Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 Keller Hall
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 04-001

Approximating Contact-Area of Supports in Layered Manufacturing

Ivaylo Ilinkin, Ravi Janardan, Michiel Smid, Eric Johnson, Paul Castillo, Jörg Schwerdt

January 5, 2004

Approximating Contact-Area of Supports in Layered Manufacturing

Ivaylo Ilinkin* Ravi Janardan* Michiel Smid† Eric Johnson* Paul Castillo‡
Jörg Schwerdt§

Abstract

Layered Manufacturing is a technology that allows physical prototypes of three-dimensional models to be built directly from their digital representation, as a stack of two-dimensional layers. A key design problem here is the choice of a suitable direction in which the digital model should be oriented and built so as to minimize the area of contact between the prototype and temporary support structures that are generated during the build. Devising an efficient algorithm for computing such a direction has remained a difficult problem for quite some time. In this paper, a suite of efficient and practical heuristics is presented for approximating the minimum contact-area. Also given is a technique for evaluating the quality of the approximation of any heuristic, which does not require knowledge of the (unknown and hard-to-compute) optimal solution; instead, it provides an indirect upper bound on the quality of the approximation via two relatively easy-to-compute quantities. The algorithms are based on various techniques from computational geometry, such as ray-shooting, convex hulls, boolean operations on polygons, and spherical arrangements, and have been implemented and tested. Experimental results on a wide range of real-world models show that the heuristics perform quite well in practice.

1 Introduction

Layered Manufacturing (LM) is a fast-growing technology with significant impact on the efficiency of the design process in a broad range of industries [Jac92, CLL03]. LM offers a flexible and cost-effective alternative to traditional methods used in the design phase of physical prototypes. Current LM technology produces high-quality prototypes with added color in a matter of hours and at low cost. The prototypes can be inspected for flaws and if necessary the design can be modified and the process repeated until the final design has reached the desired quality.

Stereolithography is a widely-used LM process. In essence, the Stereolithography Apparatus (SLA) consists of a vat of light-sensitive liquid resin, a platform, and a laser (see Figure 1). The

*Department of Computer Science & Engineering, University of Minnesota, Minneapolis, MN 55455, U.S.A. {ilinkin,janardan}@cs.umn.edu, erj@visi.com. Research supported, in part, by NSF grant CCR–9712226. This effort is also sponsored, in part, by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAD19-01-2-0014, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

†School of Computer Science, Carleton University, Ottawa, Canada, K1S 5B6 michiel@scs.carleton.ca. Research supported by NSERC

‡Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94551 castillo17@llnl.gov

§Softwarebüro Bubel GmbH, 66459 Kirkel, Germany jschwerdt@swbb.de

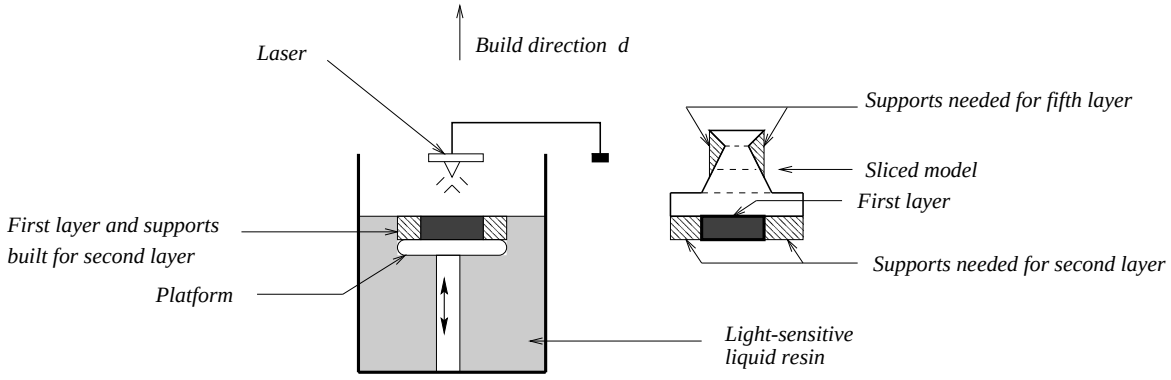


Figure 1: The Stereolithography Apparatus, along with the sliced digital model and support structures.

---

input to the process (and to virtually all other LM processes) is a surface triangulation of the digital model in the industry-standard STL format. The model is oriented suitably and sliced into horizontal 2D layers, which are then sent over a network to the SLA. The laser traces out the contour of each layer (a polygon) and then scans the interior in a zig-zag pattern. The exposure to the laser causes the scanned portion of the liquid to harden and form the physical layer. The platform is then lowered by an amount equal to the layer thickness (typically a few thousandths of an inch) and the next layer is then built on top of the previous one; thus, the 3D prototype is realized eventually as a vertical stack of 2D layers. Ideally each new layer should rest completely on top of the previous one, so that the prototype is self-supporting during the build phase. Unfortunately, the complex shape of real-world prototypes often prevents them from being self-supporting (in any orientation). Therefore, during a pre-processing step, the model is analyzed and additional structures called *supports* are created and merged with the description of the model. Supports are built simultaneously with the prototype and later removed in a post-processing step.

The choice of orientation can impact critically the efficiency of the build process and the surface quality of the physical prototype. Several competing criteria need to be addressed when choosing an optimal orientation. For example, an optimal orientation that minimizes the amount of support structures will translate into faster build times. Similarly, an orientation that minimizes the contact-area (the amount of contact between the final prototype and the support structures) would minimize the portions of the prototype that are affected during the post-processing stage and help minimize damage to the surface of the prototype during support removal.

The problem of finding a suitable orientation can be translated into purely geometric terms and this has led to considerable amount of research in recent years. Asberg *et al.* [ABB+97] (see also [Bos95]), describe efficient algorithms to decide if a given model can be built without supports using Stereolithography. Majhi *et al.* [MJSG99] give algorithms to minimize the volume of supports and contact-area for convex polyhedra. (See also [Maj98].) Schwerdt *et al.* show how to choose a build direction that protects prescribed facets from being damaged by supports [SSJ+00]. Agarwal and Desikan [AD00] have given an efficient algorithm to approximate a build direction which minimizes the contact-area for a convex polyhedron. Johnson [Joh99] shows how to compute support descriptions efficiently for a given build direction.

Unfortunately, very few results are available for the problem of optimizing support requirements for non-convex polyhedra. Majhi *et al.* [MJS+99] give support optimization algorithms for non-convex polygons in the case of 2D Stereolithography. An exact algorithm to minimize contact-area for polyhedral models is presented in [Sch01], but its high running time precludes its use in practice; specifically, the running time is $O(n^6 q(n))$, where $q(n)$ is the time to solve a certain non-linear optimization problem on the unit-sphere. Allen and Dutta [AD95] give heuristics for minimizing support contact-area for non-convex polyhedra.

Contributions: In this paper, we make further progress on the contact-area problem for polyhedral models. Specifically, we provide a suite of efficient and practical heuristics for approximating support contact-area (Sections 4–6). These methods are based on various techniques from computational geometry, such as ray-shooting, convex hulls, boolean operations on polygons, spherical algorithms etc., and make use of the CGAL [CGA] and LEDA [MN99] libraries. We also give a method for evaluating the quality of the approximation of any heuristic, which does not require knowledge of the (unknown and hard-to-compute) optimal solution; instead, it provides an indirect upper bound on the quality of the approximation via relatively easy-to-compute quantities (Section 3). Finally, we present an extensive set of experimental results on real-world STL models that show that the heuristics perform quite well in practice (Section 7).

2 Preliminaries

We denote by $\mathcal{P}$ the polyhedron of interest and by n the number of facets in $\mathcal{P}$. We assume that the facets of $\mathcal{P}$ are triangles and that its boundary is represented in some standard form, such as, for instance, a doubly-connected edge list [dBvKOS97] or a winged-edge structure [Bau75]. (If necessary, such a representation can be computed easily from the standard STL representation of $\mathcal{P}$ [MS99].) Let $\mathbf{d}$ be a given *build direction* (a unit-vector); we assume, w.l.o.g., that $\mathbf{d}$ coincides with the positive z-direction.

Let f be any facet of $\mathcal{P}$. We classify f, w.r.t. the given build direction $\mathbf{d}$, as a *front facet*, a *back facet*, or a *parallel facet* of $\mathcal{P}$ depending on whether the angle between the build direction $\mathbf{d}$ and the outward unit-normal, $\mathbf{n}_f$, of f is less than, greater than, or equal to 90°, respectively.

We now formalize the notion of supports. A facet of a polyhedron will need to be supported if and only if the angle between its outer normal and the build direction of the polyhedron is greater than 90°. This implies that the back facets of $\mathcal{P}$ will need to be supported. For concreteness, consider a back facet f of $\mathcal{P}$. The *support polyhedron* for f is the closure of the set of all points $p \in \mathbb{R}^3$ such that p is not in the interior of $\mathcal{P}$ and the ray shot from p in direction $\mathbf{d}$ first enters $\mathcal{P}$ through f. Informally, the support polyhedron of f is bounded from above by f, on the sides by vertical facets that "drop down" from the edges of f, and from below by the platform on which $\mathcal{P}$ rests and/or portions of front facets of $\mathcal{P}$. (If $\mathcal{P}$ is convex, then it is bounded from below by only the platform.) The *supports* of $\mathcal{P}$ w.r.t. a build direction is the collection of support polyhedra for the back facets.

The *support contact-area* for $\mathcal{P}$ is the total surface area of $\mathcal{P}$ that is in contact with supports. It includes the area of all the back facets of $\mathcal{P}$ and the areas of those portions of front facets and parallel facets that are in contact with supports. Note that for a convex polyhedron, the support structures are relatively simple, in that only back facets are in contact with supports and every point on a back facet is in contact with supports. Furthermore, the support structures extend all

the way down to the platform. However, for a general polyhedron, the situation is more complex: In addition to back facets, some front and parallel facets can also be in contact with supports, and the latter two types of facets may be only partially in contact. In addition, supports need not extend all the way down to the platform, but may instead terminate on other parts of the polyhedron. This is illustrated in Figure 1 for the supports for the fifth layer. (The figure is shown in 2D, for simplicity.) It is this complexity of the support structures that makes the support optimization problem that we consider so challenging.

3 An upper bound on the approximation

Ideally, we would like to find a direction $\mathbf{d}^*$ that minimizes the contact-area of $\mathcal{P}$. Unfortunately, the structure of non-convex polyhedra presents significant challenges in finding an optimal solution efficiently. Unlike convex objects, for which only the back facets are in contact with supports, non-convex objects can have portions of front and parallel facets in contact with supports, as well. Furthermore, a small change in the build orientation can result in a significantly different footprint of the support structures, possibly affecting other facets that previously were not in contact with supports. These factors make it difficult to design an efficient and practical optimal algorithm, therefore, an efficient heuristic that provides some guarantee about the quality of its approximation can be quite useful in practice.

We now develop a criterion that gives a measure of the quality of approximation for a given heuristic. Specifically, the ratio test developed below gives an indication of how close the approximation is to the optimal contact-area. Let $CA(\mathbf{d})$ denote the contact-area of $\mathcal{P}$ for a given build direction, $\mathbf{d}$, and let $\hat{\mathbf{d}}$ be the direction computed by a heuristic. We show how to obtain an upper bound on the ratio $CA(\hat{\mathbf{d}})/CA(\mathbf{d}^*)$ via two relatively easy-to-compute quantities. Let $BFA(\mathbf{d})$ be the total area of the back facets w.r.t. $\mathbf{d}$ and let $\mathbf{d}'$ be a direction that minimizes the total area of back facets.

Notice that $BFA(\mathbf{d}^*) \leq CA(\mathbf{d}^*)$, since $CA(\mathbf{d}^*)$ includes possible contact-area on front and parallel facets, and $BFA(\mathbf{d}') \leq BFA(\mathbf{d}^*)$, by definition of $\mathbf{d}'$. Therefore,

$$\frac{CA(\hat{\mathbf{d}})}{CA(\mathbf{d}^*)} \leq \frac{CA(\hat{\mathbf{d}})}{BFA(\mathbf{d}^*)} \leq \frac{CA(\hat{\mathbf{d}})}{BFA(\mathbf{d}')}$$

The above result allows us to upper-bound the quality of approximation for a set of candidate directions, relative to the (unknown) optimal solution, and to choose from these the best direction $\hat{\mathbf{d}}$. Notice that $BFA(\mathbf{d}')$ needs to be computed only once, and therefore, the quality test will depend mainly on the efficiency of computing the contact-area for a given direction. In the next section we present two algorithms that compute the contact-area, but differ in their accuracy and efficiency.

4 Computing contact-area on front facets for a fixed direction

In this section, we describe an exact algorithm and a heuristic for computing the contact-area on front facets for a fixed build direction $\mathbf{d}$. The exact algorithm is simple, but too slow to be of practical use. However, we will use it to verify the accuracy of the heuristic, which we employ in our approximation scheme.

4.1 An exact algorithm

W.l.o.g. assume that $\mathcal{P}$ rests on the xy-plane and that the build direction $\mathbf{d}$ coincides with the positive $\mathbf{z}$ direction. Let f be a fixed front facet and let b be any back facet. We project f and b to the xy-plane and compute the intersection of their projections (i.e., triangles), which yields a convex polygon, $C(b)$ (Figure 2(a)). If $C(b) \neq \emptyset$, then let p be any point in it, say the centroid. If the pre-images, p_f and p_b, of p on f and b, respectively, are such that p_b is above p_f in direction $\mathbf{d}$, then p_f is in contact with supports. This implies that the pre-image $C_f(b)$ of $C(b)$ on f is in contact with supports. This follows since no facet of $\mathcal{P}$ pierces another, so there cannot be another point p' in $C(b)$, whose pre-images on f and b are in the opposite order from those of p. (Note that it need not be the case that the cylinder bounded by $C_f(b)$ and by $C_b(b)$—the pre-image of $C(b)$ on b—is a support cylinder, since b, or parts thereof, need not be immediately above f; there could be parts of other back facets in between.) We can compute the footprint of supports on the front facet f by taking the union of the pre-images $C_f(b)$ that are found to be in contact with supports, for all back facets b. (In our implementation, we used the functions provided by LEDA [MN99] to perform the union and intersection operations.)

The most expensive part of this algorithm turns out to be the union step in the computation of the footprints. Note that the algorithm simply projects all back facets down to the xy-plane, without regard to any intervening facets. Thus, the complexity of the union of the polygons $C(b)$ on a single front facet, f, can be $\Theta(n^2)$ in the worst case, and $\Theta(n^3)$ over all front facets. (An example of this is a configuration of $\Theta(n)$ front facets stacked on top of each other and $\Theta(n)$ back facets above them that overlap in the form of a trellis.) The total time to compute the union is $O(n^2 \log n)$ in the worst case for any front facet [dBvKOS97], hence $O(n^3 \log n)$ over all front facets. The storage requirement is $O(n^2)$; since the algorithm works on a facet-by-facet basis, the space can be reused.

With some additional effort, the running time can be improved to $O(n^3)$, as follows: For each front facet, f, we take the lines that support the sides of the triangles resulting from projecting all the back facets onto f. We compute the arrangement of these lines and then determine the cells in this arrangement that are covered by the projection of at least one back facet. The union of these cells gives the footprint of the supports on f. The covering information for the cells can be computed incrementally, by doing a depth-first traversal of the dual graph of the arrangement and maintaining a counter that is incremented or decremented depending on whether or not the next cell in the traversal is covered by the triangle whose side is crossed to reach the cell. The time per front facet reduces to $O(n^2)$ and the claimed bound follows. (Note that this method, unlike the previous one, takes quadratic time per front facet, regardless of the geometric complexity of the footprint.)

We note also that a theoretically faster algorithm, running in $O(n^2 \log n)$ time and $O(n^2)$ space, is possible. This algorithm uses *cylindrical decomposition* [Mul93] and respects intervening facets during projection. An output-sensitive algorithm with running time $O(n \log^2 n + V \log n)$, where V is the complexity of the decomposition, is given in [SH02]. Unfortunately, the algorithms described in this section and in [SH02] are extremely sensitive to degenerate input configurations, and therefore, reliable implementations require the use of exact arithmetic. As the experimental results in Table 1 and in [SH02] indicate, the use of exact arithmetic introduces considerable overhead on the running time—our algorithm takes at least 3500 seconds for polyhedra with 2000 facets, while the algorithm in [SH02] takes about 400 seconds on a set of 150 triangles in space.

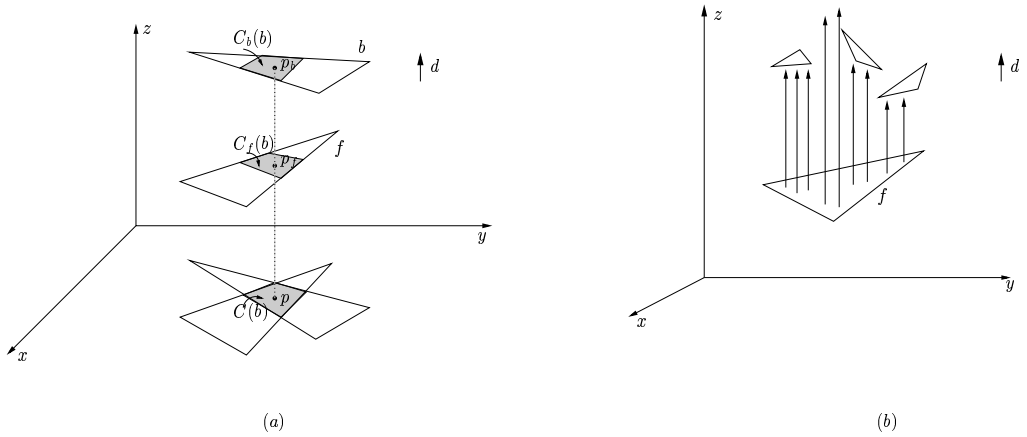


Figure 2: (a) Computing a patch that is in contact with supports in the exact algorithm; (b) Ray shooting for a front facet f in the heuristic.

Therefore, we use an exact algorithm merely for a one-time verification of the heuristic in Section 4.3, and we have chosen to implement the relatively simpler $O(n^3 \log n)$-time exact algorithm described above for this purpose.

4.2 A heuristic based on ray-shooting

In this section we describe a simple heuristic with a very fast rate of convergence, which makes it practical for real data sets. Consider a front facet f and let p be a point on f. The point p will be in contact with supports if and only if the ray originating at p in direction $\mathbf{d}$ intersects some other facet of $\mathcal{P}$. Thus, the main idea behind the heuristic is to pick a set of points in the interior of f and identify those points that will be in contact with supports by shooting rays in direction $\mathbf{d}$. Let H_f (resp. M_f) denote the set of rays, originating at points on f, that hit a facet of $\mathcal{P}$ (resp. miss all facets of $\mathcal{P}$). Then the area of f that is in contact with supports can be approximated as $(|H_f|/(|H_f| + |M_f|)) * area(f)$. As the density of the sample points is increased, the accuracy of the approximation improves (Figure $2(b)$).

The sample points are selected through a subdivision process. During the execution of the algorithm each facet is subdivided into a number of triangular patches (initially the entire face is the only patch). The centroids of the patches are selected as sample points and the results from the ray shooting are recorded. Next, each patch is subdivided into two triangles (for example, by connecting the midpoint of its longest side with the opposite vertex). The new patches are placed at the end of a queue of unprocessed patches, which guides the subdivision process in a breadth-first search fashion. This ensures that all facets are subdivided to the same depth.

Each iteration of the algorithm corresponds to a complete subdivision of the patches, and therefore, will process twice as many patches as the previous one. The algorithm terminates after a predefined number of iterations, controlled by the user, or when the change in contact-area is not significant. Currently, we use the convergence criterion $\delta = |CA_{i+1} - CA_i| < 0.01 * CA_i$, i.e. the algorithm terminates when the contact-area from iteration i to iteration $(i+1)$ (denoted by CA_i and CA_{i+1}, respectively) changes by less than 1%.

model (.stl) (#facets)	reduced #facets	algorithm	contact-area (front facets)	time (sec.)
bot_case (17642)	2000	$d = 10$	13646.9	82
		$\delta < 1\%$	13678.1	1
		exact	13642.3	4693
carcasse (22876)	2000	$d = 10$	58.44	66
		$\delta < 1\%$	57.06	1
		exact	60.20	3505
mj (2832)	2000	$d = 10$	1.68	60
		$\delta < 1\%$	1.65	1
		exact	1.68	3540
top_case (16692)	2000	$d = 10$	10239.1	110
		$\delta < 1\%$	10494.5	1
		exact	10268.1	11592
triad (11352)	2000	$d = 10$	0.33	68
		$\delta < 1\%$	0.32	1
		exact	0.33	3888

Table 1: Comparison between the exact algorithm and the heuristic to compute, for a given direction, the contact-area on front facets of the indicated models. Large models have been decimated to 2000 facets due to the slow speed of the exact algorithm. The z-direction is chosen as a build direction. The models used here and in the rest of the tables are illustrated in Table 4.

At each iteration i the algorithm has to process $2^i * n$ patches, where n is the number of facets in the model. For each patch the ray shooting takes $O(n)$ time to decide whether the ray hits any facet of $\mathcal{P}$. So, the overall time per iteration is $O(2^i * n^2)$. For a user-specified number, d, of iterations, the overall running time is $O(2^d * n^2)$.

4.3 Experimental results

Table 1 provides a comparison between the exact algorithm and the heuristic. The heuristic was run with two different terminating criteria: (i) terminating after ten iterations, denoted as "$d = 10$"; (ii) terminating based on the convergence criterion described in Section 4.2, denoted as "$\delta < 1\%$". In the latter case, we also imposed an upper limit of ten iterations, so that the computation did not become prohibitively expensive. The comparison tests were run on decimated versions of real-world STL models due to the slow performance of the exact algorithm from Section 4.1 that we implemented for purposes of comparison. (We used the *Decimator*$^{\text{TM}}$ [Dec] software from Raindrop Geomagic, Inc. to reduce the number of facets in the models, as indicated, while preserving their general topology. However, our final experiments in Table 4 were done on original models, not decimated ones.) As can be seen, the heuristic provides nearly the same answer as the exact algorithm, but in a fraction of the time. All experiments were done on a SunBlade 100 machine with 512 MB of main memory and a 500 MHz processor. Programs were written in C++ with floating-point arithmetic, and use CGAL [CGA] and LEDA [MN99].

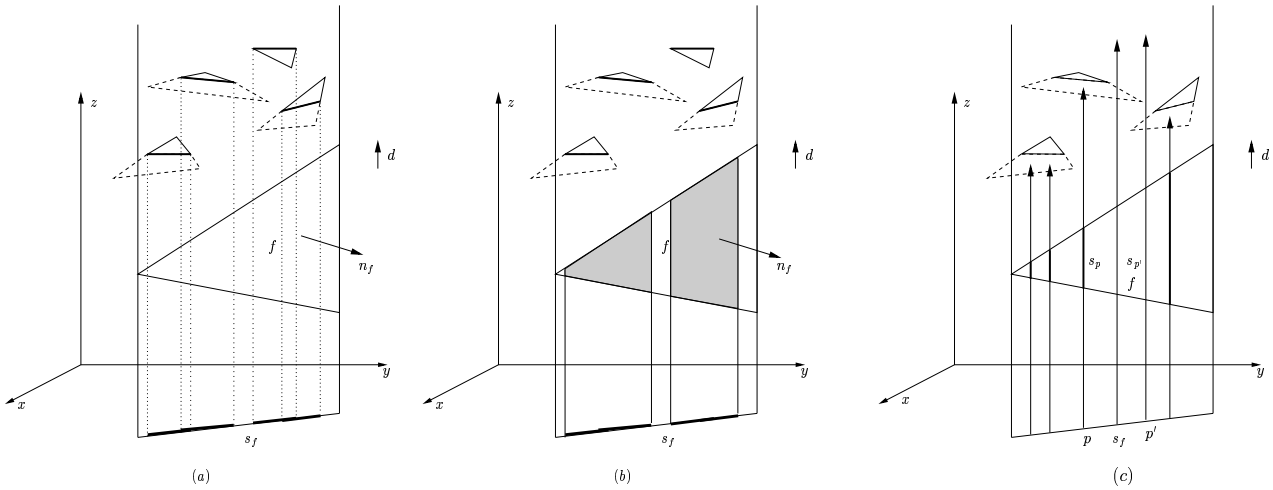


Figure 3: Computing contact-area on parallel facets: (a) (exact algorithm) identifying the segments that require supports; (b) (exact algorithm) identifying the portions of f that are in contact with supports; (c) (heuristic) segment s_p is in contact with supports but $s_{p'}$ is not.

---

5 Handling parallel facets

The previous algorithms consider only the contact-area on front and back facets. In order to get an overall estimate of the contact-area, we need to also consider the portions of parallel facets that are in contact with supports. In Section 5.1 we give an algorithm for computing the exact contact-area on parallel facets, and in Section 5.2 we give an efficient heuristic, analogous to the one in Section 4.2.

5.1 Exact algorithm

Let f be a parallel facet and let V_f be the vertical strip, which is in the supporting plane of f and contains f exactly. We may assume w.l.o.g. that no vertex of f is in the interior of V_f. (Each bounding line of V_f contains at least one vertex of f. If there is a vertex in the interior of V_f, we draw a vertical line through it and split f into two facets that each satisfy the assumption.) Let s_f be the projection of f on the xy-plane. (Notice that s_f is just a line segment.)

Consider the back facets of $\mathcal{P}$ that either pierce V_f above f, or touch V_f above f and are in the same halfspace of V_f as the outer unit-normal, $\mathbf{n}_f$, of f. (These are the back facets whose supports are potentially in contact with f when $\mathcal{P}$ is built in direction $\mathbf{d}$.) The intersections of these back facets with V_f is a set, A, of line segments (Figure 3(a)).

Let A' be the set of segments that correspond to the projections of the segments in A on the xy-plane. Clearly, all the segments in A' lie on s_f and can be merged efficiently, so that no two of the resulting segments overlap. For each merged segment s we erect a vertical strip V_s and find its overlap area with f (Figure 3(b)). The sum of the areas of overlap for all strips V_s gives the overall contact-area on f.

For each facet f the size of A is $O(n)$. Merging the segments in A' can be done efficiently in $O(n \log n)$ time by pre-sorting them on their first endpoint. For each strip V_s the overlap area can

model (.stl) (#facets)	reduced #facets	algorithm	contact-area (parallel facets)	time (sec.)
bot_case (17642)	2000	$d = 10$	701.17	1
		$\delta < 1\%$	701.17	0
		exact	701.17	6
carcasse (22876)	2000	$d = 10$	20.46	18
		$\delta < 1\%$	20.51	1
		exact	22.82	368
mj (2832)	2000	$d = 10$	1.88	5
		$\delta < 1\%$	1.88	0
		exact	1.80	132
top_case (16692)	2000	$d = 10$	4855.04	43
		$\delta < 1\%$	4867.35	2
		exact	4289.45	403
triad (11352)	2000	$d = 10$	0.065	2
		$\delta < 1\%$	0.065	0
		exact	0.062	56

Table 2: Comparison between the exact algorithm and the heuristic to compute, for a given direction, the contact-area on parallel facets of the models in Table 1. The z-direction is chosen as a build direction.

---

be found in constant time. Therefore, the algorithm takes $O(n \log n)$ time per parallel facet, or $O(n^2 \log n)$ time for all parallel facets.

5.2 Heuristic

Let f be a parallel facet and let s_f be the projection of f on the xy-plane. (Notice that s_f is just a line segment.) Let p be any point on s_f and let s_p be the segment obtained by intersecting f with the ray originating at p in direction $\mathbf{d}$. The segment s_p will be in contact with supports if and only if the supporting line of s_p intersects properly a facet of $\mathcal{P}$ at a point above s_p (Figure 3(c)).

Let S be a set of sample points on s_f. Let H_f (resp. M_f) be the set of segments, s_p, that are (resp. are not) in contact with supports. If $length(H_f)$ and $length(M_f)$ denote the sum of the lengths of the segments in H_f and M_f, respectively, we can approximate the contact-area on f as $(length(H_f)/(length(H_f) + length(M_f))) * area(f)$. As the number of sample points is increased, the accuracy of the approximation also increases.

The sample points are selected through a subdivision process similar to the one described in Section 4.2. The main difference is that the patches are line segments, and not triangles (initially, each facet f is represented by the patch s_f). The sample points are the midpoints of the corresponding patches and each patch is subdivided into two equal length segments at its midpoint. The terminating criterion is the same as the one described in Section 4.2. Table 2 summarizes the execution of the exact algorithm and the heuristic on decimated models.

6 Minimizing back facet area

The efficiency of computing the approximation bound in Section 3 depends critically on the efficiency of finding a direction $\mathbf{d}'$ that minimizes the back facet area. In this section we describe an algorithm, based on arrangements of great circles on the unit-sphere, that computes $\mathbf{d}'$.

6.1 Preliminaries

Let $\mathcal{S}^2$ denote the unit-sphere of directions. We map each facet, f, to a point on $\mathcal{S}^2$ corresponding to the unit vector, $\mathbf{n}_f$, normal to the supporting plane of f. Let $\mathcal{C}_f$ be the set of points on $\mathcal{S}^2$ that are at distance $\pi/2$ from $\mathbf{n}_f$, i.e. $\mathcal{C}_f$ is a great circle on $\mathcal{S}^2$. (Note that several facets of $\mathcal{P}$ can correspond to a single great circle.) $\mathcal{C}_f$ defines two open hemispheres: $\mathcal{H}_f^+$ with pole $\mathbf{n}_f$, and $\mathcal{H}_f^-$ with pole $-\mathbf{n}_f$. Given a build direction $\mathbf{d}$ the facet f will be a *back* facet, and therefore will require supports, if and only if $\mathbf{d} \in \mathcal{H}_f^-$. Similarly, f will be a *front* facet, requiring no supports, if and only if $\mathbf{d} \in \mathcal{H}_f^+$. Finally, f will be a *parallel* facet, requiring no supports, if and only if $\mathbf{d} \in \mathcal{C}_f$. (Although a front or parallel facet requires no support, it could be in contact with supports required by back facets, as seen previously.)

Consider the arrangement, $\mathcal{A}$, of great circles corresponding to the facets of $\mathcal{P}$. $\mathcal{A}$ decomposes $\mathcal{S}^2$ into three types of *elements*: (*i*) *cells*, which are (open) regions of intersection of the hemispheres defined by the great circles, (*ii*) *arc edges*, which are (open) portions of great circles and determine the boundaries of the cells in $\mathcal{A}$, and (*iii*) *vertices*, which are intersections of great circles and are the endpoints of arc edges.

Lemma 6.1 *The elements of $\mathcal{A}$ define regions on $\mathcal{S}^2$ that correspond to sets of directions for which the back facet area is constant.* ∎

Lemma 6.2 *The build direction $\mathbf{d}'$ minimizing the back facet area corresponds to a vertex in the $\mathcal{A}$.*

Proof We show that the back facet area corresponding to any point in a cell or arc edge is never more than the back facet area corresponding to the vertices of the cell. This implies that it is sufficient to consider only the vertices of $\mathcal{A}$ in order to find $\mathbf{d}'$.

Let c be a cell in $\mathcal{A}$. For any element q of $\mathcal{A}$, let $BFA(q)$ be the back facet area associated with any direction in q. By Lemma 6.1 $BFA(q)$ is well-defined and a constant. Notice that the boundary of c represent transitions onto great circles, which correspond to front and/or back facets becoming parallel facets. Therefore, the set of back facets corresponding to any point on the boundary of c is either the same as or a proper subset of the set of back facets corresponding to any point in the interior of c. This implies that the back facet area cannot increase. Therefore, $BFA(e) \leq BFA(c)$ for any edge e on the boundary of c.

Let e be an edge in $\mathcal{A}$ and let u be one of the vertices in $\mathcal{A}$ that is adjacent to e along the supporting great circle $\mathcal{C}(e)$ of e. The vertex u represents a transition, along the arc edge e, onto a great circle other than $\mathcal{C}(e)$ along the arc edge e. Thus, a front and/or a back facet becomes parallel. Arguing as before, $BFA(u) \leq BFA(e)$.

The above discussion shows that the back facet area corresponding to any point within a cell in $\mathcal{A}$ is never more than the back facet area corresponding to any point along the bounding edges of the cell. Furthermore, the latter is never more than the back facet area at the vertices in $\mathcal{A}$

adjacent to the edge. Thus in order to identify the direction $\mathbf{d}'$ that minimizes the back facet area, it is sufficient to examine only the directions corresponding to the vertices in $\mathcal{A}$. ∎

6.2 The algorithm

Lemma 6.2 shows that to find the direction, $\mathbf{d}'$, that minimizes the back facet area it is sufficient to consider only the directions on $\mathcal{S}^2$ that correspond to the vertices of $\mathcal{A}$. This immediately suggests an algorithm for finding $\mathbf{d}'$:

- (*pre-processing*) Compute the arrangement $\mathcal{A}$ of great circles on the unit sphere.
- (*initialization*) Let u be any vertex in $\mathcal{A}$. Identify the front, back, and parallel facets determined by the direction corresponding to u, and initialize the back facet area term to the total area of the back facets.
- (*update*) Walk along the vertices of the arrangement, by visiting adjacent vertices connected by an arc edge. Notice that a vertex in $\mathcal{A}$ is the intersection of great circles and each great circle describes a set of directions for which front and/or back facets become parallel. Therefore, during the transition from vertex u to vertex v, let $\Delta BFA(u)$ be the area of the parallel facets at u that become back facets at v, and let $\Delta BFA(v)$ be area of the parallel facets at v that were back facets at u. Then $BFA(\mathbf{d}_v) = BFA(\mathbf{d}_u) - \Delta BFA(u) + \Delta BFA(v)$.
- During the walk along the vertices of the arrangement we keep track of the vertex v for which the back facet area is minimized and report the direction, $\mathbf{d}_v$, corresponding to v.

The *pre-processing* step of the above algorithm takes $O(n^2)$ time and $O(n^2)$ space since the number of great circles in the arrangement is $O(n)$. At each vertex in the arrangement we spend time proportional to the degree of the vertex, and therefore the overall time during the *update* step of the algorithm is $O(n^2)$. Therefore, the algorithm takes $O(n^2)$ time and uses $O(n^2)$ space.

The space usage of the algorithm can be improved to $O(n)$ at the expense of increased running time to $O(n^2 \log n)$. The main idea is to walk along arc edges belonging to the same great circle. This allows us to focus on only a portion of the arrangement $\mathcal{A}$. Given a great circle $\mathcal{C}_f$ we compute its intersections with all the other great circles and sort the vertices of intersection in their circular order along $\mathcal{C}_f$. Next we pick an arbitrary vertex and initialize the back facet area term. Finally, we visit all the vertices along $\mathcal{C}_f$ and update the back facet area term following the rule described in the *update* step of the algorithm above. During the walk we keep track of the vertex, v, corresponding to the direction for which the back facet area is minimized. The optimal direction is identified after all great circles have been processed.

The running time per great circle is dominated by the time to sort $O(n)$ vertices of intersection in time $O(n \log n)$. The walk along a great circle spends $O(n)$ time for the initialization and constant time per vertex, or $O(n)$ in total. Over all great circles the running time is $O(n^2 \log n)$. Since we do not compute the whole arrangement, the space is $O(n)$ and this can be re-used.

Table 3 summarizes the results of the execution of the $O(n^2 \log n)$-time algorithm on non-decimated models.

model (.stl)	#facets	minimum area (back facets)	time (sec.)
bot_case	17642	14409.4	3996
carcasse	22876	35.98	6673
mj	2832	5.81	80
top_case	16692	7843.5	3374
triad1	11352	2.20	1554

Table 3: Performance of the algorithm for computing a direction that minimizes the area of the back facets.

---

7 Approximating the contact-area

In this section we present several heuristics for choosing a candidate build direction that approximates the optimal contact-area requirements. The quality of each heuristic is measured in terms of the ratio $CA(\hat{\mathbf{d}})/BFA(\mathbf{d}')$, where $\hat{\mathbf{d}}$ is the direction computed by the heuristic, and $\mathbf{d}'$ is the direction that minimizes the area of the back facets. As shown in Section 3 this ratio is an upper bound on $CA(\hat{\mathbf{d}})/CA(\mathbf{d}^*)$, where $\mathbf{d}^*$ is the direction that minimizes the overall contact-area.

We have implemented and tested the following choices for build direction:

- *min BFA direction* — direction that minimizes the back facet area. Since the overall contact-area includes the area of the back facets, it may be advantageous to choose a direction that results in low contact-area contribution from the back facets.

- *max PFA direction* — direction that maximizes the area of parallel facets. We consider the direction and its opposite, since both generate the same area of parallel facets, and take the better result. The intuition behind this heuristic is that parallel facets do not themselves require supports, and therefore, by maximizing the area of parallel facets the number of support structures could be reduced, which could lead to reduced amount of contact-area.

- *max PFC direction* — direction that maximizes the count of parallel facets. We consider the direction and its opposite, since both generate the same count of parallel facets, and take the better result. This is an alternative to the previous heuristic, but we try to maximize the number of facets that will not require direct supports, which could lead to a reduction in support structures, and therefore a reduction in the amount of contact-area.

- *PC direction* — direction that corresponds to the principal components of the object. Intuitively, this heuristic builds the object along one of three mutually perpendicular axes that capture the relative shape of the object. We consider each direction and its opposite and take the best result. The principal component directions were computed using MATLAB$^{\text{TM}}$ [MAT] software from The MathWorks, Inc.

- *Flat direction* — direction that is opposite to the outward unit-normal of a facet of the model. During the build phase it is often desirable to build the part such that it rests on one of its facets. In this case the facet must be contained in the boundary of the convex hull of the model (notice that a facet on the convex hull may contain several facets from the original model). We select the facet, f, on the convex hull which contains facets from the original

model that have the largest total area and use $-\mathbf{n}_f$ as the build direction, where $\mathbf{n}_f$ is the outward unit-normal of f.

- *Random directions* — directions chosen at random. This heuristic was included for comparison purposes. We chose a set of one hundred and twenty random directions, computed the contact-areas over each of these directions, took their mean, and then divided this by the minumum back facet area to arrive at the mean contact-area ratio.

Table 4 illustrates the models used for our experiment and summarizes the results. For each model the table shows the contact-area ratio computed for each heuristic and compares the ratio realized by the best heuristic with the ratio realized by the random heuristic. As seen in the column named "comparison", savings ranging from 9% to 83% are achieved on real-world (non-decimated) models. Note that even though we are comparing ratios this is equivalent to comparing the contact-areas themselves, since the denominators for both ratios are the same, namely the minimum back facet area.

8 Conclusion

We have presented a set of efficient and practical heuristics for computing approximately the contact-area of supports for polyhedral models in Layered Manufacturing. We have also shown how the quality of the approximation, w.r.t. the unknown and hard-to-compute optimal solution, can be upper-bounded as the ratio of two relatively easy-to-compute quantities. Our algorithms have been implemented and tested on a range of real-world models and have been shown to perform well in practice.

An interesting problem for further work is computing a build direction that approximates the volume of the support structures. Our technique for bounding the quality of the approximation does not appear to extend to this problem, so a different approach may be needed.

References

[ABB+97] B. Asberg, G. Blanco, P. Bose, J. Garcia-Lopez, M. Overmars, G. Toussaint, G. Wilfong, and B. Zhu. Feasibility of design in stereolithography. *Algorithmica*, 19:61–83, 1997.

[AD95] S. Allen and D. Dutta. Determination and evaluation of support structures in layered manufacturing. *Journal of Design and Manufacturing*, 5:153–162, 1995.

[AD00] P. Agarwal and P. Desikan. Approximation algorithms for layered manufacturing. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 528–537, 2000.

[Bau75] B. G. Baumgart. A polyhedron representation for computer vision. In *Proceedings of AFIPS National Computer Conference*, volume 44, pages 589–596. AFIPS Press, Alrington, Va., 1975.

[Bos95] P. Bose. *Geometric and computational aspects of manufacturing processes.* PhD thesis, School of Computer Science, McGill University, Montréal, Canada, 1995.

[CGA] Computational Geometry Algorithms Library (CGAL). `http://www.cgal.org`.

[CLL03] C. K. Chua, K. F. Leong, and C. S. Lim. *Rapid Prototyping: Principles and Applications*. World Scientific Publishing Co., Inc., 2003.

[dBvKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[Dec] Decimator^{TM}1.0, Raindrop Geomagic, Inc.,. `www.geomagic.com/products/decimate/`.

[Jac92] P. Jacobs. *Rapid Prototyping & Manufacturing: Fundamentals of StereoLithography*. McGraw-Hill, 1992.

[Joh99] E. Johnson. Support generation for three-dimensional layered manufacturing. Master's project report, Dept. of CS&E, Univ. of Minnesota, Minneapolis, MN, 1999.

[Maj98] J. Majhi. *Geometric methods in computer-aided design and manufacturing*. PhD thesis, Dept. of Computer Science & Engineering University of Minnesota, Minneapolis, MN, 1998.

[MAT] MATLAB^{TM}6.5, The Mathworks, Inc.. `www.mathworks.com/products/matlab/`.

[MJS+99] J. Majhi, R. Janardan, J. Schwerdt, M. Smid, and P. Gupta. Minimizing support structures and trapped area in two-dimensional layered manufacturing. *Computational Geometry: Theory & Applications*, 12:241–267, 1999.

[MJSG99] J. Majhi, R. Janardan, M. Smid, and P. Gupta. On some geometric optimization problems in layered manufacturing. *Computational Geometry: Theory & Applications*, 12:219–239, 1999.

[MN99] K. Mehlhorn and S. Naher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, U.K., 1999.

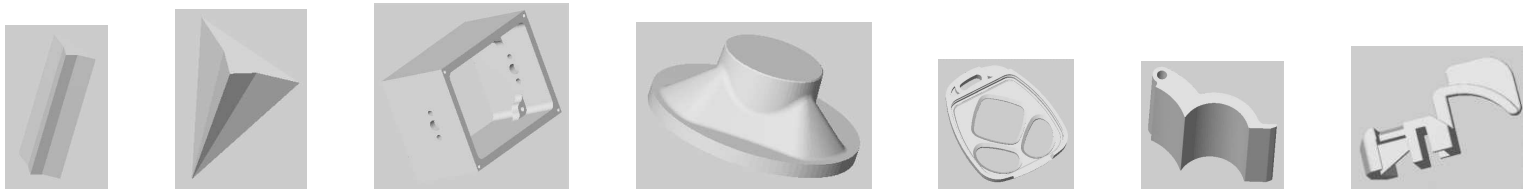
[MS99] S. McMains and C. Séquin. A coherent sweep plane slicer for layered manufacturing. In *Proceedings of the 5th ACM Symposium on Solid Modeling and Applications*, pages 285–295, 1999.

[Mul93] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1993.

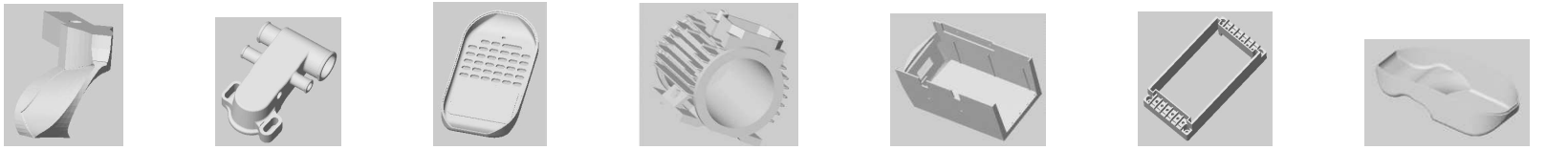
[Sch01] J. Schwerdt. *Entwurf von Optimierungsalgorithmen für geometrische Probleme im Bereich Rapid Prototyping und Manufacturing*. Ph.D. thesis, Department of Computer Science, University of Magdeburg, Magdeburg, Germany, 2001.

[SH02] H. Shaul and D. Halperin. Improved construction of vertical decompositions of 3D arrangements. In *Proceedings of the 18th Annual ACM Symposium on Computational Geometry*, pages 283–292, 2002.

[SSJ+00] J. Schwerdt, M. Smid, R. Janardan, E. Johnson, and J. Majhi. Protecting critical facets in layered manufacturing. *Computational Geometry: Theory & Applications*, 16:187–210, 2000.



prism1 pyramid1 stlbin2 ecc4 triad1 tod21 f0m27



mj 3857438 top_case carcasse cover-5 oldbasex bot_case

model (.stl)	#facets	max possible ratio	min BFA ratio	max PFA ratio	max PFC ratio	PC ratio	Flat ratio	random mean ratio	comparison (%)	time (sec.)
prism1	20	55.20	1.00 [1]	1.00 [1]	24.00 [4]	1.00 [1]	24.00 [4]	28.53 [6]	96	12
pyramid1	10	14.34	1.00 [1]	3.65 [3]	6.02 [4]	1.00 [1]	6.02 [4]	7.27 [6]	86	5
stlbin2	2761	15.76	2.85 [2]	2.57 [1]	10.02 [5]	8.51 [3]	10.02 [5]	9.58 [4]	73	180
ecc4	4994	4.89	1.18 [1]	1.18 [1]	1.37 [3]	1.92 [5]	1.80 [4]	2.65 [6]	55	1272
triad1	11352	2.89	1.87 [4]	2.13 [5]	2.13 [5]	1.43 [1]	1.43 [1]	1.74 [3]	18	3262
tod21	1128	7.19	1.05 [1]	1.05 [1]	3.87 [5]	3.81 [4]	1.05 [1]	4.31 [6]	76	124
f0m27	3730	4.29	2.40 [4]	2.33 [1]	2.33 [1]	2.39 [3]	3.26 [6]	2.69 [5]	13	319
mj	2832	5.32	2.18 [1]	2.39 [2]	2.39 [2]	2.56 [5]	2.39 [2]	3.00 [6]	27	197
3857438	12184	3.37	2.63 [6]	2.54 [3]	2.54 [3]	2.41 [2]	2.32 [1]	2.55 [5]	9	3458
top_case	16692	4.09	3.14 [5]	3.14 [5]	3.07 [4]	2.14 [2]	1.97 [1]	2.50 [3]	21	6800
carcasse	22876	6.23	3.77 [3]	3.47 [1]	4.19 [4]	4.38 [5]	3.47 [1]	4.89 [6]	29	12741
cover-5	906	5.71	3.92 [3]	3.92 [3]	3.92 [3]	3.10 [1]	4.80 [6]	3.79 [2]	18	30
oldbasex	3660	15.60	3.33 [2]	1.72 [1]	12.40 [5]	8.16 [3]	12.40 [5]	10.37 [4]	83	270
bot_case	17642	3.04	2.11 [4]	2.11 [4]	2.11 [4]	1.54 [2]	1.29 [1]	1.95 [3]	34	7291

Table 4: Performance of the heuristics for approximating the contact-area. The first two models were hand-generated; the remaining models are from Stratasys, Inc, a world-leader in the manufacture of LM machines (www.stratasys.com). All models are originals, not decimated ones. Running times exclude the time for computing the ratio for the random heuristic, which is for purposes of comparison only. The numbers in square brackets show the ranking of each heuristic. The column "comparison" shows the fraction of improvement of the ratio given by the top-ranked heuristic over the mean ratio given by the random-directions heuristic.