

# Polyhedral Assembly Partitioning with Infinite Translations or *The Importance of Being Exact*\*

Efi Fogel and Dan Halperin

**Abstract** Assembly partitioning with an infinite translation is the application of an infinite translation to partition an assembled product into two complementing subsets of parts, referred to as subassemblies, each treated as a rigid body. We present an exact implementation of an efficient algorithm to obtain such a motion and subassemblies given an assembly of polyhedra in  $\mathbb{R}^3$ . We do not assume general position. Namely, we handle degenerate input, and produce exact results. As often occurs, motions that partition a given assembly or subassembly might be isolated in the infinite space of motions. Any perturbation of the input or of intermediate results, caused by, for example, imprecision, might result with dismissal of valid partitioning-motions. In the extreme case, where there is only a finite number of valid partitioning-motions, no motion may be found, even though such exists. The implementation is based on software components that have been developed and introduced only recently. They paved the way to a complete, efficient, and concise implementation. Additional information is available at <http://acg.cs.tau.ac.il/projects/internal-projects/assembly-partitioning/project-page>.

## 1 Introduction

Assembly planning is the problem of finding a sequence of motions that move the initially separated parts of an assembly to form the assembled product. The reversed order of sequenced motions separates an assembled product to its parts. Thus, for

---

Efi Fogel

School of Comp. Science, Tel-Aviv University, Tel Aviv 69978, Israel, e-mail: [efif@post.tau.ac.il](mailto:efif@post.tau.ac.il)

Dan Halperin

School of Comp. Science, Tel-Aviv University, Tel Aviv 69978, Israel, e-mail: [danha@post.tau.ac.il](mailto:danha@post.tau.ac.il)

\* This work has been supported in part by the Israel Science Foundation (grant no. 236/06), by the German-Israeli Foundation (grant no. 969/07), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

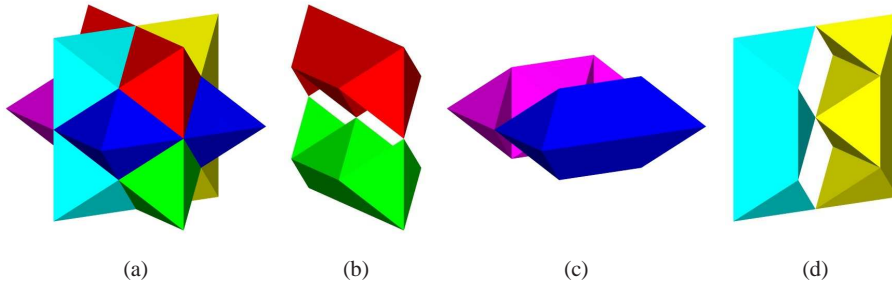


Fig. 1: (a) The Split Star puzzle, and (b),(c), and (d) the Split Star six parts divided into three pairs of symmetric parts. The six parts are named according to their color *R*(ed), *G*(reen), *B*(lue), *P*(urple), *Y*(ellow), and *T*(urquoise).

rigid parts, assembly planning and disassembly planning refer to the same problem, and used interchangeably. In this paper we concentrate on the case where the assembly consists of polyhedra in  $\mathbb{R}^3$  and the motions are infinite translations.

The motion space is the space of possible motions that assembly parts may undergo. For each motion in a motion space, a subassembly of a given assembly may collide with a different subassembly, when transformed as a rigid body according to the motion. Pairs of subassemblies that collide constitute constraints. The motion space approach dictates the precomputation of a decomposition of a motion space into regions, such that the constraints among the parts in the assembly are the same for all the motions in the same region. All constraints over a region are represented by a graph, called the *directional blocking graph* (DBG) [23]. The collection of all regions in a motion space together with their associated DBGs can be used to obtain assembly (or disassembly) sequences.

Degenerate input is commonplace in computational-geometry applications, and numerical errors are inevitable when arithmetic based on machine number-type (e.g., floating-point) is used to carry out algebraic computation. Traditional implementations, which ignore these observations, may yield incorrect results, get into an infinite loop, or just crash, while running on a degenerate, or nearly degenerate, input (see [15] for examples). The problem intensifies in assembly planning, as motions might be isolated in the infinite space of motions. Any perturbation of the input or of intermediate results, caused by, for example, imprecision, might result with dismissal of valid partitioning-motions. In the extreme case, where there is only a finite number of valid partitioning-motions, as occurs in the assembly shown in Figure 1, no motion may be found, even though such exists.

The general framework and some of the techniques presented here have already been described in a series of papers and reports published in the past mainly during the late 90's. Halperin, Latombe, and Wilson made the connection between previously presented techniques that had used the motion space approach, and introduced a unified general framework [11] at the end of the previous millennium. Only few publications related to this topic appeared ever since, to the best of our knowledge, which creates a long gap in the time line of the respective research. We certainly

hope that the tools exposed in this paper will help revive the research on algorithmic assembly planning, a research subject of considerable importance. Moreover, we believe that the machinery presented here, together with other recent advances in the practice of computational-geometry algorithms, can more generally support the development of new and improved techniques in *algorithmic automation*.<sup>2</sup>

Solution to the assembly planning problem enables better feedback to designers. It helps them to create products that are more cost-effective to manufacture and maintain. This is emphasized in light of the strategy to “plan anywhere, build anywhere” many CAD/CAM companies are trying to adopt. Assembly sequences are also useful for selecting assembly tools and equipment, and for laying out manufacturing facilities.

We restrict ourselves to two-handed partitioning steps, meaning that we partition the given assembly into two complementing subsets each treated as a rigid body. Even for two-handed partitioning, if we allow arbitrary translational motions (and not restrict ourselves to infinite translations) the problem becomes NP-hard [12]. The assembly-sequencing general problem of planning a total ordering of assembly operations that merge the individual parts into the assembled product, is PSPACE-hard, even when each part is a polygon with a constant number of vertices [18].

Notice that the problem that we address in this paper, namely partitioning with *infinite translations*, is technically considerably more complex than partitioning with *infinitesimal motions*. Although the latter may sound more general, as it handles infinitesimal translations and *rotations*, it is far simpler to implement, since it deals only with constraints that can be described linearly. Thus, the problem can be reduced to solving linear programs. Indeed, there are several implementations for partitioning with infinitesimal motions (see, e.g., [9, 19]), but none that we are aware of, dealing robustly with infinite translations. The shortcoming of using infinitesimal motions only is that suggested disassembly moves may not be extendible to practical finite-length separation motions.

Infinite-translation partitioning was not fully robustly implemented until recently, in spite of being more useful than infinitesimal partitioning, most probably due to the hardship of accurately constructing the underlying geometric primitives. What enables the solution that we present here, is the significant headway in the development of computational-geometry software over the past decade, the availability of stable code in the form of the Computational Geometry Algorithms Library (CGAL)<sup>3</sup> in general and code for Minkowski sums of polytopes in  $\mathbb{R}^3$  and arrangements in  $\mathbb{R}^2$  in particular [22].

The implementation presented in this paper is based on a package of CGAL called `Arrangement_on_surface_2` [21]. It supports the robust construction and maintenance of arrangements of curves embedded on two-dimensional parametric surfaces [2], and robust operations on them, e.g., overlay computation. The implementation uses in particular arrangements of geodesic arcs embedded on the sphere. It exploits supported operations, and requires additional operations, e.g., polyhedra

---

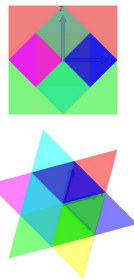
<sup>2</sup> <http://goldberg.berkeley.edu/algorithmic-automation>

<sup>3</sup> <http://www.cgal.org>

central projection, which we implemented. We plan to make these new components available as part of a future public release of CGAL as well. The ability to robustly construct such arrangements, and carry out exact operations on them using only (exact) rational arithmetic is a key property that enables an efficient implementation.

### 1.1 Split Star Puzzle

We use the assembly depicted in Figure 1 as a running example throughout the paper. The name “Split Star” was given to this shape by Stewart Coffin in one of his Puzzle Craft booklet editions back in 1985. He uses the term *puzzle* to refer to any sort of geometric recreation having pieces that come apart and fit back together. We use it as an assembly. He describes how to produce the actual pieces out of wood [4], and suggests that they are made very accurately. He observes that finding the solution requires dexterity and patience, when the pieces are accurately made with a tight fit. Even though the assembly seems relatively simple, this should come as little surprise, since the first partitioning motion is one out of only eight possible translations of four symmetric pairs of motions in opposite directions associated with two complementing subassemblies of three parts each. Evidently, any automatic process that introduces even the slightest error along the way, will most likely fail to compute the correct first motion in the sequence, and falsely claim that the assembly is interlocked.



The Split Star assembly has the assembled shape of the first stellation of the rhombic dodecahedron [17], illustrated atop the right pedestal in M. C. Escher’s Waterfall woodcut [3]. Its orthogonal projection along one of its fourfold axes of symmetry is a square, while the Star of David is obtained when it is projected along one of its threefold axes of symmetry, as seen on the left; for more details see [4]. The assembly is a space-filling solid

when assembled. It consists of six identical concave parts. Each part can be decomposed into eight identical tetrahedra yielding 48 tetrahedra in total. As manufacturing the pieces requires extreme precision, it is suggested to produce the 48 identical pieces and glue them as necessary. Each part can also be decomposed into three convex polytopes — two square pyramids and one octahedron, yielding 18 polytopes in total. The partitioning described in this paper requires the decomposition of the parts into convex pieces. The choice of decomposition may have a drastic impact on the time consumption of the entire process, as observed in a different study in  $\mathbb{R}^2$  [1], and shown by experiments in Section 5.

## 1.2 Outline

The rest of this paper is organized as follows. The partitioning algorithm is described in Section 2 along with the necessary terms and definitions. In Section 3 we provide implementation details. Section 4 presents optimizations that are not discussed in the preceding sections, some of which we have already implemented, and proved to be useful. We report on experimental results in Section 5.

## 2 The Partitioning Algorithm

The main problem we address in this paper, namely, *polyhedral assembly partitioning with infinite translations*, is formally defined as follows: Let  $A = \{P_1, P_2, \dots, P_n\}$  be a set of  $n$  pairwise interior disjoint polyhedra in  $\mathbb{R}^3$ .  $A$  denotes the assembly that we aim to partition. We look for a proper subset  $S \subset A$  and a direction  $\mathbf{d}$  in  $\mathbb{R}^3$ , such that  $S$  can be moved as a rigid body to infinity along  $\mathbf{d}$  without colliding with the rest of the parts of the assembly  $A \setminus S$ . (We allow sliding motion of one part over the other. We disallow the intersection of the interior of two polyhedra.)

We follow the NDBG approach [23], and describe it here using the general formulation and notation of [11]. The *motion space* in our case, namely the space of all possible partitioning directions, is represented by the unit sphere  $\mathbb{S}^2$ . Every point  $p$  on  $\mathbb{S}^2$  defines the direction from the center of  $\mathbb{S}^2$  towards  $p$ . Every direction  $\mathbf{d}$  is associated with the directed graph  $DBG(\mathbf{d}) = (V, E)$  that encodes the blocking relations between the parts in  $A$  when moved along  $\mathbf{d}$  as follows: The nodes in  $V$  correspond to polyhedra in  $A$ ; we denote a node corresponding to the polyhedron  $P_i$  by  $v(P_i) \in V$ . There is an edge directed from  $v(P_i)$  to  $v(P_j)$ , denoted  $e(P_i, P_j) \in E$ , if and only if the interior of the polyhedron  $P_i$  intersects the interior of the polyhedron  $P_j$  when  $P_i$  is moved to infinity along the direction  $\mathbf{d}$ , and  $P_j$  remains stationary.

The key idea behind the NDBG approach is that in problems such as ours, where the number of parts is finite, and any allowable partitioning motion can be described by a small number of parameters, there is only a relatively small (polynomial) number of distinct DBGs that need to be constructed in order to detect a possible partitioning direction. Stated differently, the motion space can be represented by an arrangement subdivided into a finite number of cells each assigned with a fixed DBG. Once this arrangement is constructed, we construct the DBG over each cell of the arrangement, and check it for strong connectivity. A DBG that is not strongly connected is associated with a direction, or a set of directions in case the cell is not a singular point, that partition the given assembly. The desired movable subset  $S \subset A$  is a byproduct of the algorithm that checks for strong connectivity. If all the DBGs over all the cells of the arrangement are strongly connected, we conclude that the assembly is *interlocked*, as a subset of the parts in  $A$  that can be separated from the rest of the assembly by an infinite translation does not exist.

Next we show how to construct the motion-space arrangement and compute the DBG over each one of the arrangement cells. Each ordered pair of distinct polyhedra

$\langle P_i, P_j \rangle$  defines a region  $Q_{ij}$  on  $\mathbb{S}^2$ , which is the union of all the directions  $\mathbf{d}$ , such that when  $P_i$  is moved along  $\mathbf{d}$  its interior will intersect the interior of  $P_j$ . How can we effectively compute this region? Let  $M_{ij}$  denote the Minkowski sum  $P_j \oplus (-P_i) = \{b - a \mid a \in P_i, b \in P_j\}$ . We claim that the central projection of  $M_{ij}$  onto  $\mathbb{S}^2$  is exactly  $Q_{ij}$ .

**Lemma 0.1.** *A direction  $\mathbf{d}$  is in the interior of the central projection of  $M_{ij}$  onto  $\mathbb{S}^2$  if and only if when  $P_i$  is moved along  $\mathbf{d}$  its interior will intersect the interior of  $P_j$ .*

*Proof.* Let  $\mathbf{d}$  be some direction in the central projection of  $M_{ij}$  onto  $\mathbb{S}^2$ . In other words, there exists a point  $m \in M_{ij}$ , such that  $m = s \cdot \mathbf{d}$ , for some positive scalar  $s$ . As  $m$  is in  $M_{ij}$ , there exist two points  $p_i \in P_i$  and  $p_j \in P_j$ , such that  $m = p_j - p_i$ . Thus,  $p_j = p_i + s \cdot \mathbf{d}$ , meaning that the point  $p_i$  intersects  $p_j$  when moved along  $\mathbf{d}$ . A similar argument can be used to show the other way around.  $\square$

Next, we describe how, given two polyhedra  $P_i$  and  $P_j$ , we compute the region  $Q_{ij}$ , using a robust and efficient hierarchy of building blocks, which we have developed in recent years. The existing tools that we use are (i) computing the arrangement of spherical polygons [2, 8, 7, 22], and (ii) construction of Minkowski sums of convex polytopes [2, 5, 8, 7]. We also need some extra machinery, as explained below.

Assume  $P_i$  is given as the union of a collection of (not necessarily disjoint) convex polytopes  $P_1^i, P_2^i, \dots, P_{m_i}^i$ , and similarly  $P_j$  is given as the collection of convex polytopes  $P_1^j, P_2^j, \dots, P_{m_j}^j$ . It is easily verified that  $M_{ij} = \bigcup_{k=1, \dots, m_i, \ell=1, \dots, m_j} P_\ell^j \oplus (-P_k^i)$ . So we compute the Minkowski sum of each pair  $P_\ell^j \oplus (-P_k^i)$ , and centrally project it onto  $\mathbb{S}^2$ . Finally, we take the union of all these projections to yield  $Q_{ij}$ .

There are several ways to effectively compute the central projection of a convex polyhedron  $C$  (one of the polytopes  $M_{kl}^{ij} = P_\ell^j \oplus (-P_k^i)$ ) from the origin onto  $\mathbb{S}^2$ . We opted for the following. An edge  $e$  of  $C$  is called a *silhouette edge*, if the plane  $\pi$  through the origin and  $e$  is tangent to  $C$  at  $e$ . Namely, it intersects  $C$  in  $e$  only. We assume for now that there is no tangent plane that contains a facet of  $C$ ; we relax this assumption in Section 3.5, where we provide a detailed description of the procedure. We traverse the edges of  $C$  till we find a silhouette edge  $e_1$ . One can verify that the silhouette edges form a cycle on  $C$ . We start with  $e_1$ , and search for a silhouette edge adjacent to  $e_1$ . We proceed in the same manner, till we end up discovering  $e_1$  again. Projecting this cycle of edges onto  $\mathbb{S}^2$  is straightforward.

All the boundaries of the regions  $Q_{ij}$  form an arrangement of geodesic arcs on the sphere. We traverse the motion-space arrangement in say a breadth-first fashion. For the first face we check which one of the regions  $Q_{ij}$  contain it. We construct the corresponding DBG and check it for strong connectivity. If it is not strongly connected, we stop and announce a solution as described above. Otherwise we move to an adjacent feature of the current face. During this move we may step out from a set of regions  $Q_{ij}$ , and may step into a new set of regions  $Q_{ij}$ . We update the current DBG according to the regions we left or entered, test the new DBG for strong connectivity, and so on till the traversal of all the arrangement cells is complete. Notice that it is important to visit also vertices and edges of the arrangement, since

the solution may not lie in the interior of a face. Indeed, in our Split Star example, solutions are on vertices of the arrangement. Without careful exact constructions, such solutions could easily be missed.

### 3 Implementation Details

The implementation of the assembly-partitioning operation consists of eight phases listed below. They all exploit arrangements of geodesic arcs embedded on the sphere [2, 7] in various ways. The `Arrangement_on_surface_2` package of CGAL already supports the construction and maintenance of such arrangements, the computation of union of faces of such arrangements, the construction of Gaussian maps of polyhedra represented by such arrangements, and the computation of their Minkowski sums. It provides the ground for efficient implementation of the remaining required operations, such as central projection.

1. **Convex Decomposition**
2. **Sub-part Gaussian map construction**
3. **Sub-part Gaussian map reflection**
4. **Pairwise sub-part Minkowski sum construction**
5. **Pairwise sub-part Minkowski sum projection**
6. **Pairwise Minkowski sum projection**
7. **Motion-space construction**
8. **Motion-space processing**

The partitioning process is implemented as a free (general) function that accepts as input an ordered list of polyhedra in  $\mathbb{R}^3$ , which are the parts of the assembly. Each part is represented as a polyhedral mesh in  $\mathbb{R}^3$ . A polyhedral mesh representation consists of an array of vertices and a collection of facets, where each facet is described by an array of indices into the vertex array. We proceed with a detailed discussion of the implementation of each phase.

We deal below with various details that are typically ignored in reports on geometric algorithms (for example, under the general-position assumption). However, in assembly planning, or more generally in movable-separability problems in tight scenarios, much of the difficulty shifts exactly to these technical details and in particular to handling degeneracies. This is especially emphasized in Phases 5 and 6 (Subsections 3.5 and 3.6 respectively), but prevails throughout the entire section.

#### 3.1 Convex Decomposition

We decompose each concave part into convex polyhedra referred to as sub-parts. The output of this phase is an ordered list of parts, where each part is an ordered list of convex sub-parts represented as polyhedral surfaces. Each polyhedral surface is maintained as a CGAL `Polyhedron_3` [14] data-structure, which consists of vertices,

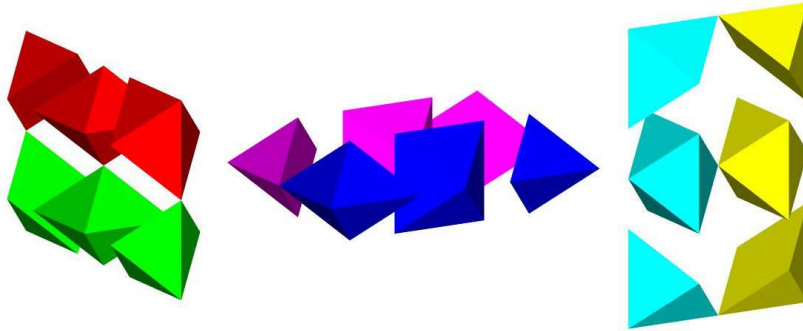


Fig. 2: The six parts of the Split Star decomposed into three convex sub-parts each.

edges, and facets and incidence relations on them [13]. A part that is convex to start with is simply converted into an object of type `Polyhedron_3`.

A new package of CGAL that supports convex decomposition of polyhedra has been recently introduced [10], but has not become publicly available yet. As we aim for a fully automatic process, we intend to exploit such components, once they become available, and study their impact. For the time being we resorted to a manual procedure. A simple decomposition of the Split Star parts used in the running example is illustrated in Figure 2.

### 3.2 Sub-part Gaussian Map Construction

The *Gaussian Map*  $G = G(P)$  of a compact convex polyhedron  $P$  in Euclidean three-dimensional space  $\mathbb{R}^3$  is a set-valued function from  $P$  to the unit sphere  $\mathbb{S}^2$ , which assigns to each point  $p$  on the boundary of  $P$  the set of outward unit normals to support planes to  $P$  at  $p$ . A vertex  $v$  of  $P$  is mapped by  $G$  to a spherical polygon  $G(v)$  [6]. Likewise, the inverse Gaussian Map, denoted by  $G^{-1}$ , maps the spherical features to the polytope boundary.

We convert each sub-part represented as a polyhedral surface into a Gaussian map, represented as an arrangement of geodesic arcs embedded on the sphere, where each face  $f$  of the arrangement is extended with the coordinates of its associated primal vertex  $v = G^{-1}(f)$ , resulting with a unique representation. The construction of an arrangement from the polytope features and their accessible incident relations provided by the `Polyhedron_3` data-structure amounts to the insertion of geodesic segments that are pairwise disjoint in their interior into the arrangements, an operation that can be carried out efficiently.

The output of this phase is an ordered list of parts, where each part is an ordered list of the Gaussian maps of the convex sub-parts. Figure 3 depicts the Gaussian maps of six of the 18 polytopes that comprise the set of sub-parts of our Split Star assembly.

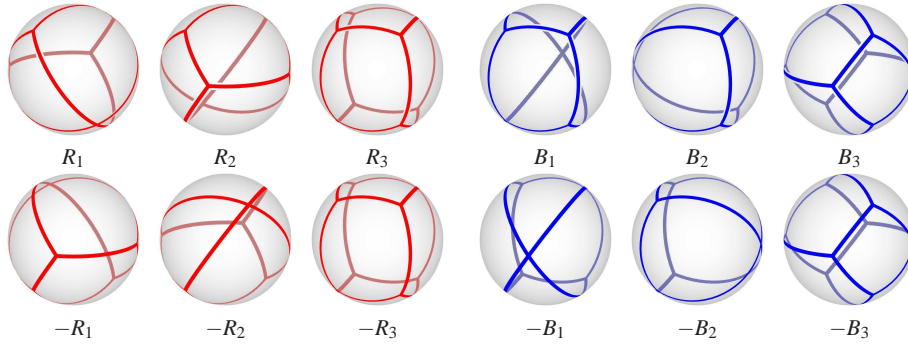


Fig. 3: Samples of the Gaussian maps of sub-parts of the Split Star assembly. The bottom row contains the reflections of the Gaussian maps at the top row.

### 3.3 Sub-part Gaussian Map Reflection

We reflect each sub-part  $P_k^i$  through the origin to obtain  $-P_k^i$ . This operation can be performed directly on the output of the previous phase, reflecting the underlying arrangements of geodesic arcs embedded on the sphere, which represent the Gaussian maps, while handling the additional data attached to the arrangement faces. As a matter of fact, this phase can be reduced as part of an optimization discussed in Section 4.

The output of this phase is an ordered list of parts, where each part is an ordered list of Gaussian maps of the reflected convex sub-parts. Figure 3 depicts the Gaussian maps of six of the 18 polytopes that comprise the set of reflected sub-parts of the Split Star example.

### 3.4 Pairwise Sub-part Minkowski Sum Construction

---



---

Construct Pairwise Sub-part  
Minkowski Sums

---

**for**  $i$  **in**  $\{1, 2, \dots, n\}$   
  **for**  $j$  **in**  $\{1, 2, \dots, n\}$   
    **if**  $i == j$  **continue**  
    **for**  $k$  **in**  $\{1, 2, \dots, m_i\}$   
      **for**  $\ell$  **in**  $\{1, 2, \dots, m_j\}$   
         $M_{k\ell}^{ij} = P_k^i \oplus (-P_\ell^j)$

---

We compute the Minkowski sums of the pairwise sub-parts and reflected sub-parts. Aiming for an efficient output-sensitive algorithm, the construction of an individual Minkowski sum from two Gaussian maps is performed by overlaying the two arrangements. When the overlay operation progresses, new vertices, edges, and faces of the resulting arrangement are created based on features of the two operands. When a new feature is created its attributes are updated. There are ten cases that must be handled [5]. For example, a new face  $f$  is induced by the overlap of two faces  $f_1$

and  $f_2$  of the two summands respectively. The primal vertex associated with  $f$  is set to be the sum of the primal vertices associated with  $f_1$  and  $f_2$  respectively.

The `Arrangement_on_surface_2` package conveniently supports the overlay operation allowing users to provide their own version of these ten operations. The overlay operation is exploited below on several different variants of arrangements of geodesic arcs embedded on the sphere. Each application requires the provision of a different set of those ten operations.

The output of this phase is a map from ordered pairs of distinct indices into lists of Minkowski sums represented as Gaussian maps. Each ordered pair  $\langle i, j \rangle, i \neq j$  is associated with the list of Minkowski sums  $\{M_{k\ell}^{ij} \mid k = 1, 2, \dots, m_i, \ell = 1, 2, \dots, m_j\}$ . In case of our Split Star the map consists of 30 entries that correspond to all configurations of ordered distinct pairs of parts. Each entry consists of a list of nine Minkowski sums, that is, 270 Minkowski sums in total.

### 3.5 Pairwise Sub-part Minkowski Sum Projection

---



---

```

Project Pairwise Sub-part
      Minkowski sums
-----
for  $i$  in  $\{1, 2, \dots, n\}$ 
  for  $j$  in  $\{1, 2, \dots, n\}$ 
    if  $i == j$  continue
    for  $k$  in  $\{1, 2, \dots, m_i\}$ 
      for  $\ell$  in  $\{1, 2, \dots, m_j\}$ 
         $Q_{k\ell}^{ij} = \text{project}(M_{k\ell}^{ij})$ 

```

---

We centrally project all pairwise sub-part Minkowski sums computed in the previous phase onto the sphere. Each projection is represented as an arrangement of geodesic arcs on the sphere, where each cell  $c$  of the arrangement is extended with a Boolean flag that indicates whether all infinite rays emanating from the origin in all directions  $\mathbf{d} \in c$  pierce the corresponding Minkowski sum. As

the Minkowski sums are convex, their spherical projections are spherically convex.

Given a convex Minkowski sum  $C$ , we distinguish between four different cases as follows:

1. The origin is contained in the interior of a facet of  $C$ .
2. The origin lies in the interior of an edge of  $C$ .
3. The origin coincides with a vertex of  $C$ .
4. The origin is separated from  $C$ .

Computing the projection of a convex polytope  $C$  can be done efficiently using dedicated procedures that handle the four cases respectively. Recall that  $C$  is represented as a Gaussian map, which is internally represented as an arrangement of geodesic arcs embedded on the sphere. We traverse the vertices of the arrangement. For each vertex  $v$  we extract its associated primal facet  $f = G^{-1}(v)$ . We dispatch the appropriate computation based on the relative position of the origin with respect to the supporting plane to  $f$ , and the supporting plane to adjacent facets of  $f$ .

**If the origin is contained in the interior of a facet  $f$  of  $C$ ,** the projection of the silhouette of  $C$  is a great (full) circle that divides the sphere into two hemispheres. The normal to the plane that contains the great circle is identical to the normal to the

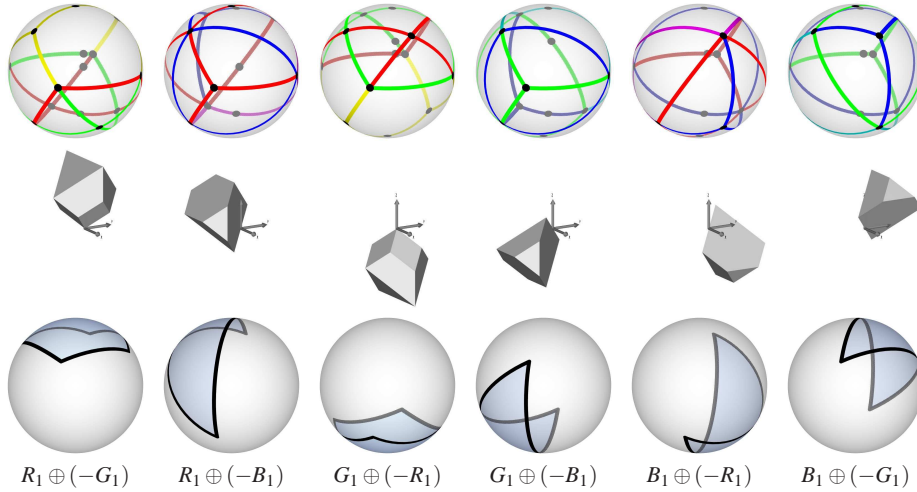
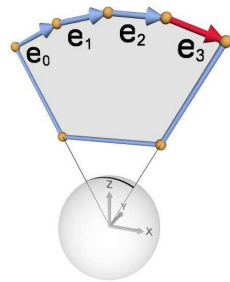


Fig. 4: Samples of the pairwise Minkowski sums of sub-parts of the Split Star assembly. The middle row contains six Minkowski sums. The top row contains the corresponding Gaussian maps. The bottom row contains the corresponding central projections of the Minkowski sums on  $\mathbb{S}^2$ .

supporting plane to  $f$ , easily extracted from the arrangement representing the Gaussian map of  $C$ . The `Arrangement_on_surface_2` package conveniently supports the insertion of a great circle, provided by the normal to the plane that contains it, into an arrangement of geodesic arcs embedded on the sphere.

We omit the implementation details of the succeeding two cases, and proceed to the general case. **If the origin is separated from  $C$ ,** we traverse all edges of  $C$  until we find a silhouette edge characterized as follows: Let  $v_s$  and  $v_t$  be the source and target vertices of some edge  $e$  in the arrangement representing the Gaussian map of  $C$ , and let  $f_s = G^{-1}(v_s)$  and  $f_t = G^{-1}(v_t)$  be their associated primal facets respectively.  $e$  is a silhouette edge, if and only if, the origin is not in the negative side of the supporting plane to  $f_s$  and not in the positive side of the supporting plane to  $f_t$ . We start with the first silhouette edge we find, and search for an adjacent silhouette edge in



a loop, until we rediscover the first one. We project only the target vertices of significant silhouette edges, and connect consecutive projections using arcs of great circle. Let  $e$  and  $e'$  be adjacent silhouette edges. We skip  $e$ , if the projections of  $e$  and  $e'$  lie on the same great circle. For example, all but the last adjacent silhouette edges incident to a facet supported by a plane that contains the origin are redundant, as illustrated in the figure on the left. Here we skip  $e_0$ ,  $e_1$ , and  $e_2$ , and project the target vertex of  $e_3$ .

The output of this phase is a map from ordered pairs of distinct indices into lists of arrangements as described above. Each ordered pair  $\langle i, j \rangle, i \neq j$  is associated with the list of central projections of the pairwise Minkowski sums of  $P_j$ 's sub-parts and the reflection through the origin of  $P_i$ 's sub-parts.

### 3.6 Pairwise Minkowski Sum Projection

---



---

```

Unite Pairwise Sub-part
Minkowski sums Projections
for  $i$  in  $\{1, 2, \dots, n\}$ 
  for  $j$  in  $\{1, 2, \dots, n\}$ 
    if  $i == j$  continue
     $Q_{ij} = \emptyset$ 
    for  $k$  in  $\{1, 2, \dots, m_i\}$ 
      for  $\ell$  in  $\{1, 2, \dots, m_j\}$ 
         $Q_{ij} = Q_{ij} \cup Q_{k\ell}^{ij}$ 

```

---



---

For each pair of distinct parts  $P_i$  and  $P_j$  we compute the union of projections of the pairwise Minkowski sums of all sub-parts of part  $P_j$  and reflections of all sub-parts of part  $P_i$ .

The output of this phase is a map from ordered pairs of distinct indices into arrangements. Each ordered pair  $\langle i, j \rangle, i \neq j$  is associated with a single arrangement extended as described above, that represents the central projection  $Q_{ij}$  of  $M_{ij} = P_j \oplus (-P_i)$ .

We exploit the overlay operation in this phase the second time throughout this process, this time in a loop. Given two distinct parts  $P_i$  and  $P_j$  we traverse all projections in the set  $\{Q_{k\ell}^{ij} \mid k = 1, 2, \dots, m_i, \ell = 1, 2, \dots, m_j\}$ , and accumulate the result in the arrangement  $Q_{ij}$ . As mentioned in Section 3.4, when the overlay operation progresses, new vertices, edges, and faces of the resulting arrangement are created. When a new face  $f$  is created as a result of the overlay of a face  $g$  in some projection  $Q_{k\ell}^{ij}$ , and a face in the accumulating arrangement, the Boolean flag associated with  $f$ , which indicates whether all directions  $\mathbf{d} \in f$  pierce  $M_{ij}$ , is turned on, if  $\mathbf{d}$  pierces  $M_{k\ell}^{ij}$ , that is, if the flag associated with the face of  $g$  is on.

The intermediate result of this step are arrangements with potentially redundant edges and vertices. It is desired, (but not necessary,) to remove these cells, as this will reduce the time consumption of the succeeding operations, which is directly related to the complexity of the arrangements. It has even a larger impact when the optimization described in Section 4 is applied, as the optimization decreases the number of preceding operation at the account of slightly increasing the number of succeeding operations. We remove all edges and vertices that are in the interior of the projection, that is all marked edges and vertices. We also remove spherically collinear vertices on the boundary of the projection, the degree of which decreased below three, as a result of the redundant-edge removal.

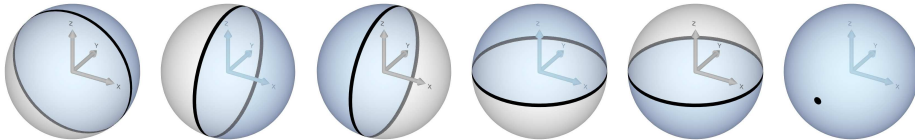
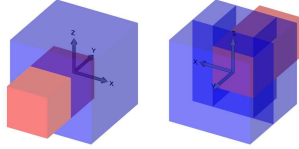


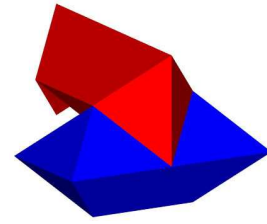
Fig. 5: Peg-in-the-hole Minkowski sum projections. (a), (b), (c), (d), and (e) are the sub-part projection. (f) is the union of all the former.

CGAL also supports the union operation among other Boolean operations applied to general polygons.<sup>4</sup> However, it consumes and produces *regularized* general polygons. This regularization operation is harmful in the realm of assembly



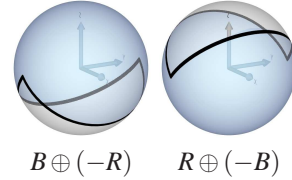
planning. Therefore, we work directly on the cells of the arrangements  $Q_{ij}$ . Quite often the projection contains isolated vertices and edges, as occurs in the example depicted on the left, referred to as “peg-in-the-hole”. Here the assembled product is translucently viewed from two opposite directions. The blue part is stationary and is decomposed into five sub-parts. Figure 5 illustrates the corresponding five pairwise Minkowski sum projections, and their union. The complement of the union consists of a single isolated vertex.

Recalling our Split Star assembly, the projection of the Minkowski sum of the red part and the reflection of the blue part, and its reflection, that is, the projection of the Minkowski sum of the blue part and the reflection of the red part are depicted on the right.



### 3.7 Motion-Space Construction

We compute a single arrangement that represents the motion space, where each cell  $c$  of the arrangement is extended with a DBG. We use the adjacency-matrix storage format provided by BOOST<sup>5</sup> to represent each DBG. Recall that for a graph with  $n$  vertices such as ours, an  $n \times n$  matrix is used, where each element  $a_{ij}^c$  of a DBG associated with cell  $c$  is a Boolean flag that indicates whether part  $P_i$  collides with part  $P_j$  when moved along any direction  $\mathbf{d} \in c$ . Handling large assemblies with sparse blocking relations may require different representations of DBGs to reduce memory consumption.



We exploit the overlay operation in this phase the third time similar to its application in Section 3.6. We iterate over all central projections in the set  $Q_{ij}$ , and accumulate the result in the final motion-space arrangement. As mentioned above in Section 3.4, when the overlay operation progresses, new vertices, edges, and faces of the resulting arrangement are created. When a new cell  $c$  is created as a result of the overlay of a face  $g$  in some projection  $Q_{ij}$ , and a cell in the accumulating arrangement, the DBG associated with  $c$  is updated. That is, if the flag associated with  $g$  is turned on, we insert an edge between vertex  $i$  and vertex  $j$  into the DBG associated with  $c$ .

Depicted on the right is the motion-space arrangement computed by our program for the Split Star assembly.



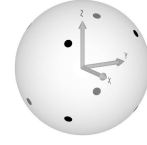
<sup>4</sup> The code supports point sets bounded by algebraic curves embedded on parametric surfaces referred to as general polygons.

<sup>5</sup> <http://www.boost.org>

### 3.8 Motion-Space Processing

We traverse all vertices, edges, and faces of the motion-space arrangement in this order, and test the DBG associated with each cell for strong connectivity using the BOOST global function `strong_components()`. This function computes the strongly connected components of a directed graph using Tarjan's algorithm based on DFS [20]. The set of constraints associated with a vertex  $v$  is a proper subset of the constraints associated with edges incident to  $v$ . Similarly, the set of constraints associated with an edge  $e$  is a proper subset of the constraints associated with the two faces incident to  $e$ . Therefore, if the DBGs of all vertices are strongly connected, we terminate with the conclusion that the assembly is interlocked. Similarly, if we are interested in finding all solutions, and the DBGs of all edges are strongly connected, we terminate, as no further solutions on faces exist.

For the Split Star assembly, our program successfully identifies all the eight partitioning directions depicted above along with the corresponding subset of parts listed on the right.



|    | Direction    | Subset     |
|----|--------------|------------|
| 1. | $-1, -1, -1$ | <i>GBT</i> |
| 2. | $-1, -1, 1$  | <i>RBT</i> |
| 3. | $-1, 1, -1$  | <i>GPT</i> |
| 4. | $-1, 1, 1$   | <i>RPT</i> |
| 5. | $1, -1, -1$  | <i>GBY</i> |
| 6. | $1, -1, 1$   | <i>RBY</i> |
| 7. | $1, 1, -1$   | <i>GPY</i> |
| 8. | $1, 1, 1$    | <i>RPY</i> |

## 4 Additional Optimization

The reflection of the sub-parts through the origin as described in Section 3.3 has been naively implemented. The computation is applied to the polyhedral-mesh representation of each sub-part. An immediate optimization calls for an application of the reflection operation directly on the arrangements that represent the Gaussian map. We are planning to implement the reflection operation, which operates on any applicable arrangement. This operation alters incidence relations between the arrangement features and their geometric embeddings. For each vertex, it reflects its associated point about the origin, and inverts the order of the halfedges incident to it. For each edge, it reflects its associated curve about the origin. For each face, it inverts the order of the halfedges along its outer boundary. Similar to the overlay operation (see Section 3.4), where the user can provide a set of ten functions, which are invoked when new vertices, edges, and faces of the resulting arrangement are created, while the overlay operation progresses, the user can provide a set of three functions that are invoked when a new vertex, halfedge, and face are created, while the reflection operation progresses. Extended data associated with these types, such as a primal vertex associated with an arrangement face as in the case of an arrangement representing a Gaussian map, can easily be updated with the provision of an appropriate function.

The trivial observation that  $P \oplus (-Q) = -((-P) \oplus Q)$  leads to another optimization. Instead of reflecting all sub-parts in the set  $\{P_k^i \mid i = 1, 2, \dots, n, k = 1, 2, \dots, m_i\}$ , we reflect only the sub-parts in the set  $\{P_k^i \mid i = 2, \dots, n, k = 1, 2, \dots, m_i\}$ , and com-

Table 1: Time consumption (in seconds) of the execution of the eight phases (see Section 3) using the Split Star assembly as input. **A** — number of convex sub-parts per part. **B** — number of sub-part vertices per part. **C** — total number of convex sub-parts. **D** — total number of Minkowski sums. **E** — total number of arrangements of geodesic arcs embedded on the sphere constructed throughout the process.

| <b>A</b> | <b>B</b> | <b>C</b> | <b>D</b> | <b>E</b> | <b>1</b> | <b>2</b> | <b>3</b> | <b>4</b> | <b>5</b> | <b>6</b> | <b>7</b> | <b>8</b> |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 3        | 16       | 18       | 270      | 607      | NA       | 0.01     | 0.04     | 2.38     | 0.41     | 2.05     |          |          |
| 5        | 22       | 30       | 750      | 1591     | NA       | 0.01     | 0.05     | 5.03     | 1.09     | 7.07     | 0.36     | 0.01     |
| 8        | 32       | 48       | 1920     | 3967     | NA       | 0.01     | 0.06     | 11.12    | 2.41     | 27.99    |          |          |

pute only the pairwise sub-parts Minkowski sums in the set  $\{M_{k\ell}^{ij} \mid 1 \leq i < j \leq n, k = 1, 2, \dots, m_i, \ell = 1, 2, \dots, m_j\}$ , their central projection, and the union of the appropriate projections to yield the set  $\{Q_{ij} \mid 1 \leq i < j \leq n\}$ . Then, we apply the reflection operation described above on each member of this set, and obtain the full set of projections  $\{Q_{ij} \mid i = 1, 2, \dots, n, j = 1, 2, \dots, n\}$ . The Boolean flag associated with a face of an arrangement that represents a central projection is equal to the flag associated with its reflection. In other words, a face of  $Q_{ij}$  consists of directions that pierce  $M_{ij}$ , if and only if, its reflection in  $Q_{ji}$  consists of directions that pierce  $M_{ji}$ .

Phase 8 is purely topological. Thus, we do not expect the time consumption of this phase to dominate the time consumption of the entire process for any input. Nevertheless, it might be possible to reduce its contribution to the total time consumption through efficient testing for strong connectivity applied to all the DBGs [16], exploiting the similarity between DBGs associated with incident cells. Recall, that the set of arcs in a DBG associated with a vertex  $v$  is a subset of the set of arcs associated with an edge incident to  $v$ . Similarly, the set of arcs in a DBG associated with an edge  $e$  is a subset of the set of arcs associated with a face incident to  $e$ . The proposed technique reduces the cost from  $O(n^2)$  per DBG to an amortized cost of  $O(n^{1.376})$ , where  $n$  is the maximum number of arcs in any blocking graph.

## 5 Experimental Results

Our program can handle all inputs. However, due to lack of space, we limit ourselves to a small set of test cases, where we compare the impact of different decompositions on the process time-consumption. The results listed in Table 1 were produced by experiments conducted on a Pentium PC clocked at 1.7 GHz. In all three test cases we use the Split Star assembly as input. Naturally, in all three cases identical projections are obtained as the intermediate results of Phase 6, hence the identical time consumption of the succeeding last two phases. Evidently, it is desired to decompose each part into as few as possible sub-parts with as small as possible number of features. However, an automatic decomposition operation may require large amount of resources to arrive at optimal or near optimal decompositions. Notice that Phases 4 and 6 dominate the time complexity. This is due to the large number of geometric predicates that must be evaluated during the execution of the overlay operation.

## References

1. P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of Minkowski sums. *Comput. Geom. Theory Appl.*, 21:39–61, 2002.
2. E. Berberich, E. Fogel, D. Halperin, K. Melhorn, and R. Wein. Sweeping and maintaining two-dimensional arrangements on surfaces: A first step. In *Proc. 15th Annu. Eur. Symp. Alg.*, pages 645–656, 2007.
3. F. H. Bool et al. *M. C. Escher: His Life and Complete Graphic Work*. Harry N. Abrams, Inc., 1982.
4. S. T. Coffin. *Geometric Puzzle Design*. A.K. Peters, Ltd., 2nd edition, 2006.
5. E. Fogel and D. Halperin. Exact and efficient construction of Minkowski sums of convex polyhedra with applications. *CAD*, 39(11):929–940, 2007.
6. E. Fogel, D. Halperin, and C. Weibel. On the exact maximum complexity of Minkowski sums of convex polyhedra. In *Proc. 23rd Annu. ACM Symp. Comput. Geom.*, pages 319–326, 2007.
7. E. Fogel, O. Setter, and D. Halperin. Exact implementation of arrangements of geodesic arcs on the sphere with applications. In *Abstracts of 24th Eur. Workshop Comput. Geom.*, pages 83–86, 2008.
8. E. Fogel, O. Setter, and D. Halperin. Movie: Arrangements of geodesic arcs on the sphere. In *Proc. 24th Annu. ACM Symp. Comput. Geom.*, pages 218–219, 2008.
9. L. J. Guibas, D. Halperin, H. Hirukawa, J.-C. Latombe, and R. H. Wilson. Polyhedral assembly partitioning using maximally covered cells in arrangements of convex polytopes. *Int. J. Comput. Geom. Appl.*, 8:179–200, 1998.
10. P. Hachenberger. Exact Minkowski sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces. In *Proc. 15th Annu. Eur. Symp. Alg.*, volume 4698 of *LNCS*, pages 669–680. Springer, 2007.
11. D. Halperin, J.-C. Latombe, and R. H. Wilson. A general framework for assembly planning: The motion space approach. *Algorithmica*, 26:577–601, 2000.
12. L. Kavraki and M. Kolountzakis. Partitioning a planar assembly into two connected parts is NP-complete. *Inf. Process. Lett.*, 55:159–165, 1995.
13. L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Comput. Geom. Theory Appl.*, 13(1):65–90, 1999.
14. L. Kettner. 3D polyhedral surfaces. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.3 edition, 2007.
15. L. Kettner, K. Melhorn, S. Pion, S. Schirra, and C. K. Yap. Classroom examples of robustness problems in geometric computations. In *Proc. 12th Annu. Eur. Symp. Alg.*, volume 3221 of *LNCS*, pages 702–713. Springer, 2004.
16. S. Khanna, R. Motwani, and R. H. Wilson. On certificates and lookahead in dynamic graph problems. *Algorithmica*, 21(4):377–394, 1998.
17. D. Luke. Stellations of the rhombic dodecahedron. *The Math. Gazette*, 41(337):189–194, 1957.
18. B. K. Natarajan. On planning assemblies. In *Proc. 4th ACM Symp. Comput. Geom.*, pages 299–308, 1988.
19. J. Snoeyink and J. Stolfi. Objects that cannot be taken apart with two hands. *Disc. Comput. Geom.*, 12:367–384, 1994.
20. R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. on Computing*, 1(2):146–160, 1972.
21. R. Wein, E. Fogel, B. Zukerman, and D. Halperin. 2D arrangements. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.3 edition, 2007.
22. R. Wein, E. Fogel, B. Zukerman, and D. Halperin. Advanced programming techniques applied to CGAL’s arrangement package. *Comput. Geom. Theory Appl.*, 38(1–2):37–63, 2007. Special issue on CGAL.
23. R. H. Wilson and J.-C. Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71(2):371–396, 1994.