Proceedings of DETC00
2000 ASME Design Engineering Technical Conferences
September 10-13, 2000, Baltimore, Maryland, USA

DETC2000/DAC-14285

LAYERED MANUFACTURING OF THIN-WALLED PARTS

Sara McMains
Jordan Smith
Jianlin Wang
Carlo Séquin
Computer Science Department
University of California, Berkeley
Berkeley, California, 94720-1776
Email:sara | jordans | jianlin | sequin@cs.berkeley.edu

ABSTRACT

We describe a new algorithm we have developed for making partially hollow layered parts with thin, dense walls of approximately uniform thickness, for faster build times and reduced material usage. We have implemented our algorithm on a fused deposition modeling (FDM) machine, using separate build volumes for a loosely filled interior and a thin, solid, exterior wall. The build volumes are derived as simple boolean combinations of slice contours and their offsets. We make use of an efficient algorithm for computing the Voronoi diagram of a general polygon as part of the process of creating offset contours. Our algorithm guarantees that the surface of the final part will be dense while still allowing an efficient build.

INTRODUCTION

Designers who want to make prototypes of solid three-dimensional parts directly from CAD descriptions are increasingly turning to a class of technologies collectively referred to as layered manufacturing or solid freeform fabrication (SFF). These technologies include stereolithography (SLA), 3-D printing, fused deposition modeling (FDM), selective laser sintering (SLS), and laminated object manufacturing (LOM)(4). In all these processes, a triangulated boundary representation (b-rep) of the CAD model of the part in STL format (1) is sliced into horizontal, 2.5-D layers of uniform thickness. Each cross sectional layer is successively deposited, hardened, fused, or cut, depending on the particular process, and attached to the layer beneath it. (For technologies such as SLA and FDM, a sacrificial support structure must also be built to support overhanging geometry.) The stacked layers form the final part.

With most additive layered SFF processes, build time is roughly proportional to the solid volume of the final part. With FDM, it is proportional to the amount of material deposited (for the part and for supports). With SLS or SLA, it is proportional to the scan and dwell time of the laser solidifying the build material.

When making a model of a solid part with a low surface area to volume ratio, using a process such as (FDM), we can complete the build considerably faster if we don't fill the interior of the part densely. For a fairly sturdy final part, we can fill the interior with a loose cross-hatched pattern for support, with a solid wall several layers thick at the surface.

The QuickSlice 6.2 software (19) that currently ships with the Stratasys FDM machine includes a fast build option. The software identifies slices that are "hidden" by slices above and below, and for these it builds a dense shell consisting of three concentric "roads" inside the perimeter, and fills the interior (the hidden part) with a looser fill pattern, as shown in figure 1.

The drawback of this straight-forward approach is that if the slice intersects any part surfaces that approach horizontal, the software will just do a solid fill on the entire slice because the concentric outer roads might not entirely hide the loose fill pattern in adjacent layers. For large layers whose interiors would be almost entirely hidden, this is a waste of time and material. Experienced

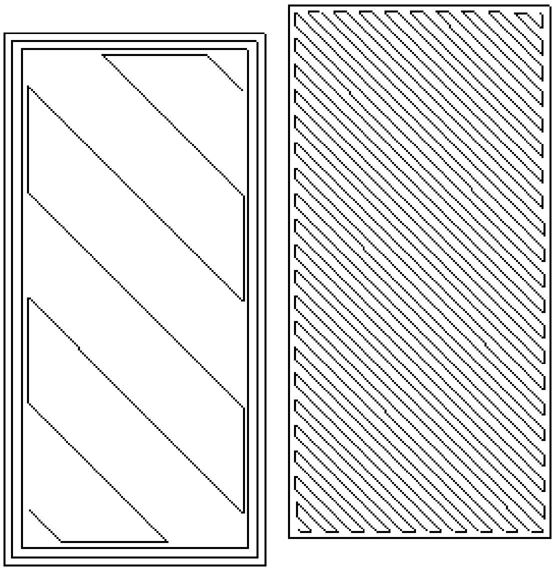

Figure 1. FOR A SIMPLE RECTANGULAR BLOCK, ALL OF THE INTERIOR SLICES ARE "HIDDEN" AND THUS CAN BE BUILT USING THE FAST BUILD STYLE PICTURED ON THE LEFT. FOR CONTRAST, THE REGULAR BUILD STYLE USED ON THE TOP AND BOTTOM SLICES IS PICTURED ON THE RIGHT, WITH DENSELY SPACED PARALLEL ROADS IN THE INTERIOR. IN AREAS WHERE A PART SURFACE SHOWS A SHALLOW SLOPE WITH RESPECT TO THE BUILD PLANE, THE BUILD STYLE ON THE LEFT CANNOT BE USED.

users of FDM machines may manually re-assign such layers to be built with the fast build option and change the number of concentric roads, but if they are too aggressive the result is a part such as the one pictured in figure 2.

Ideally, we would like to divide the part into a thin outer wall region (for the solid fill) and interior region(s) (for the loose fill). This division could be accomplished by finding the exact interior offset surface in 3D and then slicing this offset surface along with the original part; unfortunately, calculating a 3D offset is slow, difficult to program, and subject to failures caused by numerical accuracy limitations in floating point calculations. Since the wall need not be of perfectly uniform thickness, we can use a robust, easier-to-compute approximation while still obtaining full coverage at the part's surface.

RELATED WORK

Yu et al (22) describe applying Rossignac's solid offset algorithm for 3D constructive solid geometry (CSG) solids (16) in order to obtain offset surfaces for faster rapid prototyping. For input described with a b-rep, they suggest offsetting each slice contour individually in 2D, an approach that is clearly inadequate at

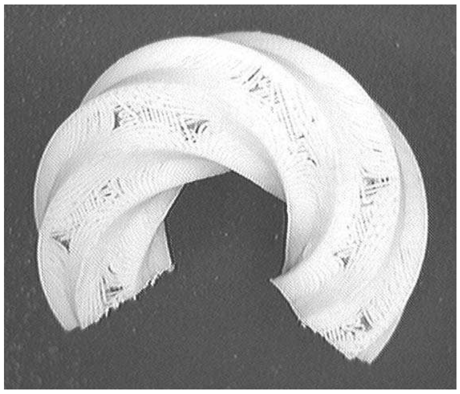

Figure 2. A PART BUILT WITH AN OVER-AGGRESSIVE MANUAL EXTENSION OF THE QUICKSLICE SOFTWARE'S FAST BUILD REGION. NOTE THE GAPS IN THE SURFACE ON THE NEAR-HORIZONTAL FACES.

horizontal and near-horizontal faces. The implementation of the CSG solid offset algorithm applied to SLA is described in Li et al (10). In Lam et al (8), they expand upon this work by describing how to derive an explicit representation for the FDM interior support geometry using an octree.

Allen and Dutta have studied the related problem of minimizing the need for supports in FDM by selectively thickening different wall areas. In (2) they describe their algorithm for building a subset of thin shell surfaces without any supports, and minimizing supports for more general surfaces and solids. The original implementation was for surfaces and solids of revolution. In (3) they detail an extension of the algorithm to general closed surfaces. This algorithm discretizes each layer to a grid, reclassifies cells inside each original contour to be solid or support cells depending on the propagation of information from neighboring cells, and then derives new contours around connected groups of solid and support cells for input to the FDM machine. This algorithm is not designed to produce walls of uniform thickness.

OUR THIN-WALLED ALGORITHM

Our algorithm uses internal 2D offset contours and regularized boolean set operations to approximate the true internal 3D offset surface. We generate the thin-walled region, one layer at a time, based only on the 2D slice information of the current slice and a few slices above and below.

We use regularized boolean set operations to ensure that our resulting contours are closed and have non-zero area. A regularized boolean operation is defined as the closure of the interior of the result of the corresponding standard boolean operation (15).

For each layer, at the least we want the solid fill pattern in the region between the boundary of the slice and its 2D inner offset. We will refer to this region as the "slice offset region." For the inner layers in the part shown in figure 3, the only region where we need a solid fill is the slice offset region. Call this slice offset region Region 1.

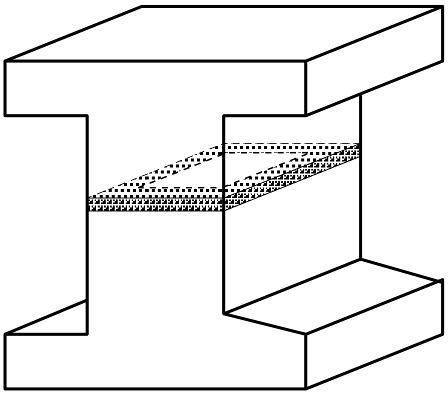

Figure 3. FOR THIS CENTRAL SLICE, THE AREA WE WANT TO FILL DENSELY WITH THE BUILD MATERIAL (A SOLID FILL) IS SIMPLY THE SLICE OFFSET REGION (REGION 1). THE INTERIOR REGION OF THIS LAYER WILL BE FILLED WITH A LOOSER CROSS-HATCHED PATTERN FOR SUPPORT.

But not all of our slices will be through vertical faces. At horizontal faces, we want a solid fill pattern not only in the slice offset region but also in the whole horizontal region, since it will be visible from the exterior of the part, as shown in figure 4. At angled faces, we want a solid fill pattern in the region of the current slice that is not covered by the two adjacent slices, since this will also be visible from the exterior of the part, as shown in figure 5. For near-vertical faces, this region will be a subset of Region 1, but we will need to explicitly calculate it for near-horizontal faces. Both these cases – horizontal faces and angled, near-horizontal faces – are taken care of by doing a solid fill (in addition to inside Region 1) inside any part of the current slice that doesn't appear in the slice below or above it. We subtract the slice above and the slice below from the current slice to find this region. Call this Region 2.

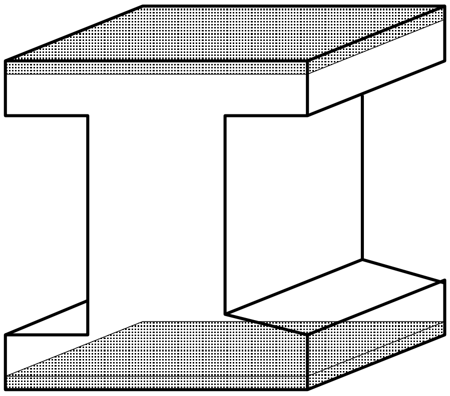

Figure 4. WE USE A SOLID FILL IN THE SLICES DIRECTLY ABOVE OR BELOW HORIZONTAL FACES (REGION 2).

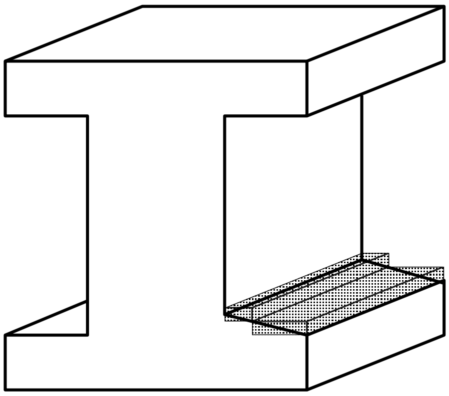

Figure 5. WE ALSO USE A SOLID FILL AT ANGLED FACES ANYWHERE THE CURRENT SLICE IS NOT COVERED BY THE SLICE ABOVE OR BELOW (REGION 2).

This will give us a solid fill at the visible surface of the part, but we won't have a very good approximation of a thin wall yet. Where horizontal or near horizontal faces meet vertical faces, for example, we'll get "gaps" in the interior boundary of the wall as shown in figure 6.

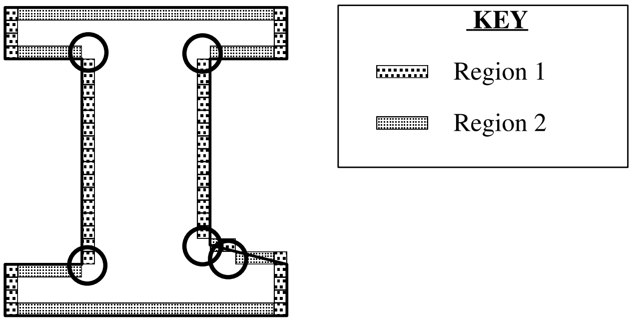

Figure 6. HERE WE ARE LOOKING AT A CROSS-SECTION OF THE PART. ALL OF THE REGIONS THAT ARE IN THE SLICE OFFSET REGIONS ARE LABELED WITH REGION 1 SHADING. THE REGIONS THAT WERE IN REGION 2 BUT NOT REGION 1 ARE LABELED WITH REGION 2 SHADING. THE "GAPS" IN THE THIN WALL ARE CIRCLED.

To avoid such gaps, we also do a solid fill anywhere in the current slice that also appeared in the slice offset of the slice above or below (this is equivalent to the intersection of the current slice with both of the two adjacent slice offsets). Call this Region 3.

Solid filling these three regions, we can get a good approximation of a one-slice thick offset (see figure 7).

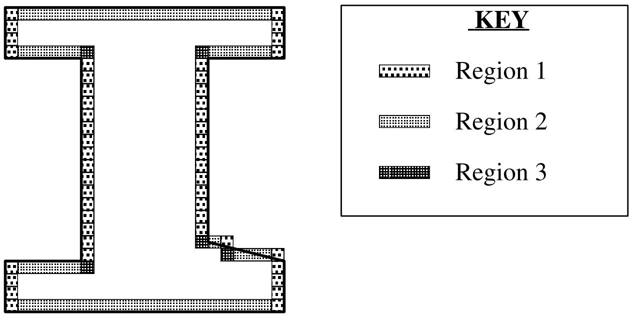

Figure 7. LOOKING AT THE SAME CROSS SECTION, WE SEE THE ADDITIONAL AREAS THAT ARE SOLID FILLED WHEN WE ADD REGION 3.

We can express the combination mathematically as follows: Call the region of the nth slice S_n, and call its slice offset (the region between its boundary and its inner offset) $Offset(S_n)$.

Region 1 is $Offset(S_n)$.

Region 2 is $(S_n -^* S_{n+1}) \cup^* (S_n -^* S_{n-1})$.

Region 3 is $S_n \cap^* (Offset(S_{n+1}) \cup^* Offset(S_{n-1}))$.

Taking the union of these three regions, the formula for the entire region that will get solid fill at the nth layer is:

$$Offset(S_n) \cup^* (S_n -^* S_{n+1}) \cup^* (S_n -^* S_{n-1}) \cup^* (S_n \cap^* (Offset(S_{n+1}) \cup^* Offset(S_{n-1}))).$$

Typically the wall should be thicker than only a single layer to obtain a sturdy part. For Region 1, we simply change the offset distance based on our desired wall thickness. For Regions 2 and 3, we can modify our previous formulas to take slices further than one layer away into consideration, to get a thicker wall.

For Region 2 (the horizontal and near-horizontal region), we want to extend this region down or up, depending upon the orientation of the face, through as many layers as we want our wall to be thick, as shown in figure 8. Of course, we only want to fill this extension if it falls inside our current slice. Call the number of layers needed to achieve our desired thickness T. Then the extended Region 2/2+ for layer n is:

$$S_n \cap^* (\bigcup_{i=1}^{T}{}^* ((S_{n+i-1} -^* S_{n+i}) \cup^* (S_{n-i+1} -^* S_{n-i}))).$$

Similarly for Region 3 (the extension of the neighbor's offset slice), we extend the region up and down T layers, as long as it falls inside these slices, as shown in figures 9 and 10. The extended Region 3/3+ for layer n is:

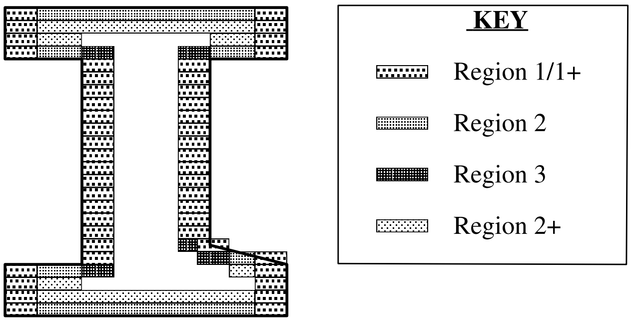

Figure 8. FOR A WALL THAT WE WANT TO BE TWO LAYERS THICK, WE MUST EXTEND REGION 2 UP AN ADDITIONAL LAYER ABOVE DOWN FACES AND DOWN AN ADDITIONAL LAYER BELOW UP FACES.

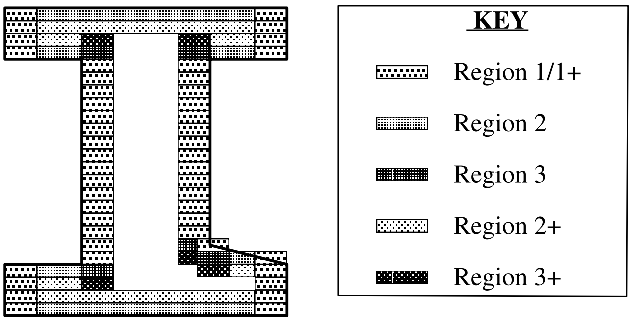

Figure 9. THE CROSS SECTION OF THE FULL 2-LAYER THICK WALL, SHOWING THE ADDITION OF THE EXTENDED REGION 3.

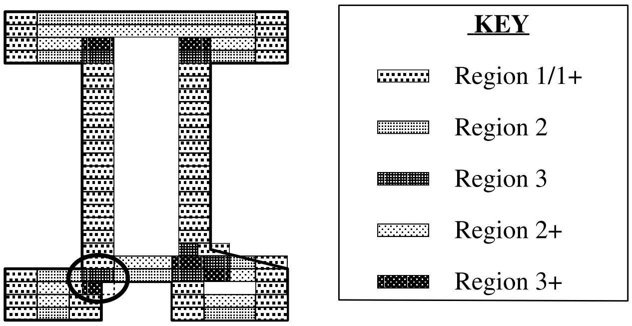

Figure 10. AN EXAMPLE WHERE IT IS NECESSARY TO CLIP THE EXTENDED REGION 3 AGAINST THE BOUNDARY OF THE CURRENT SLICE.

$$S_n \cap^* (\bigcup_{i=1}^{T}{}^* (Offset(S_{n+i}) \cup^* Offset(S_{n-i}))).$$

Combining these three extended regions, the final formula for the region that will be solid filled at the nth layer is:

$Offset(S_n) \cup^*$

$(S_n \cap^* ((\bigcup_{i=1}^{T}{}^*((S_{n+i-1} -^* S_{n+i}) \cup^* (S_{n-i+1} -^* S_{n-i})) \cup^*$
$(Offset(S_{n+i}) \cup^* Offset(S_{n-i}))))).$

This yields a very close approximation to a wall T layers thick.

THIN WALL IMPLEMENTATION

Our algorithm takes a tessellated b-rep as input, either in STL (1) or in SIF, our Solid Interchange Format (14; 12). We use our slicing software described in (13) to slice the input, exploiting geometric and topological inter-slice coherence to output clean slices with explicit nesting of contours. We output the slices in our neutral layer format, L-SIF (Layered Solid Interchange Format) (21). The L-SIF format describes each layer as a boolean combination of closed, oriented contours, so we can also express the boolean combination of the layers and their offsets, derived using the formula above, in L-SIF. The 2D offsetting algorithm is described in detail in the section below. For more efficient calculations, we store the intermediate results (the offsets, the differences between adjacent slices, and subsets of the summations of unions) for reuse in neighboring slice calculations.

To go from L-SIF to the FDM machine, we have written a program that resolves these 2D booleans and outputs oriented, nested contours in the SSL slice format used by QuickSlice (19). We then rely on QuickSlice to calculate where external supports for overhangs are needed and to compute the actual roads, using the loose-filled and solid-filled build styles as specified in our SSL file.

A bug in the QuickSlice 6.2 software manifests itself when we use identical contours to describe the outer contour of the loosely-filled inner region and the hole contour for the solid-filled outer wall region. Unfortunately, if we describe them with identical coincident contours, the fact that the two regions they bound have different build characteristics confuses the software and it overwrites the loosely-filled build style with the solid-filled build style on alternate slices. As a work-around, we assign the build characteristics of the loosely-filled build style to both copies of the contour, even though one copy bounds the inside of the solid-filled region, and then QuickSlice produces the result we want when it generates roads.

GENERATING OFFSET CONTOURS

One of the steps in our algorithm for the creation of a thin-walled part for SFF is the creation of 2D offset contours for each of the 2D slices through the 3D input polyhedron. There are two types of algorithms for creating 2D offset contours: algorithms that offset all edges and vertices of a contour separately and then trim the resulting offset segments, or algorithms that use the combined Voronoi diagram of all the input contours. We present an algorithm that computes the combined Voronoi diagram of a set of oriented input contours and uses this Voronoi diagram to create the offset contours.

Offsetting

Algorithms that offset each site of a contour separately generate offset segments that self intersect each other. These techniques then rely on boolean intersections to trim away excess offset geometry. It is necessary to compare and possibly compute the intersection of every pairing of offset segments, so the worst case running time is $O(n^2)$. The boolean operations are also susceptible to numeric drift because some round off error is incurred during the creation of each intersection point.

Offsetting algorithms that use the combined Voronoi diagram (*VD*) of all the input contours can be faster and more robust numerically than algorithms that offset each site independently. The Voronoi diagram of a set of contours partitions $\Re^2$ into a mesh of Voronoi faces (*VF*'s). Each VF_i is associated with a single input site s_i. The input sites are the vertices and edges that are part of the input contours. The edges can in general be straight line segments, circular arcs, or any free form curve. For the purposes of this discussion of Voronoi diagrams, we will limit our contours to be constructed out of straight line segments and circular arcs with $\Theta < \pi$. Each VF_i contains the set of points in $\Re^2$ that are closer to s_i than any other input site. *VF*'s are continuous. VF_i will have an infinite area if s_i is the closest site to points at infinity in some set of directions. *VF*'s are bounded and connected by Voronoi edges (*VE*'s). *VE*'s are the set of points which are equidistant from two input sites. *VE*'s are segments of straight lines, parabolas, hyperbolas, or ellipses. If we limit our set of input sites further to only vertices and straight line edges, then the *VE*'s will be segments of straight lines or parabolas. *VE*'s are bounded and connected by Voronoi vertices (*VV*'s). *VV*'s are points in $\Re^2$ that are equidistant to at least three input sites. The Voronoi diagram is the mesh consisting of all the *VF*'s, *VE*'s, and *VV*'s. For a more detailed description of Voronoi diagrams, consult Held (11).

The *VD* is useful in the creation of contour offsets. The *VD* encodes the distance function to the closest input site for $\Re^2$. This distance function can be viewed in 3D as the Voronoi mountain (20). The Voronoi mountain is generated by lifting each point p of $\Re^2$ in z by its signed distance to the nearest input site. The signed distance is positive if p is inside the general polygon defined by the oriented input contours, and it is negative if p is outside this polygon. Applying this lifting map transforms the 2D Voronoi mesh into a 3D mountainous surface as shown in figure 12. The *VF* of a line segment site turns into a piece of a 45° tilted plane. The *VF* of a vertex or circular arc segment site turns into a piece of a 45° angle cone. The *VE*'s become the intersection curves of these planes and cones. The *VD* provides global nearest neigh-

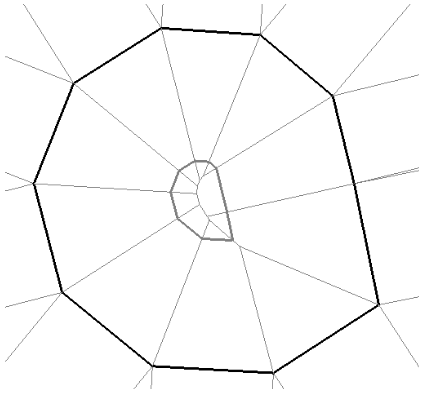

Figure 11. INPUT CONTOUR (THICK BLACK), VORONOI DIAGRAM (THIN GRAY), AND OFFSET CONTOUR (THICK GRAY).

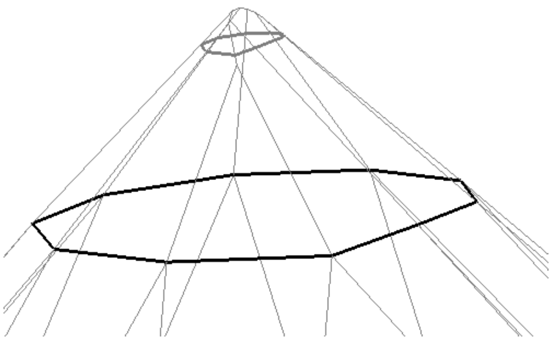

Figure 12. INPUT CONTOUR IN $z = 0$ (THICK BLACK), VORONOI MOUNTAIN (THIN GRAY), AND OFFSET CONTOUR IN $z = d$ (THICK GRAY).

bor information of the input sites by the *VF* adjacencies encoded in the *VE*'s. The offset contour may step directly between *VF*'s of input sites that were not consecutive sites in the input contour. An example is illustrated in the lower right corner of the offset contour in figure 11. This nearest neighbor information eliminates the extraneous pairwise comparisons of sites found in the brute force boolean approach to offsetting.

The lower bound on creating the Voronoi diagram of the input contours is $O(n \log n)$. This is true because the creation of the *VD* can be reduced to sorting. Algorithms that use the Voronoi diagram to create offset contours can run in $O(n)$ time, so total running time including the creation of the Voronoi diagram has a lower bound of $O(n \log n)$. It is also easier to create numerically stable algorithms for creating Voronoi diagrams than it is for performing boolean operations.

Kim (6) describes an $O(n)$ algorithm for constructing an offset contour of a simple input polygon using the Voronoi diagram of the polygon and two stacks to manage the creation of disjoint contour loops in the offset contour. The algorithm proceeds by walking around the input contour, creating offset segments when they exist, and jumps to create new contour loops when discontinuities are encountered. This only works for a single simple input contour.

A more general approach to creating the offset contours is to slice the Voronoi mountain by a plane $z = d$ where d is the signed offset distance: $d > 0$ offsets the polygon inward, $d = 0$ is the set of input contours, and $d < 0$ offsets the polygon outward. The offset contours at a distance d are equivalent to the slice contours created by slicing the Voronoi mountain with the plane $z = d$ and then projecting back into the $z = 0$ plane.

Voronoi Diagrams

Many algorithms exist for the creation of the Voronoi diagram of a contour. A major class of these algorithms is the divide and conquer algorithms. These algorithms were derived from the divide and conquer algorithm used by Shamos and Hoey (17) to create the Voronoi diagram of a set of vertices in $\Re^2$. The algorithm divides the input vertices into two half sets V_L and V_R based on their geometric position, recursively creates the VD_L and VD_R of those two sets respectively, and then merges VD_L and VD_R to create the *VD* of the whole set. The key operation in the algorithm is the $O(n)$ merge step. The merge builds the bisector polyline that separates the sites of V_L and V_R, and it prunes away the defunct geometry from the VD_L and VD_R. The key insight in the merge step is that all portions of VD_L that lie to the right of the bisector polyline will not be in the final *VD*, and vice versa. Others ((9) and (5)) describe algorithms that do the same work in creating the Delaunay triangulation of a set of vertices. The Delaunay triangulation is the planar dual of the Voronoi diagram.

Held (11) and Kim et al (7) describe algorithms that create the *VD* of an input polygon with straight lines and circular arcs as edges. These algorithms are both generalizations of the Voronoi diagram for point sets. Instead of dividing the input sites by geometric position, these algorithms divide the contour into two halves based on the topological counterclockwise ordering around the input contour. Both algorithms limit themselves to creating the *VD* of the inside of the contour only. Both algorithms create the bisector polyline during the merge step by creating representations of the *VE*'s between the *VF*'s and then intersecting the *VE*'s. The major difference between the two approaches is the representation of the functions for the *VE*'s. Held uses an implicit form parameterized by the distance to the input contour. Kim et al use rational quadratic Bézier segments to represent the conic curves for the *VE*'s. Held's approach gives a parameterization that is very intuitive, but it does not always have unique points for a parameter value. Kim et al create *VE*'s with monotonic parameterizations, so when offsetting the parameterization yields at most one unique point. Kim et al are able to simplify coding to a single operation, the intersection of a rational Bézier with a rational Bézier, but this may come at the expense of some numerical accuracy. Held's algorithm has many more intersection cases, but it is possible for it to generate higher numeric accuracy. Held specifically mentions the handling of a single general outer contour with convex inner contours. It is not clear whether the algorithm can be extended to handle island contours of arbitrary shapes. These divide and conquer algorithms are worst case optimal with $O(n \log n)$.

Our Offset Implementation

When creating polygon offsets for SFF, it is necessary to handle contours made up of arbitrary straight line segments with arbitrary island contours. The 2D contours are created by slicing 3D polyhedral approximations of free form shapes. The primitives are always linear, but the configurations can be very general and may be higher in complexity than examples normally experienced in NC pocket machining. Our approach is to create the Voronoi diagram of these contours, and then create the offset contours by conceptually slicing the Voronoi mountain.

We construct the Voronoi diagram using a divide and conquer method that builds the Voronoi diagram on the inside and on the outside of each input contour. By computing the inside and outside Voronoi diagram, it makes it possible to handle arbitrary island contours as shown in figure 13. A major concern in this algorithm is numeric accuracy, so we are careful to make all calculations based on the original input data without creating intermediate bisector representations. In the future, we plan on incorporating the exact arithmetic methods presented by Shewchuk (18) when calculating the *VV*'s. The individual contour *VD*'s can be combined using the same merge operation to form the combined *VD* of all the input contours. Details for merging the *VD*'s of island contours are discussed by Held (11).

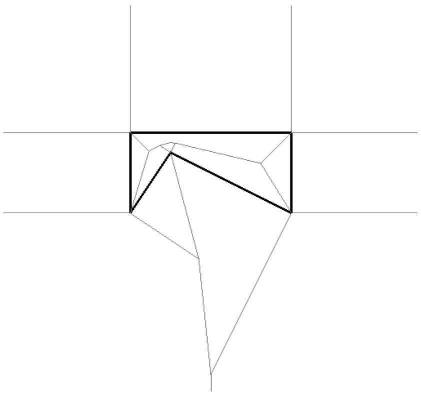

Figure 13. INPUT CONTOUR (THICK BLACK) AND VORONOI DIAGRAM (THIN GRAY).

Once the *VD* is created, we compute the offset contours by walking the edges of the *VD* as shown in figure 14. The algorithm starts by marking all of the *VE*'s as unprocessed. It then picks the first unprocessed *VE*. If it does not have an intersection point with the offset plane it is ignored and marked as processed. If it does intersect the offset plane, then it becomes the starting *VE* in a crawl of the *VD* to recover a single offset contour around a single peak of the Voronoi mountain. The crawl proceeds by walking the interior sides of the edges of the *VF* in clockwise order until the next *VE* intersected by the offset plane is encountered. A contour segment is created between the previous point and the new point. Then we step over the newly intersected *VE* into the adjacent *VF*. We continue to trace out the offset contour in counterclockwise order around the peak until the starting *VE* is encountered again. All edges that are touched in this walk are marked as processed. The algorithm repeats until all edges have been processed. This offsetting algorithm works for arbitrary sets of input contours and can offset inwards or outwards as shown in figure 14. The running time of the offsetting algorithm only is $O(n)$ because it touches each of the edges of the Voronoi diagram a constant number of times.

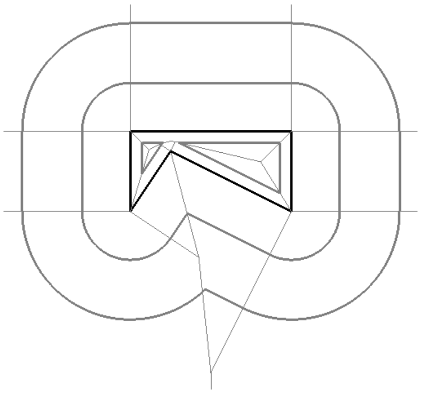

Figure 14. INPUT CONTOUR (THICK BLACK), VORONOI DIAGRAM (THIN GRAY), AND ONE INNER AND TWO OUTER OFFSET CONTOURS (THICK GRAY).

RESULTS

We have manufactured the simple test part shown in figure 15, a very short asymmetrical pyramid with a near horizontal face on the right. Using our algorithm and a desired wall thickness of $T = 3$, the build time for this part was 110 minutes and it used 203 inches (5.2 m) of material. The QuickSlice software didn't find any slices to manufacture with the fast build style for this part. Using the regular build style, the build time was 131 minutes and it used 252 inches (6.4 m) of material, taking 19% longer and using 24% more material than with our algorithm, and producing a part which looks identical on the outside, as shown in figure 16.

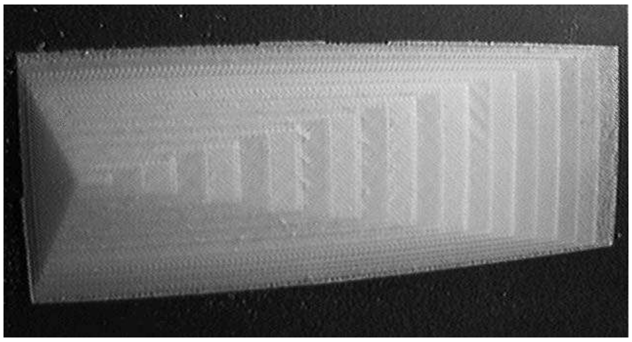

Figure 15. A SIMPLE TEST PART WITH A SOLID THIN EXTERIOR WALL AND LOOSE FILLED INTERIOR BUILT USING OUR ALGORITHM.

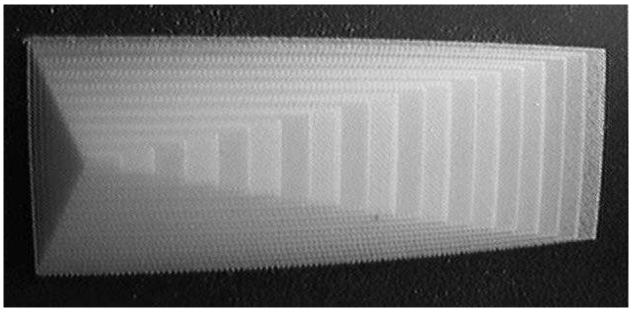

Figure 16. THE SAME PART BUILT USING THE FDM'S MACHINE'S QUICKSLICE SOFTWARE. THE PART LOOKS IDENTICAL ON THE EXTERIOR.

In this simple test part, the volume to surface ratio was quite small. For a part with a higher volume to surface ratio, such as the screw part shown in figure 17, the differences in build times can be more dramatic. For this part, the gentle slope of the screw threads prevents the QuickSlice software from applying its fast build algorithm (figure 18), but our algorithm still builds a thin-walled part (figure 19). Using our algorithm and a desired wall thickness of $T = 5$, the build time for this part was 232 minutes (3:52) and it used 301 inches (7.6 m) of material. Using the QuickSlice software directly, the build time was 504 minutes (8:24) and it used 872 inches (22.1 m) of material, taking 2.17 times as long and using 2.9 times as much material.

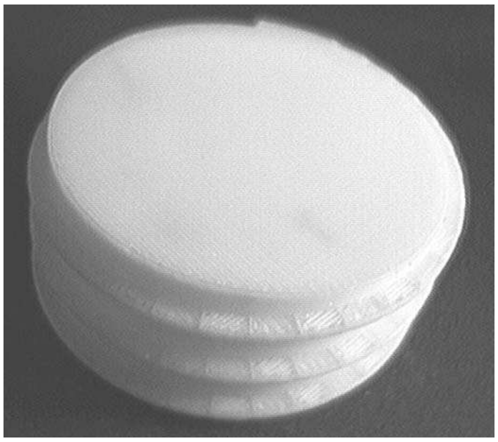

Figure 17. THE SCREW PART MANUFACTURED USING OUR ALGORITHM. USING THE QUICKSLICE SOFTWARE DIRECTLY, THE BUILD TOOK OVER TWICE AS LONG TO COMPLETE.

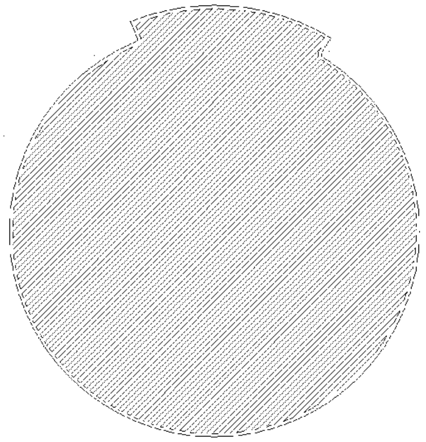

Figure 18. A SAMPLE SLICE THROUGH THE SCREW PART, USING THE QUICKSLICE SOFTWARE'S FAST BUILD OPTION. ALL OF THE ROADS ARE DENSELY SPACED.

CONCLUSION

In this paper, we have described a conceptually simple algorithm for making partially hollow layered parts with thin, dense walls of approximately uniform thickness. We have implemented this algorithm for an FDM machine to produce lighter parts using less material and in less time than with the commercial software.

The techniques described in this paper, with small modifications, should also speed up manufacturing with other calligraphic (random-scan) SFF technologies, such as SLA and SLS. With those technologies, however, modifications to the algorithm would be needed to obtain the same material savings, since it would be necessary to eliminate the trapped volumes of material in the part interiors.

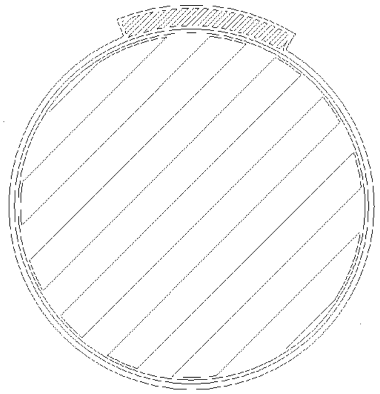

Figure 19. THE SAME SLICE USING OUR ALGORITHM. THE INTERIOR ROADS ARE LOOSELY FILLED FOR A FASTER BUILD.

ACKNOWLEDGMENT

This work was supported in part by grant EIA-9905140 from the National Science Foundation. The purchase of the FDM machine was funded in part by Visteon Automotive Systems and grant EIA-96-17995 from the National Science Foundation.

REFERENCES

3D Systems, Inc. Stereolithography Interface Specification. Company literature, 1988.

Seth Allen and Debasish Dutta. Wall thickness control in layered manufacturing. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 240–247, Nice, France, June 1997. ACM.

Seth Allen and Debasish Dutta. Wall thickness control in layered manufacturing for surfaces with closed slices. *Computational Geometry: Theory and Applications*, 10(4):223–238, July 1998.

Joseph J. Beaman et al. *Solid Freeform Fabrication : A New Direction in Manufacturing*. Kluwer Academic Publishers, Dordrecht, 1997.

L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, April 1985.

D.-S. Kim. Polygon offsetting using a Voronoi diagram and two stacks. *Computer Aided Design*, 30(14):1069–1076, Dec. 1998.

D.-S. Kim, P. K. Hwang, and B.-J. Park. Representing the Voronoi diagram of a simple polygon using rational quadratic Bezier curves. *Computer Aided Design*, 27(8):605–614, Aug. 1995.

T. W. Lam, K. M. Yu, K. M. Cheung, and C. L. Li. Octree reinforced thin shell object rapid prototyping by fused deposition modelling. *International Journal of Advanced Manufacturing Technology*, 14(9):631–636, 1998.

D. T. Lee and B. J. Schachter. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, June 1980.

C. L. Li, K. M. Yu, and T. W. Lam. Implementation and evaluation of thin-shell rapid prototype. *Computers in Industry*, 35(2):185–193, March 1998.

M. Held. *On the Computational Geometry of Pocket Milling*. Springer, Berlin, lncs edition, 1991.

Sara McMains. The SIF_SFF Page. http://www.cs.berkeley.edu/ũg/sif_2_0/ SIF_SFF.shtml, 1999.

Sara McMains and Carlo Séquin. A Coherent Sweep Plane Slicer for Layered Manufacturing. In *Fifth Symposium on Solid Modeling and Applications*, pages 285–295, Ann Arbor, MI, June 1999. ACM.

Sara McMains, Carlo Séquin, and Jordan Smith. SIF: A Solid Interchange Format for Rapid Prototyping. In *Proceedings of the 31st CIRP International Seminar on Manufacturing Systems*, pages 40–45. CIRP, May 1998.

A. A. G. Requicha. Representations for Rigid Solids: Theory, Methods, and Systems. *ACM Computing Surveys*, pages 437–464, December 1980.

J. R. Rossignac. *Blending and Offsetting Solid Models*. PhD thesis, University of Rochester, 1985.

M. I. Shamos and D. Hoey. Closest Point Problems. *Proceedings 16th Annual IEEE Symposium on Foundation of Computer Science*, Oct. 1975.

J. R. Shewchuk. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry*, 18(3):305–363, Oct. 1997.

Stratasys, Inc., Eden Prairie, MN. *QuickSlice 6.2*, 1999.

D. Veeramani and Y.-S. Gau. Selection of an optimal set of cutting-tool sizes for 2 1/2 D pocket machining. *Computer Aided Design*, 29(12):869–877, Dec. 1997.

Jianlin Wang. L-SIF Version 1.0. http://www.cs.berkeley.edu/ũg/LSIF/ LSIF.html, 1999.

K. M. Yu and C. L. Li. Speeding up rapid prototyping by offset. *Proceedings of the Institution of Mechanical Engineers, Part B*, 209(B1):1–8, 1995.