Optimal Algorithms for Separating a Polyhedron from its Single-Part Mold

Prosenjit Bose
School of Computer Science, Carleton University, Ottawa, Canada

Efi Fogel
The Blavatnik School of Computer Science, Tel Aviv University, Israel

Tzvika Geft
The Blavatnik School of Computer Science, Tel Aviv University, Israel

Dan Halperin
The Blavatnik School of Computer Science, Tel Aviv University, Israel

Shahar Shamai
The Blavatnik School of Computer Science, Tel Aviv University, Israel

Abstract

Casting is a manufacturing process where liquid material is poured into a mold having the shape of a desired product. After the material solidifies, the product is removed from the mold. We study the case where the mold is made of a single part and the object to be produced is a three-dimensional polyhedron. Objects that can be produced this way are called castable with a single-part mold. A direction in which the object can be removed without breaking the mold is called a valid removal direction. We give an $O(n)$-time algorithm that decides whether a given polyhedron with n facets is castable with a single-part mold. When possible, our algorithm provides an orientation of the polyhedron in the mold and a direction in which the product can be removed without breaking the mold. Moreover, we provide an optimal $\Theta(n \log n)$-time algorithm to compute all valid removal directions for polyhedra that are castable with a single-part mold. Both algorithms are an improvement by a linear factor over the previously best known algorithms for both of these problems. We also present an exact implementation of our algorithms using the CGAL library and employ our implementation to demonstrate the castability of a variety of polyhedra.

Keywords and phrases Casting, Cast removal, Cast design, Separability

Digital Object Identifier 10.57717/cgt.v3i1.15

Funding This work has been supported in part by the Israel Science Foundation (grant nos. 1736/19 and 2261/23), by NSF/US-Israel-BSF (grant no. 2019754), by the Israel Ministry of Science and Technology (grant no. 103129), by the Blavatnik Computer Science Research Fund, by The Yandex Machine Learning Initiative for Machine Learning at Tel Aviv University, and by the Natural Sciences and Engineering Research Council of Canada.

1 Introduction

Casting is a widely used manufacturing process, where liquid material is poured into a cavity inside a mold, which has the shape of a desired product. After the material solidifies, the product is taken out of the mold. Typically, a mold is used to manufacture numerous copies of the product; thus to ensure that a mold can be re-used, the solidified product must be separated from its mold without breaking the object or the mold.

The problems that we study belong to the larger topic termed *Movable Separability of Sets*; see [24]. Problems in this area are often challenging from a combinatorial- and computational-geometry point of view (see, e.g., [22]). At the same time, solutions to these problems are needed in various application areas such as mold design [2], assembly planning [15], and 3D printing [4] to mention a few.

In this paper we focus on a basic movable-separability question. We are given a polyhedron P in $\mathbb{R}^3$ with n facets. No assumptions about the polyhedron are made beyond that it is a closed regular set; namely, it does not have dangling edges or facets. The mold is box-shaped and the cavity has the shape of P rotated in such a way that one of P's facets becomes the top of the cavity; see Figure 1 for an illustration in 2D. Any facet that becomes the top of the cavity is referred to as a *top facet*. For any top facet, we detect whether there is a direction in which the solidified object can be removed from the mold, such that its interior does not collide with the mold. Such a direction is a *valid* removal direction and the corresponding top facet is a *valid* top facet. A polyhedron P is *castable with a single-part mold* if P has at least one valid top facet.

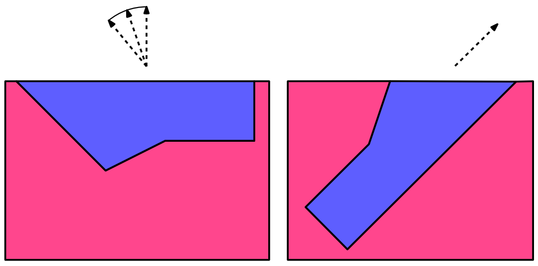


Figure 1 Polygons (blue) in their molds (pink) and valid removal directions.

We address two problems:

1. **All Facets Single Direction** (AllFSD):
   Determine which facets of P are valid top facets, and for each such facet indicate *one* valid removal direction.
2. **All Facets All Directions** (AllFAD):
   Same as above, but for each valid facet indicate *all* the valid removal directions.

Why would anyone bother to solve AllFAD and not be satisfied with AllFSD? First, a solution is more stable if there is a continuum of directions rather than a single direction of separation. Second, we can use the availability of many possible directions to optimize other criteria, e.g., minimizing the product's contact with the mold during removal.

1.1 Previous results

The previous best algorithms that we are aware of are based on [2, 4] and solve AllFAD in $O(n^2 \log n)$ time and AllFSD in $O(n^2)$ time. These algorithms are summarized in [13, Chapter 4], which we refer to here after. They solve the AllFAD and AllFSD problems by solving the following two simpler problems n times—handling each facet as a candidate top facet separately:

1. **Single Facet Single Direction** (SingleFSD):
   Determine whether a given facet F_i of a polyhedron P is a valid top facet and if so, indicate *one* direction in which P can be removed from the mold with F_i as its top facet.
2. **Single Facet All Directions** (SingleFAD):
   Same as above except indicate *all* the directions in which P can be removed from the mold with F_i as its top facet.

The existing algorithm for SingleFSD takes $O(n)$ time and the existing algorithm for SingleFAD takes $O(n \log n)$ time. A variant of AllFSD in which we restrict the removal direction to be the outer normal of the top facet is solvable in $O(n)$ time [4]. For this variant, the top facet on the left of Figure 1 would be valid but not the one on the right. The linear running time stems from the existence of a set of six candidate facets that need to be considered. However, when considering all possible removal directions for a facet, as we do here, it was not known how to find such a set of constant size. All the algorithms including the ones that we present use linear storage space. Related work on computational aspects of manufacturing processes has been done for casting with a two-part mold [2, 8], gravity casting, and stereolithography [4]; see also the survey [10].

1.2 Contribution

Our contribution is an $O(n)$-time algorithm for the AllFSD problem and an $O(n \log n)$-time algorithm for the AllFAD problem. We obtain the improved running time by showing that it suffices to consider a *constant* number of candidate top facets. Specifically, we prove that for any polyhedron there are at most six valid top facets. As a result, we can solve AllFSD (resp. AllFAD) by solving SingleFSD (resp. SingleFAD) a constant number of times, rather than n times as in the previous best algorithm. Both our algorithms have an optimal running time. We also present an $O(n)$-time algorithm for the AllFAD problem when the input polyhedron is convex.

The efficiency of our solution makes it a good candidate for implementation, which may enable product designers and engineers to quickly verify the castability of their design. Indeed, the algorithms that solve AllFSD, AllFAD, SingleFSD, SingleFAD, in the plane have been implemented and are available as free functions in the Cgal (Computational Geometry Algorithms Library) package *2D Movable Separability of Sets* [21]. The implementation of the corresponding algorithms in three-dimensional space is currently private and can be obtained upon request; see Section 7.

A previous version of this paper [9] was presented at the IEEE 13th International Conference on Automation Science and Engineering (CASE 2017). In this paper we extend our previous work by presenting an $\Omega(n \log n)$ lower bound for the AllFAD (and SingleFAD) problem, proving that our $O(n \log n)$-time algorithm is optimal. Moreover, we present here our implementation of the algorithms along with examples and experiments of their application to various polytopes.

2 Preliminary analysis

Instead of considering all of the facets as candidate top facets and running a separate algorithm for each of them, as in [13, Chapter 4], we start by finding a small set (whose size is bounded by a constant) of top facets. Then we solve SingleFSD/SingleFAD [13, Chapter 4] for each of the candidates.

In order to find the possible top facets we consider an arrangement of great circles on the unit sphere $\mathcal{S}^2$. An arrangement of curves on the sphere is a subdivision of the sphere into vertices, edges, and faces induced by the given curves: Vertices are the intersection points of the curves, edges are the maximal portions of a curve not intersected by any other curve, and faces are the maximal portions of the sphere that are not intersected by any curve; see, e.g., [6, 7]. Each point p on $\mathcal{S}^2$ represents a direction in $\mathbb{R}^3$—the direction of the vector from the center of $\mathcal{S}^2$ to p. We use the terms *points* and *directions* on $\mathcal{S}^2$ interchangeably.

Remark. In the sequel, we illustrate our main concepts in 3D. The reader may also benefit from an illustration of their analogs in 2D; see "Illustration" in Section 4. We occasionally refer to the 2D figures.

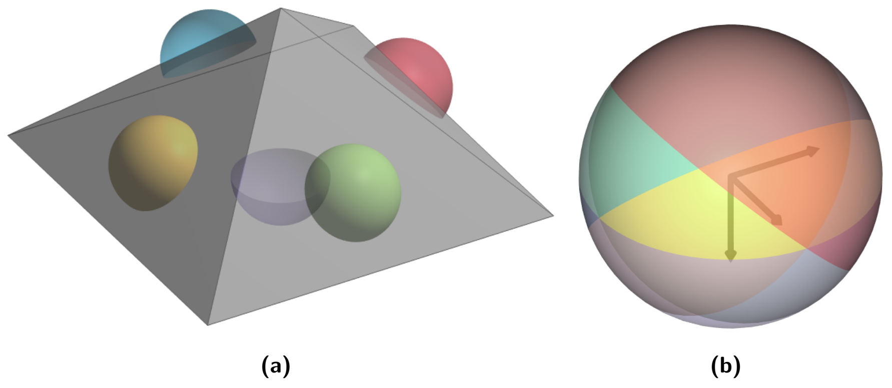
Let $F_1, \ldots, F_n$ be the facets of the given polyhedron P. Let $\nu(F_i)$ be the normal to the facet F_i pointing inwards, into the polyhedron. Every facet F_i is associated with a rotation R_i, such that F_i becomes a top facet when R_i is applied to P. During our analysis, we use the notation $\vec{d}$ to refer to a removal direction. For convenience this direction is relative to the given original orientation of the polyhedron. Once we determine that $(F_i, \vec{d})$ is a pair of a valid top facet and a corresponding valid removal direction, we still need to apply the rotation R_i to our polyhedron, so that F_i becomes the top facet, to yield the exact shape of the cavity. We also need to apply R_i to d to yield a valid removal direction in the coordinate system of the mold.

▶ **Definition 1.** *A* valid pair *is a pair $(F_i, \vec{d})$ of a valid top facet and a corresponding valid removal direction.*

▶ **Observation 2.** *The pair $(F_i, \vec{d})$ is valid if and only if for each point p in the polyhedron, the ray that emanates from p with direction $\vec{d}$*
(i) intersects F_i, and
(ii) $\forall j \neq i$, the ray does not intersect F_j. (It may partially overlap[1] *F_j.)*

▶ **Lemma 3.** *The pair $(F_i, \vec{d})$ is valid if and only if*
(i) $\vec{d} \cdot \nu(F_i) < 0$, and
(ii) $\forall j \neq i, \;\; \vec{d} \cdot \nu(F_j) \geq 0$

Proof. For the case where F_i is the top facet of P, this fact is proved in Lemma 4.1 in [13]. It remains to notice that conditions (i) and (ii) of Observation 2 are invariant under rotation. They hold in P's given orientation if and only if they hold when P is rotated such that F_i becomes the top facet. ◀



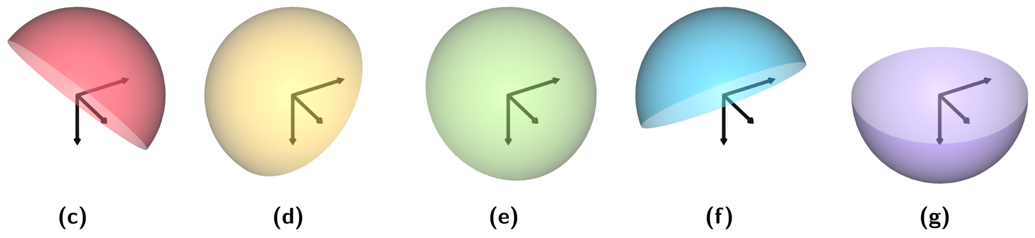


Figure 2 (a) A pyramid with a hemisphere of directions $\bar{h}(F_i)$ on each of its facets. (c,d,e,f,g) The individual hemispheres. (b) The overlay of these hemispheres on $\mathcal{S}^2$.

[1]This corresponds to allowing P to be moved out while sliding in contact with F_j.

Denote by $h_i := h(F_i)$ the closed hemisphere of directions $\vec{d}$ on $\mathcal{S}^2$ for which $\vec{d} \cdot \nu(F_i) \geq 0$, and by $\bar{h}_i$ or $\bar{h}(F_i)$ the open complement hemisphere. For a set of facets X we denote by $\bar{H}(X)$ the set $\{\bar{h}(F_i) \mid F_i \in X\}$. Let $\bar{H} = \bar{H}(\{F_1, \ldots, F_n\}) = \{\bar{h}_1, \ldots, \bar{h}_n\}$. Let c_i denote the boundary great circle of h_i, and let $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$. Consider the arrangement $\mathcal{A}(\mathcal{C})$ on $\mathcal{S}^2$, namely, the subdivision of $\mathcal{S}^2$ induced by the great circles in $\mathcal{C}$. See Figure 2 for an illustration.

▶ **Definition 4.** *The* depth *of a point p on $\mathcal{S}^2$ is the number of hemispheres in $\bar{H}$ containing p.*

▶ **Observation 5.** *All the points in any fixed cell of the arrangement $\mathcal{A}(\mathcal{C})$ have the same depth.*

▶ **Lemma 6.** *There cannot be a face of zero depth in $\mathcal{A}(\mathcal{C})$.*

Proof. Assume, for the sake of contradiction, that there exists a face of zero depth in the arrangement $\mathcal{A}(\mathcal{C})$. Let $\vec{d}$ be a direction in this face. If we choose an arbitrary point inside the polyhedron (not on its boundary) and go from it in the direction $\vec{d}$ we will eventually leave the polyhedron through some facet F_j. That is, $\vec{d}$ must be pointing out of the polyhedron from F_j, an hence $\vec{d}$ is in $\bar{h}(F_j)$, in contradiction to $\vec{d}$ lying inside a face with zero depth.

Notice that in order to avoid special degenerate cases, like crossing the polyhedron in an edge, we choose the arbitrary point such that it does not lie on any of the planes spanned by the direction $\vec{d}$ and an edge of the polyhedron. ◀

▶ **Lemma 7.** *The polyhedron P is castable with a single-part mold if and only if the arrangement $\mathcal{A}(\mathcal{C})$ contains a point of depth 1. A cell ξ of depth 1 in $\mathcal{A}(\mathcal{C})$, which is covered by the hemisphere $\bar{h}_i$, represents a mold whose top facet is F_i and each point $\vec{d}$ in ξ represents a valid removal direction $R_i\vec{d}$, where R_i is the orthonormal matrix that rotates $\nu(F_i)$ to point vertically down (in the negative z direction).*

Proof. Let ξ be a cell of depth 1 in $\mathcal{A}(\mathcal{C})$ covered by $\bar{h}_i$, and let $\vec{d}$ be a point in ξ. We establish that $(F_i, \vec{d})$ is a valid pair by verifying that the conditions of Lemma 3 above hold for it.

It remains to show that no point in any cell of different depth can represent a valid removal direction for any top facet. Consider a cell ψ of depth greater than 1 and a point $\vec{d}$ in it. Let J be the index set of the hemispheres $\bar{h}_i$ that cover ψ: $J = \{i \,|\, \vec{d} \in \bar{h}_i\}$. One of the facets F_j, for $j \in J$, must serve as the top facet for Condition (i) of Lemma 3 to hold. But then for each of the remaining facets F_k, for $k \in J, k \neq j$, Condition (ii) of the lemma is violated. Finally, as shown in Lemma 6, no cell can have depth zero. ◀

Let $\mathcal{T} \subseteq \{F_1, \ldots, F_n\}$ be the set of all *valid top facets*. Our goal is to find $\mathcal{T}$. In Section 3 we show that $|\mathcal{T}| \leq 6$ and then give an algorithm that finds a set of up to 12 facets that contains $\mathcal{T}$.

▶ **Definition 8.** *A* covering set *is a set of open hemispheres $S \subseteq \bar{H}$ such that the union of all the hemispheres in S covers the entire unit sphere.*

For a two-dimensional example of a covering set, which covers the unit circle, see Figure 4.

▶ **Lemma 9.** *For each covering set S, $\bar{H}(\mathcal{T}) \subseteq S$.*

Proof. Let $S \subseteq \bar{H}$ be a covering set. Assume, for the sake of contradiction that there exists some facet $F_i \in \mathcal{T}$ for which $\bar{h}_i \notin S$. By Lemma 7, a facet F_i is a *valid top facet* if and only if there exists a depth 1 cell ξ in $\mathcal{A}(\mathcal{C})$ which is covered by $\bar{h}_i$ (and only by $\bar{h}_i$). We know that each point in the unit sphere is covered by some hemisphere $h \in S$, therefore some hemisphere $\bar{h}_j \in S$ covers a point in ξ. By the definition of a cell, if some point in the cell is contained in $\bar{h}_j$ then the entire cell is contained in $\bar{h}_j$. We also assumed that $\bar{h}_i$ covers ξ, which means that ξ is of depth at least two, in contradiction to F_i being a valid top facet. ◀

Our next step is to give an algorithm that finds, in linear time, a set of 12 open hemispheres in $\bar{H}$ whose union covers the entire unit sphere.

3 Finding a covering set

We begin by showing that a covering set of size 6 exists, which also implies that every polyhedron has at most 6 valid top facets. As we do so using a proof that does not directly imply an algorithm, we continue by showing how our observations lead to an algorithm for finding a slightly larger covering set of size 12. In order to do so, we introduce a definition and prove the following claims.

▶ **Definition 10.** *The half-plane of a hemisphere h with respect to the upper hemisphere is the central projection of the intersection of h and the upper hemisphere onto the plane $z = 1$ [13, Chapter 4]. See Figure 3 (and also Figure 5b for the analogous central projection of a semicircle on $y = 1$).*

Remark. The "half-plane" of the lower hemisphere with respect to the upper hemisphere is empty and the "half-plane" of the upper hemisphere with respect to the upper hemisphere is the entire plane $z = 1$.

▶ **Lemma 11.** *If the union of a set of half-planes, B, is the entire plane then there exists a set $S \subseteq B$ such that the union of the half-planes in S is the entire plane and $|S| = 3$. The set S can be computed in linear time in the size of B.*

Proof. For a given collection $X_1, X_2, \ldots, X_n$ of convex subsets of $\mathbb{R}^d$, where $n > d$, Helly's theorem [12] states that if the intersection of any $d+1$ objects in this collection is nonempty, then the entire collection has a nonempty intersection. The contrapositive of this theorem is that if the intersection of the entire collection is empty then there exists a subset of size $d + 1$ such that its intersection is empty.
In our case $d = 2$, so we have $\bigcup\limits_{b \in B} b = \mathbb{R}^2 \Rightarrow \overline{\bigcup\limits_{b \in B} b} = \emptyset \Rightarrow \bigcap\limits_{b \in B} \bar{b} = \emptyset$. By Helly's theorem, there exist three half-planes $b_1, b_2, b_3 \in B$ such that $\overline{b_1} \cap \overline{b_2} \cap \overline{b_3} = \emptyset$. By that we learn that $b_1 \cup b_2 \cup b_3 = \mathbb{R}^2$. This subset can be computed in linear time in the size of B with linear space using a version [19] of Megiddo's LP algorithm [18], or in linear expected time and constant space using Seidel's randomized incremental algorithm [20]. ◀

As an intermediate step, we describe how to find a covering set $S \subseteq \bar{H}$ of size three for the upper open hemisphere in linear time. If some hemisphere in $\bar{H}$ is the upper open hemisphere, return it and stop; we are done. Otherwise, transform each hemisphere in $\bar{H}$ into a half-plane on $z = 1$, as described earlier. Then, use the procedure we have just described to find three half-planes that cover the entire plane $z = 1$ in linear time. A covering set for any open hemisphere in $\mathcal{S}^2$ can be found similarly. In summary,

▶ **Observation 12.** *Any open hemisphere in $\mathcal{S}^2$ is covered by up to three hemispheres in $\bar{H}$. The covering hemispheres can be found in $O(|\bar{H}|)$.*

The observation leads to the following.

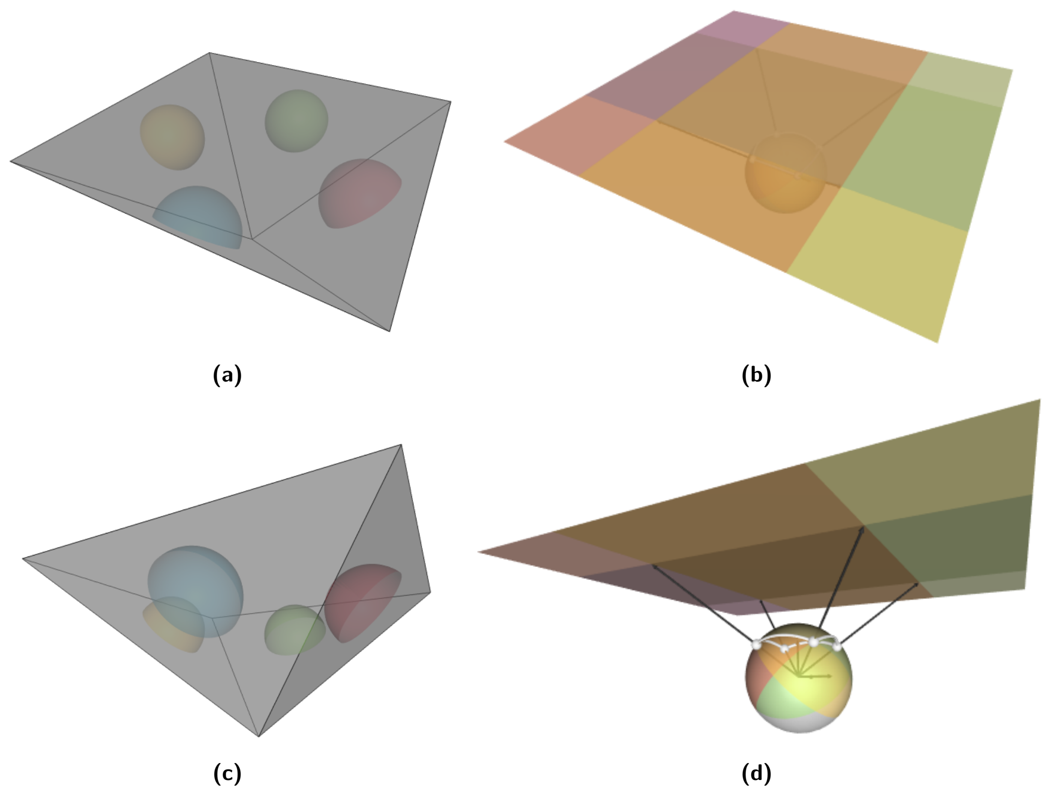


Figure 3 (a) Top view of the square pyramid depicted in Figure 2a, oriented upside down. Here the visible facet serves as the top facet. A hemisphere of potential removal (inwards) directions lies on each of the other four facets. (b) Top view of the central projection of the hemispheres onto the plane $z = 1$. The middle rectangle is the intersection of the projected half-planes. (c), (d) Bottom view of the scene.

▶ **Lemma 13.** *There exists a covering set of size six for $\mathcal{S}^2$.*

Proof. Let G be some great circle on $\mathcal{S}^2$ that does not contain a vertex of $\mathcal{A}(\mathcal{C})$ and is not contained in $\mathcal{C}$. The circle G intersects with various cells from $\mathcal{A}(\mathcal{C})$. Each such intersected cell is partially contained in the two open hemispheres defined by G. This means that any covering set of an open hemisphere defined by G, covers G as well (i.e., a cell of $\mathcal{A}(\mathcal{C})$ cannot be partially covered). By that and by Observation 12, the covering set of $\mathcal{S}^2$ is of size six at most. ◀

▶ **Theorem 14.** *The number of valid top facets for any polyhedron is at most six and this bound is tight.*

Proof. As we showed in Lemma 13, there exists a covering set of size six, and by Lemma 9, we know that the number of valid top facets in a polyhedron is bounded by the size of any covering set. Thus, the number of valid top facets is bounded by six. Notice that this is tight—all the six facets of a parallelepiped are valid top facets. ◀

We note that Lemma 13 uses a proof of existence that does not provide an algorithm to compute a covering set of size six. The problem with turning the proof into a linear-time algorithm is that we do not know how to find deterministically in linear time a great circle on $\mathcal{S}^2$ that avoids all the vertices of the arrangement. We therefore use a different strategy to find a covering set.

In order to find a covering set $S \subseteq \bar{H}$ for $\mathcal{S}^2$, we first choose four arbitrary open hemispheres (not from $\bar{H}$) that cover $\mathcal{S}^2$. This can be done by taking the set $\bar{H}(X)$, where X is the set of facets of some fixed tetrahedron, which covers $\mathcal{S}^2$ by Lemma 6. We then find a covering set of size three for each of the four hemispheres, as described above. Finally, by combining the resulting covering sets, we obtain a covering set for $\mathcal{S}^2$ of total size at most 12.

4 Algorithms

The algorithms for both the AllFAD and AllFSD problems are similar. We first find a covering set of constant size, as previously discussed. Then, for each facet in the covering set we solve SingleFAD or SingleFSD [13, Chapter 4], depending on whether we are solving AllFAD or AllFSD. Finally, for each valid top facet in the covering set we return either all the removal directions (if solving AllFAD) or a single removal direction (if solving AllFSD).

Algorithm 1 sketches both solutions. The difference between the AllFSD and AllFAD variants is in the function handleSingleFacet(P, F), which for a given polyhedron and a given facet, solves SingleFAD in $O(n \log n)$ time or SingleFSD in linear time [13, Chapter 4].

Algorithm 1 3D AllFAD/AllFSD

```
Input: $P$ a polyhedron
Output: A list of all the top facets and their removal direction(s)
1 $\mathcal{T} \leftarrow \emptyset$
2 $S \leftarrow$ coveringSet($P$)
3 for $\bar{h}_i \in S$ do
4     $directions \leftarrow$ handleSingleFacet($P$,$F_i$)
5     if $directions.notEmpty$ then
6         $\mathcal{T}$.add($F_i$, $directions$)
7 return $\mathcal{T}$
```

We obtain the following:

▶ **Theorem 15.** *Given a polyhedron P with n facets as input, our algorithm for AllFSD outputs a list of all valid top facets and a single removal direction for each such facet in total $O(n)$ time.*

▶ **Theorem 16.** *Given a polyhedron P with n facets as input, our algorithm for AllFAD outputs a list of all valid top facets and all valid removal directions for each such facet in total $O(n \log n)$ time.*

When solving the AllFAD variant, one may wish to run the AllFSD algorithm first to determine in linear-time whether the polyhedron is castable with a single-part mold.

Illustration

We now illustrate the algorithm for the 2D analog of the problem, where the object to be cast is a polygon R. Each possible removal direction is now represented by a point on the unit circle, rather than a point on the unit sphere $\mathcal{S}^2$. That is, instead of considering the arrangement of great circles on $\mathcal{S}^2$ ($\mathcal{A}(\mathcal{C})$ above) we now have an arrangement on the unit circle that is induced by a set of semicircles. Each semicircle is defined by an edge of R.

Figure 4a shows an input polygon R, along with the outer semicircles of its edges, which correspond to the set $\bar{H}$. Figure 4b shows the overlay of the outer semicircles on the unit circle, which results in an arrangement. The first stage of the algorithm consists of finding a covering set for the unit circle (line 2), which is done using linear programming techniques as described earlier. Figure 4c shows a covering set that is obtained from the outer semicircles. Observe that the cell with a depth of 1 in the arrangement is covered by the semicircle of edge F_i, which is in the covering set.

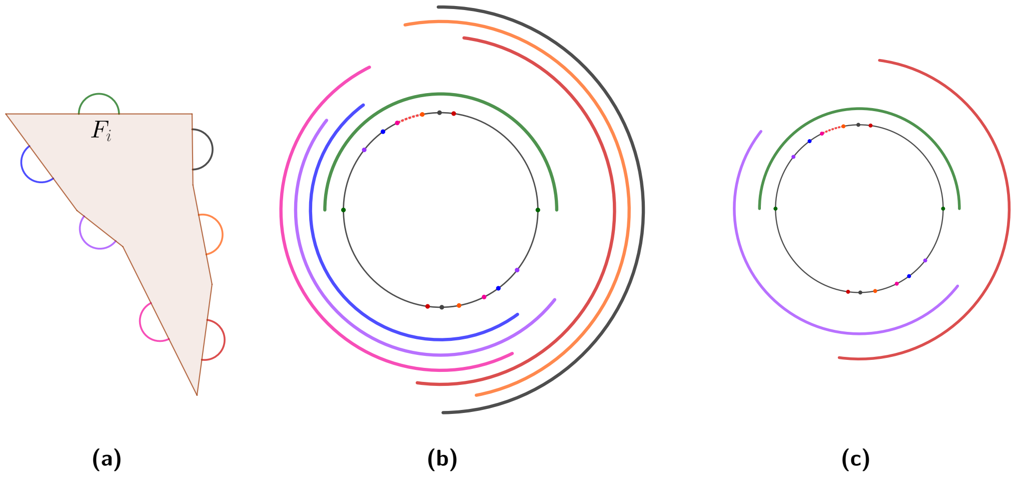


Figure 4 *Finding a covering set.* (a) The outer semicircle for each edge of R. (b) The arrangement of semicircles for R (innermost circle with breakpoints). There is one cell with a depth of 1, which is shown using a colored striped arc. The outer semicircles are shown around the arrangement for clarity. (c) A covering set consisting of three semicircles.

In the next stage of the algorithm, we apply the function HANDLESINGLEEDGE(R, F) (which is the 2D analog of HANDLESINGLEFACET(P, F) in 3D) on each edge that corresponds to a semicircle in the covering set. Here we illustrate the procedure for F_i; for full details, see [13, Chapter 4]. Given a fixed candidate top edge F_i, the goal is to find the set of valid removal directions for it. As per Lemma 3, a valid removal direction must lie in the outer semicircle of F_i and the inner semicircles of all others edges, which are shown in Figure 5a. That is, for each inner semicircle of a non-candidate edge, we are interested only in the directions with a positive y component, i.e., the *upward* directions. (Note that F_i is already oriented as the top edge. If that is not the case, we rotate the input polygon accordingly.) Each such direction may be projected onto the line $y = 1$. For example, Figure 5b shows this projection for all the upward directions in the inner semicircle of edge F_j, resulting in a ray. The range of valid directions, as constrained by edge F_j, now corresponds to the resulting ray. We therefore perform the projection for all non-candidate edges, obtaining a set of rays, as shown in Figure 5c. By finding the intersection of all the rays, we find the range of valid removal directions for the candidate top edge F_i, and we are done. If we are interested in a single removal direction, i.e., when solving ALLFSD, then it suffices to find a point contained in the intersection.

We remark that for 2D the algorithm can be simpler than illustrated. For example, Asberg et al. [4] solve the 2D analog of ALLFAD in linear time and constant space (in

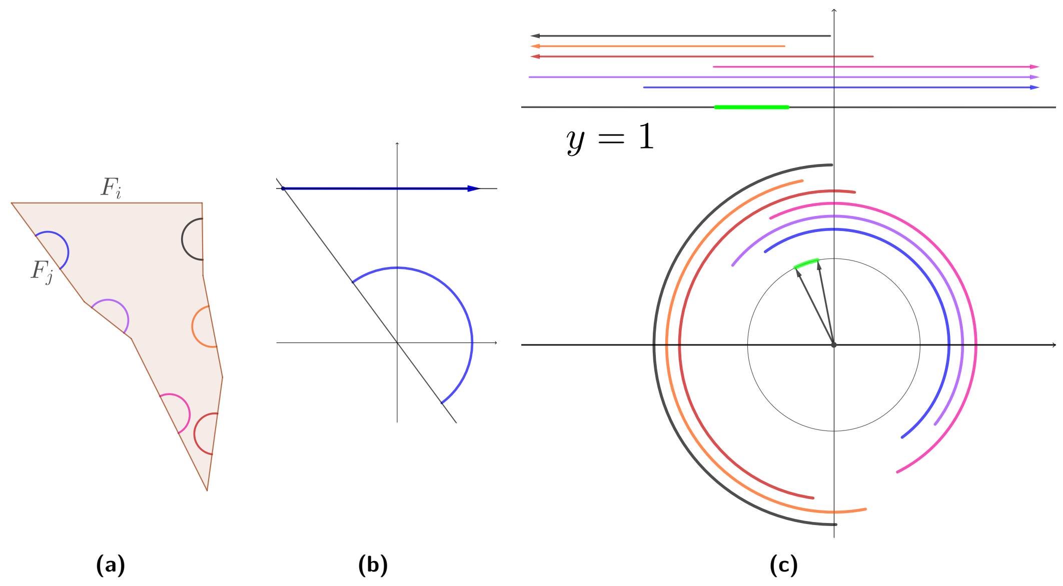


Figure 5 *Running* HANDLESINGLEEDGE(R, F_i). (a) The inner semicircle for each edge of R that is not the candidate top edge F_i. (b) For each edge other than F_i, we project its inner semicircle onto the line $y = 1$. The result of the projection is a ray representing the valid removal directions for the edge. (c) We find the common intersection of the projected rays. The intersection (green) represents the range of valid directions for the candidate top edge F_i.

addition to the input) using similar techniques. We stick to the steps analogous to those performed by the 3D algorithm only for explanatory purposes.

5 Lower bound

In this section we show an $\Omega(n \log n)$ lower bound for SINGLEFAD. We establish the lower bound in the algebraic computation tree model [5] using a reduction from sorting. The lower bound proves the optimality of both the existing SINGLEFAD algorithm [13, Chapter 4] and our new algorithm for ALLFAD in this model.

Let $X = \{x_1, \ldots, x_n\}$ be n positive real numbers, which we wish to sort. We describe a polyhedron P such that the convex polygon Q representing all the removal directions of

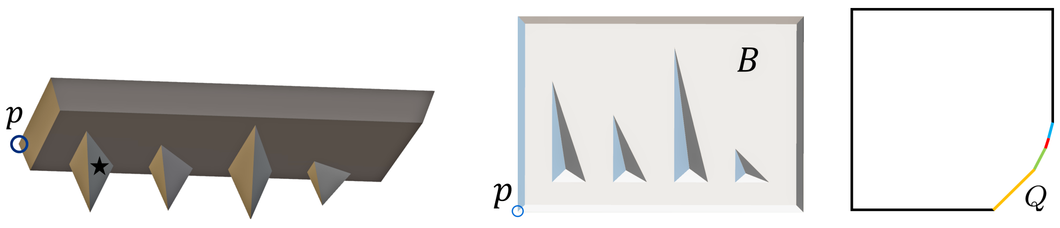


Figure 6 Left and middle: A construction of P for $X = \{3, 2, 4, 1\}$ viewed from the side (left) and from below (middle). The tetrahedra $T_1, \ldots, T_4$ appear from left to right and their darkest facets are the F_i's (the facet marked by a star is F_1). Right: The polygon Q representing the removal directions for P, which is obtained after running SINGLEFAD on P and its top facet. The numbers in X are represented in Q in a sorted manner as the slopes of the four colored edges, which appear in ascending order from the bottom up.

its top facet will contain the numbers of X as the slopes of its edges. We can then read off X in sorted order from the edges of Q. We begin with P as a rectangular frustum whose two bases, which are rectangular facets, are parallel to the plane $z = 0$; see Figure 6. Let B denote the bottom base of P, which is smaller than the top base. For a facet F of P, let $g(F)$ denote the half-plane obtained by the central projection of F's inner closed hemisphere, $h(F)$, onto the plane $z = 1$ (as in Definition 10)[2] and let $\ell(F)$ denote the line that defines $g(F)$. We call $g(F)$ the *corresponding half-plane* of F and $\ell(F)$ the *corresponding line* of F. Other than the two bases of P, each facet F in P will have the property that $g(F)$ contains the origin and $\ell(F)$ is tangent to the unit circle. For such a facet F, we can find its inner normal $\nu(F)$ such that we attain a desired line $\ell(F)$ that is tangent to the unit circle as follows: Take the normal vector v of $\ell(F)$ that points to the origin and tilt it upwards about $\ell(F)$ by $\pi/4$ radians, which results in $\nu(F)$.

We now describe the remaining facets of P. The corresponding lines of the four non-base facets of P are $x = -1, x = 1, y = -1, y = 1$, respectively. For each $x_i \in X$, P has a corresponding facet F_i such that $\ell(F_i)$ has the slope x_i. To add F_i to P, we define a tetrahedron T_i whose top facet fully overlaps with B, i.e., T_i is attached to B from below. The remaining three facets of T_i are F_i and two facets whose corresponding lines are $x = -1$ and $y = 1$, respectively. The T_i's are arranged in a row along B.

After applying a SingleFAD algorithm on the top base of P, the output convex polygon Q is the common intersection of the corresponding half-planes of all of P's facets except the top base (where each point in the common intersection corresponds to a removal direction). The corresponding half-planes of the facets $F_1, \ldots, F_n$ contribute to a contiguous set of edges along Q that are ordered by their slopes, which are $x_1, \ldots, x_n$. Finally, P has $3n + 6$ facets and can be constructed in linear time. Therefore, we obtain the following:

▶ **Theorem 17.** *There is an $\Omega(n \log n)$ lower bound for SingleFAD and AllFAD in the algebraic computation tree model [5].*[3]

▶ **Corollary 18.** *Our algorithms for AllFSD and AllFAD are optimal.*

A natural question is whether we can obtain even more efficient algorithms for special classes of polyhdera. In the next section we show that this is indeed possible for convex polyhedra. We now consider another restricted class of polyhedra and show that the lower bound still holds for it.

A polyhedron $\mathcal{P}$ is called *star-shaped* if it contains a point p such that, for any other point q in $\mathcal{P}$, the line segment $\overline{pq}$ lies in $\mathcal{P}$, i.e., p can see all of $\mathcal{P}$'s interior. Such a point p exists if the intersection of the half-spaces defined by the facets of $\mathcal{P}$ (facing into $\mathcal{P}$) is not empty. We may modify the construction of P above so that P is star-shaped. For the point p that can see of all P, we take the top left vertex (in the xy-plane) of the top facet, which is parallel to B; see Figure 6. The point p is contained in the half-spaces corresponding to all facets of P except for facets of the tetrahedra $T_1, \ldots, T_n$ that have the corresponding lines $x = -1$ and $y = 1$. We may tilt such facets so that their corresponding lines become $x = -c$ and $y = c$ respectively, where c is sufficiently large so that p can fully see the tilted facets. This change has no effect on the possible removal directions, since the frustum part of P

[2] $g(F)$ is a half-plane on $z = 1$ containing all the directions $\vec{d}$ with a positive z component such that $\vec{d} \cdot \nu(F) \geq 0$

[3] The theorem holds if the valid removal directions for a facet are to be returned as an explicit polygon, as is done by our algorithm. One may also elect to return the set of removal directions as the intersection of (non-sorted) half-planes, in which case the $\Omega(n \log n)$ lower bound does not hold.

remains the same, and so it still contains the two facets with the corresponding lines $x = -1$ and $y = 1$. Therefore, Q remains the same and we may read off X in sorted order from its edges as before.

6 Casting convex polyhedra

In this section we show how to solve the AllFAD problem for a *convex* polyhedron more efficiently than for an arbitrary polyhedron, namely, in $O(n)$ time. We start with some simple observations.

▶ **Observation 19.** *Let Q be a convex polygon in the plane whose edges are given in cyclic order $\mathcal{E}$, say clockwise. Given a direction $\vec{d}$ in the plane, all the edges of Q whose inner-facing normals have a non-negative scalar product with $\vec{d}$ form a consecutive subchain of $\mathcal{E}$.*

Let π be a plane in R^3 that is parallel to a direction $\vec{d}$, intersects some convex polyhedron, P, but does not intersect any of its vertices. Let Q be the convex polygon, which is the non-empty intersection of the convex polyhedron P with π. For each F_i that intersects with π we denote by e_i the edge $F_i \cap \pi$ of Q. Let $\nu(e_i)$ denote the inner facing normal of e_i in Q.

▶ **Lemma 20.** *If* $\operatorname{sign}(\nu(F_i) \cdot \vec{d}) \neq 0$*, then* $\operatorname{sign}(\nu(e_i) \cdot \vec{d}) = \operatorname{sign}(\nu(F_i) \cdot \vec{d})$.

Proof. Let D be a vector in direction $\vec{d}$ and length ε for some small ε. Then $\operatorname{sign}(\nu(F_i) \cdot \vec{d})$ indicates whether the direction $\vec{d}$ points into or out of P when it starts from a point on F_i [13, Chapter 4], i.e., if $\operatorname{sign}(\nu(F_i) \cdot \vec{d}) > 0$ ($\operatorname{sign}(\nu(F_i) \cdot \vec{d}) < 0$) for each point $p \in F_i$, then $p + D$ is in (outside) the polyhedron.

After the intersection with π, each point on e_i must still fulfill these conditions, since for each point $p \in e_i$, $p + D$ is in π and therefore it is in the polygon Q if and only if it is in the original polyhedron. ◀

We say that two facets of the input polyhedron P are neighbors if their closures intersect at an edge of the polyhedron. Denote by $\mathcal{M}_i$ the set of neighbors of the facet F_i, by m_i the cardinality of this set, and by J_i the index set of the facets in $\mathcal{M}_i$. The efficient AllFAD algorithm for convex polyhedra stems from the observation that it suffices to consider only the neighbors of F_i in order to determine if F_i is a valid top facet, which we formalize next.

▶ **Lemma 21.** *For a convex polyhedron, the pair $(F_i, \vec{d})$ is a valid top facet and a corresponding valid removal direction if and only if (i) $\vec{d} \cdot \nu(F_i) < 0$, and (ii) $\forall j \in J_i,\ \ \vec{d} \cdot \nu(F_j) \geq 0$.*

Proof. The only difference between Lemma 3 and the current lemma is the set of facets on which Condition (ii) is tested: here it is only on the neighboring facets of F_i. We argue that if the condition holds for the neighboring facets it holds for all facets other than F_i. Assume, for a contradiction that it holds for all neighboring facets, and there is a non-neighboring facet $F_k, k \neq i$ for which it does not hold. Let s be a segment connecting a point inside F_i and a point inside F_k. Let π be a plane containing s and parallel to $\vec{d}$. If π intersects a vertex of F_i then we move s (and π) slightly parallel to itself such that it does not cross any vertex of F_i. Let $Q := P \cap \pi$, and as above let $e_t := F_t \cap \pi$ for every facet F_t of P that intersects π. Let e' and e'' be the two neighboring edges to e_i in Q — notice that e' and e'' are the intersection of π with two neighboring facets of F_i. By Lemma 20 the scalar product of $\vec{d}$ with both $\nu(e')$ and $\nu(e'')$ is non-negative and with both $\nu(e_i)$ and $\nu(e_k)$ is negative. However, these form an intertwining cyclic subsequence of edges of Q: e', e_i, e'', e_k, in contradiction with Observation 19. This means that the inner-facing normal of e_k has non-negative scalar product with $\vec{d}$ and the same holds for the inner-facing normal of F_k. ◀

▶ **Lemma 22.** *Given a convex polyhedron P and a specific top facet F_i of P, it is possible to find* all *the removal directions in $O(m_i)$ time.*

Proof. Without loss of generality let's assume that F_i is horizontal (parallel to the xy-plane) and $\nu(F_i)$ points vertically downwards. We wish to find all the removal directions for which the conditions of Lemma 21 hold. This can be done in $O(m_i \log m_i)$ time by finding the half-plane $g(F_j)$ for each facet F_j in $\mathcal{M}_i$ and intersecting all of the resulting half-planes (see [13, Chapter 4]). We wish to reduce the running time of this procedure to $O(m_i)$ time. Luckily, these half-planes are sorted by their slope since the slope of the half-plane created by F_j on $z = 1$ is equal to the slope of the edge on F_i created by F_j (again, $z = 1$ and F_i are parallel). The edges of a convex polygon are ordered by their slope. The intersection of a given set of half-planes which are sorted according to the slope of their bounding lines, can be computed in $O(m_i)$ time [17]. ◀

The algorithm proceeds by fixing a facet F_i, computing its edge-neighboring facets and computing the set of allowable directions using Lemma 22. We repeat the procedure for each facet of P. Notice that m_i is in fact the number of edges on the boundary of F_i. The overall cost of $O(m_i)$ time over all candidate top facets F_i is $O(n)$ time by Euler's formula [3, Chapter 13]. In summary,

▶ **Theorem 23.** *Given a convex polyhedron P with n facets, we can solve the* ALLFAD *problem for P in time $O(n)$.*

We present one more result concerning the relation between the castability of a polyhedron P and the castability of its convex hull $CH(P)$. We show that every valid pair (see Definition 1) of P, induces a corresponding valid pair for $CH(P)$. This means that the set of valid pairs of the convex hull of P is a super-set of the valid pairs of P.

▶ **Proposition 24.** *Given a polyhedron P, one of its valid top facets F_i, and a removal direction $\vec{d}$, P's convex hull Q, can be removed through $CH(F_i)$ (the convex hull of F_i) with direction $\vec{d}$.*

Proof. Assume, for the sake of a contradiction, that Q cannot be removed in direction $\vec{d}$ through $CH(F_i)$. This means that for some point $q \in Q$ one of the conditions of Observation 2 does not hold. Since Q is convex, the open ray that emanates from q in direction $\vec{d}$ cannot intersect two facets of Q. This means that any point in Q that fulfills the first condition must fulfill the second condition as well. Thus, the first condition must not hold for q. Let v be a ray in direction $-\vec{d}$, and L be the locus of all the points (not necessarily in Q) for which the first condition holds. Then $L = CH(F_i) \oplus v$, where $\oplus$ is the Minkowski sum operator [1]. Every point in P satisfies the first condition of Observation 2. Therefore $P \subseteq L$ and since L is convex, $Q \subseteq L$, which is a contradiction. ◀

A similar relationship between a polyhedron and its convex hull was shown for castability with a two-part mold [8] and for feasibility of stereolithography [4].

7 Implementation

In this section we provide implementation details and experimental results. As CGAL is our implementation host, we start with some general details about this open-source library. We continue with a description of the implementation in the plane and proceed with a detailed description of the implementation in three-dimensional space. We conclude with experimental results.

7.1 CGAL

CGAL (Computational Geometry Algorithms Library) is an open-source software project that provides easy access to efficient and reliable implementations of geometric algorithms in the form of a C++ library [23]. CGAL has evolved through the years and now represents the state of the art in applied computational geometry software. The software modules of CGAL rigorously adhere to the generic programming paradigm—a discipline that consists of the gradual lifting of concrete algorithms abstracting over details, while retaining the algorithm semantics and efficiency. The software modules of CGAL also follow the exact geometric-computation paradigm, which amounts to ensuring that errors in predicate evaluations do not occur; it guarantees the robustness of the applied algorithms. As a result, the software of CGAL is stable despite the use of (inexact) floating point arithmetic, found in standard hardware, it is complete as it handles all degenerate cases, which typically are in abundance, and it is efficient—all at the same time.

One salient piece of CGAL consists of the geometric kernels [11]. A geometric kernel consists of types of constant-size non-modifiable geometric primitive objects, e.g., points, lines, triangles, and circles and operations on objects of these types. Another distinguishable piece, referred to as the "Support Library," consists of non-geometric facilities, such as number-type support. The rest of the library offers a collection of geometric data structures and algorithms such as 2D arrangements. These data structures and algorithms are parameterized by types referred to as *traits*; traits types define the interface between the data structures and algorithms and the primitives they use. In many cases (such as ours) a kernel can be used as a traits class.

7.2 Two-dimensional casting

As part of this work, a new package was added to CGAL called *2D Movable Separability of Sets* [21], which consists of free functions that compute various casting algorithms in the plane. The package was introduced with the release of version 4.13. The namespace of the functions in this package, omitted hereafter for clarity, is `CGAL::Set_movable_separability_2` ↪ `::Single_mold_translational_casting`. The package provides a function called `top_edges()` that accepts a simple closed polygon P and determines whether it is castable with a single-part mold. This package provides two additional functions, namely,[4] `pullout_directions()` and `is_pullout_direction()`. The former accepts a simple closed polygon P and an edge e of the polygon P; it determines whether e is a valid top edge of P, and if so, it computes the range of valid removal directions of e. The latter is overloaded with two versions: The first version accepts a simple closed polygon P and a direction d; it determines whether d is a valid removal direction of some top edge of P. The other version accepts, in addition, an edge e of the polygon P; it determines whether d is a removal direction of e. Overloads of each of the functions above that accept (i) an additional argument that indicates the orientation of the input polygon, (ii) an additional traits argument (see Section 7.1), (iii) both, are also provided by the package.

[4] Notice that what we refer to as *removal* direction in the paper is called `pullout_direction` in our software packages.

7.3 Three-dimensional casting

The implementation of the algorithms in the plane was mainly used as a proof of concept, and encouraged us to start implementing the algorithms in three-dimensional space. We have introduced a new package called *3D Movable Separability of Sets*, which consists of free functions that compute various casting algorithms in 3D. This package is available in a private branch that has not been exposed yet as part of an official release of CGAL. (The exposition of new features into CGAL requires rigorous testing, documentation, and examples nowadays, which are still under development.) The *2D Arrangements* package of CGAL [25] can be used to construct, maintain, alter, and display 2D arrangements. This package has been part of CGAL ever since the first version of CGAL was released over two decades ago [14]. It supports 2D arrangements in the plane. Recently, this package has been extended to support 2D arrangements embedded in curved surfaces, and especially in the sphere. We internally use this extension in our implementation.

We have implemented two function templates that accept a simple closed polyhedron P and determine whether P is castable with a single-part mold; the function templates have identical signatures and they reside in the following namespace:
`CGAL::Set_movable_separability_3::Single_mold_translational_casting`.

```
template <typename Polyhedron,  typename NamedParameters, typename OutputIterator>
OutputIterator
top_facets(const Polyhedron& P, const NamedParameters& np, OutputIterator oi);
```

If the polyhedron is castable, the functions compute the set of valid top facets. One function computes one removal direction in 3D for each valid top facet (solving AllFSD); the other function computes the closed region of removal directions in 3D for each valid top facet (solving AllFAD). Recall, that a polyhedron in 3D may have up to six valid top facets. The input polyhedron must satisfy two conditions as follows. First, it has to be simple. Essentially, a simple polyhedron is topologically equivalent to a ball the facets of which are simple polygons. Second, any consecutive three vertices in the counterclockwise sequence of vertices along the boundaries of every facet cannot be colinear. When the function is called (and instantiated) the template parameter `Polyhedron` must be substituted with a type that represents a polyhedron. CGAL provides two alternatives for such types, namely, an instance of the `CGAL::Polyhedron_3<>` class template and an instance of the `CGAL`
↪ `::Surface_mesh<>` class template. The latter is a model of the `MutableFaceGraph` and `FaceListGraph` concepts, which refine corresponding concepts of BGL (Boost Graph Library).[5] Each of these concepts requires, among others, the provision of the nested types `vertex_descriptor`, `edge_descriptor`, and `face_descriptor`, which are types used to describe a vertex, a halfedge, and a face, respectively. The parameter `np` is used to pass additional parameters to the procedure.[6] An example of such a parameter is a flag that determines whether the input polyhedron is convex. This information can be used to expedite the computation; see Section 6. The output iterator `oi` points at a container that holds the result when the function returns. Dereferencing this iterator yields an object the type of which is `std::pair`.

- The first element in the pair identifies a valid top facet. Its exact representation depends on the representation of the polyhedron. For example, if the polyhedron is represented by

[5]See `https://www.boost.org/libs/graph/doc/IncidenceGraph.html`.
[6]Using *named parameters* allows users to provide parameters in any order, a technique used, e.g., in BGL; see `https://www.boost.org/doc/libs/1_76_0/libs/graph/doc/bgl_named_params.html`.

an instance of the class template `CGAL::Surface_mesh<>`, the top facet is represented by its index, the type of which is convertible to `face_descriptor`.

- The second element is either one valid removal three-dimensional direction or a closed spherical patch of valid removal three-dimensional directions bounded by arcs of great circles. The patch is represented as a sequence of the extreme directions of the patch of type `Kernel::Direction_3`. It is a face of an arrangement induced by arcs of great circles embedded on the sphere.

7.4 Experimental results

We ran experiments on an *Intel Core i7-4770* CPU clocked at 3.4 GHz with 16 GB of RAM, a high-class desktop CPU, which would be a good fit for CAD applications. We have conducted two rounds of experiments. In the first round we examined different polyhedra of small complexities, but with different numbers of valid top facets. In the second round we used a procedural method to gradually increase the complexity of a polyhedron with a certain pattern; we used the resulting polyhedra to measure the performance of our code.

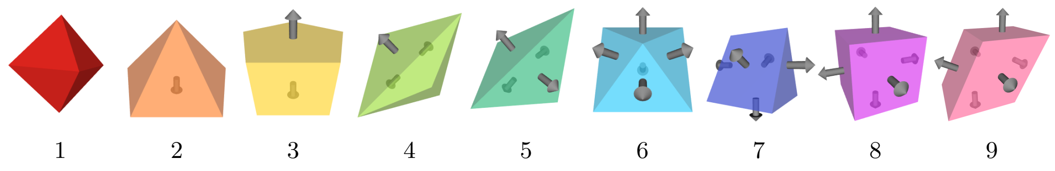
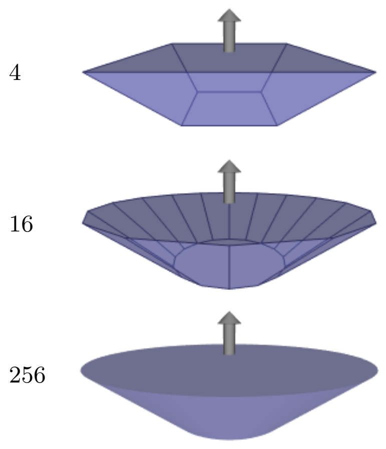


Figure 7 Various polyhedra and some of their valid removal directions.

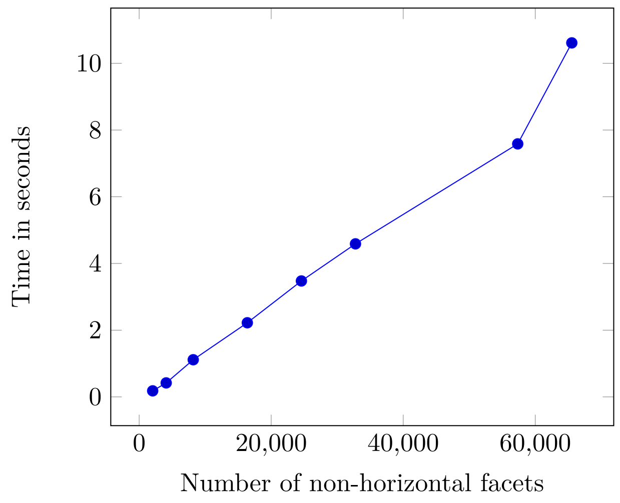
Table 1 Various polyhedra and the number of valid top faces.

No.	Polyhedron	# facets	# top facets	# removal directions per top facet					
				1	2	3	4	5	6
1	Octahedron	8	0						
2	Pentagonal pyramid	6	1	∞					
3	Pentagonal prism	7	2	1	1				
4	Tetrahedron plus	6	3	∞	∞	∞			
5	Tetrahedron	4	4	∞	∞	∞	∞		
6	Square pyramid	5	5	∞	∞	∞	∞	∞	
7	Triangular prism	5	5	1	1	∞	∞	∞	
8	Cube	6	6	1	1	1	1	1	1
9	Parallelepiped	6	6	1	1	1	1	1	1

Figure 7 depicts eight polyhedra that we have examined. Polyhedron (4), **Tetrahedron plus**, is constructed from a tetrahedron by taking one of its four triangular facets, subdividing the facet into three triangular facets using a new vertex, and slightly lifting the new vertex in the direction of the outward normal. Observe that in some cases, a valid top facet has exactly one valid removal direction. In other words, the removal directions are isolated in the infinite space of directions. Naturally, finding these particular directions would fail if the computation is naively carried out without taking special measures to deal with round-off errors. Our code successfully handled these degenerate cases. Table 1 lists the polyhedra, their total number of facets, and the number of top facets. The number of each polyhedron in the table matches the number in the figure.



(a)



(b)

Figure 8 (a) Truncated pyramids. The numbers on the left indicate the number of the corresponding non-horizontal facets. (b) A graph of the time it takes to compute one valid removal direction of truncated pyramids of growing complexities.

We developed a benchmark that finds a valid removal direction for the single valid top facet of each polyhedron in a sequence of truncated pyramids with gradually growing complexities; see Figure 8a. In this benchmark we used the overloaded function `top_facets()` that solves the AllFSD problem, and we measured its performance. The plot depicted in Figure 8b shows the graph of the time it takes to compute a (single) valid removal direction as a function of the complexity of the input polyhedron. As expected the graph is approximately linear.

8 Conclusion and further work

We have described and implemented efficient solutions to determine the castability of a polyhedron with a single-part mold. Our algorithms are an order of magnitude faster than the best previously known algorithms for these problems. We showed that our algorithms are optimal in the algebraic computation tree model.

We outline several directions for further research:

(I) Our focus here was on separability with a single translation. Can one devise efficient algorithms in case we are allowed to remove the polyhedron from its cast using arbitrary motion? This problem is already interesting for separating a planar polygon from its planar cast. The latter can be resolved by considering the motion planning problem for a robot (the polygon) among obstacles (the mold). This approach however would yield a near-quartic time solution [16]. Thus, the goal here is to take advantage of the special structure of the problem to obtain a more efficient solution.

(II) There are many interesting problems when the mold is made of two or more parts (see, e.g., [2]), and we hope that our novel observations here may open the door to more efficient algorithms for these more complicated casting problems. Furthermore, we believe that our algorithms and their implementation can be used for special types of two-part molds, where we split the mold into two by a cutting plane. Then, one can try many candidate cutting planes and for each one apply the efficient single-part mold algorithms on each of the two resulting object pieces.

References

1 Pankaj K. Agarwal, Eyal Flato, and Dan Halperin. Polygon decomposition for efficient construction of minkowski sums. *Comput. Geom.*, 21(1-2):39–61, 2002. doi:10.1016/S0925-7721(01)00041-4.

2 Hee-Kap Ahn, Mark de Berg, Prosenjit Bose, Siu-Wing Cheng, Dan Halperin, Jiří Matoušek, and Otfried Schwarzkopf. Separating an object from its cast. *Computer-Aided Design*, 34(8):547–559, 2002. doi:10.1016/S0010-4485(01)00119-1.

3 Martin Aigner and Günter M Ziegler. *Proofs from the Book, 4th Edition*, volume 274. Springer, 2010.

4 Boudewijn Asberg, Gregoria Blanco, Prosenjit Bose, Jesus Garcia-Lopez, Mark H. Overmars, Godfried T. Toussaint, Gordon T. Wilfong, and Binhai Zhu. Feasibility of design in stereolithography. *Algorithmica*, 19(1/2):61–83, 1997. doi:10.1007/PL00014421.

5 Michael Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the fifteenth Annual ACM Symposium on Theory of Computing*, pages 80–86, 1983. doi:10.1145/800061.808735.

6 Eric Berberich, Efi Fogel, Dan Halperin, Michael Kerber, and Ophir Setter. Arrangements on parametric surfaces II: concretizations and applications. *Mathematics in Computer Science*, 4(1):67–91, 2010. doi:10.1007/s11786-010-0043-4.

7 Eric Berberich, Efi Fogel, Dan Halperin, Kurt Mehlhorn, and Ron Wein. Arrangements on parametric surfaces I: general framework and infrastructure. *Mathematics in Computer Science*, 4(1):45–66, 2010. doi:10.1007/s11786-010-0042-5.

8 Prosenjit Bose, David Bremner, and Marc J. van Kreveld. Determining the castability of simple polyhedra. *Algorithmica*, 19(1/2):84–113, 1997. doi:10.1007/PL00014422.

9 Prosenjit Bose, Dan Halperin, and Shahar Shamai. On the separation of a polyhedron from its single-part mold. In *13th IEEE Conference on Automation Science and Engineering, CASE*, pages 61–66. IEEE, 2017. doi:10.1109/COASE.2017.8256076.

10 Prosenjit Bose and Godfried T. Toussaint. Geometric and computational aspects of manufacturing processes. *Comput. Graph.*, 18(4):487–497, 1994. doi:10.1016/0097-8493(94)90061-2.

11 Hervé Brönnimann, Andreas Fabri, Geert-Jan Giezeman, Susan Hert, Michael Hoffmann, Lutz Kettner, Sylvain Pion, and Stefan Schirra. 2D and 3D linear geometry kernel. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5.3 edition, 2023. URL: https://doc.cgal.org/5.5.3/Manual/packages.html#PkgKernel23.

12 Ludwig Danzer, Branko Grunbaum, and Victor Klee. Helly's theorem and its relatives. *Monatsh. Math*, 31:60–97, 1921.

13 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications, 3rd Edition*. Springer, 2008. URL: https://www.worldcat.org/oclc/227584184.

14 Efi Fogel, Dan Halperin, and Ron Wein. *CGAL Arrangements and Their Applications, A Step by Step Guide*. Springer, Berlin Heidelberg, Germany, 2012. doi:10.1007/978-3-642-17283-0.

15 Dan Halperin, Jean-Claude Latombe, and Randall H Wilson. A general framework for assembly planning: The motion space approach. *Algorithmica*, 26(3-4):577–601, 2000. doi:10.1007/s004539910025.

16 Dan Halperin and Micha Sharir. A near-quadratic algorithm for planning the motion of a polygon in a polygonal environment. *Discrete & Computational Geometry*, 16(2):121–134, 1996. doi:10.1007/BF02716803.

17 Mark Keil. A simple algorithm for determining the envelope of a set of lines. *Information Processing Letters*, 39(3):121–124, 1991. doi:http://dx.doi.org/10.1016/0020-0190(91)90106-R.

18 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, jan 1984. `doi:10.1145/2422.322418`.

19 Carolyn Haibt Norton, Serge A. Plotkin, and Éva Tardos. Using separation algorithms in fixed dimension. *Journal of Algorithms*, 13(1):79–98, 1992. `doi:10.1016/0196-6774(92)90006-X`.

20 Raimund Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6(3):423–434, 1991. `doi:10.1007/BF02574699`.

21 Shahar Shamai and Efi Fogel. 2D movable separability of sets. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5.3 edition, 2023. URL: `https://doc.cgal.org/5.5.3/Manual/packages.html#PkgSetMovableSeparability2`.

22 Jack Snoeyink and Jorge Stolfi. Objects that cannot be taken apart with two hands. *Discrete & Computational Geometry*, 12:367–384, 1994. `doi:10.1007/BF02574386`.

23 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5.3 edition, 2023. URL: `https://doc.cgal.org/5.5.3/Manual/packages.html`.

24 Godfried Toussaint. Movable separability of sets. In *Computational Geometry*. North-Holland Publishing Company, 1985. `doi:10.1016/B978-0-444-87806-9.50018-9`.

25 Ron Wein, Eric Berberich, Efi Fogel, Dan Halperin, Michael Hemmer, Oren Salzman, and Baruch Zukerman. 2D arrangements. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5.3 edition, 2023. URL: `https://doc.cgal.org/5.5.3/Manual/packages.html#PkgArrangementOnSurface2`.