: Exact and efficient construction of Minkowski sums of convex polyhedra with applications. Comput. Aided Des. **39**(11), 929–940 (2007)
14. Guibas, L.J., Ramshaw, L., Stolfi, J.: A kinetic framework for computational geometry. In: 24th Symposium on Foundations of Computer Science (FOCS), pp. 100–111 (1983)
15. Hachenberger, P., Kettner, L., Mehlhorn, K.: Boolean operations on 3D selective Nef complexes: Data structure, algorithms, optimized implementation and experiments. Comput. Geom.: Theory Appl. **38**(1–2), 64–99 (2007)
16. Joe, B.: A software package for the generation of meshes using geometric algorithms. Adv. Eng. Softw. Workst. **13**(5–6), 325–331 (1991)
17. Kaul, A., Rossignac, J.R.: Solid-interpolating deformations: Construction and animation of pips. Computers & Graphics **16**(1), 107–115 (1992)
18. Kim, Y.J., Otaduy, M.A., Lin, M.C., Manocha, D.: Fast penetration depth computation using rasterization hardware and hierarchical refinement. In: Proc. of Workshop on Algorithmic Foundations of Robotics (2002)
19. Latombe, J.-C.: Robot Motion Planning. Kluwer Academic, Norwell (1991)
20. Lin, M.C., Manocha, D.: Collision and proximity queries. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, pp. 787–808. CRC Press LLC, Boca Raton (2004), Chap. 35

21. O'Rourke, J.: Art Gallery Theorems and Algorithms. Oxford University Press, New York (1987)
22. Rossignac, J.R.: Private communication (2007)
23. Rossignac, J.R., Requicha, A.A.G.: Offsetting operations in solid modelling. Comput. Aided Geom. Des. **3**(2), 129–148 (1986)
24. Rössl, C., Kobbelt, L., Seidel, H.-P.: Extraction of feature lines on triangulated surfaces using morphological operators. In: Proc. of the 2000 AAAI Symp. (2000)
25. Shaul, H., Halperin, D.: Improved construction of vertical decompositions of three-dimensional arrangements. In: SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry, New York, NY, USA, pp. 283–292. ACM Press, New York (2002)
26. Varadhan, G., Manocha, D.: Accurate Minkowski sum approximation of polyhedral models. Graph. Models **68**(4), 343–355 (2006)
27. Wein, R.: Exact and efficient construction of planar Minkowski sums using the convolution method. In: 14th European Symposium on Algorithms (ESA 06), pp. 829–840 (2006)
28. Wesley, M.A., Lozano-Prez, T., Lieberman, L.I., Lavin, M.A., Grossman, D.D.: A geometric modeling system for automated mechanical assembly. IBM J. Res. Dev. **24**(1), 64–74 (1980)