Iterative carving for self-supporting 3D printed cavities

Samuel Hornus, Sylvain Lefebvre

Iterative carving for self-supporting 3D printed cavities †

Samuel Hornus and Sylvain Lefebvre

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Abstract

Additive manufacturing technologies fabricate objects layer by layer, adding material on top of already solidified layers. A key challenge is to ensure that there is always material below, for otherwise added material simply falls under the effect of gravity. This is a critical issue with most technologies, and with fused filament in particular. In this work we investigate how to compute as large as possible empty cavities which boundaries are self-supporting. Our technique is based on an iterated carving algorithm, that is fast to compute and produces nested sets of inner walls. The walls have exactly the minimal printable thickness of the manufacturing process everywhere. Remarkably, our technique is out-of-core, sweeping through the model from the top down. Using our approach, we can print large objects using as little as a single filament thickness for the boundary, providing one order of magnitude reduction in print time and material use while guaranteeing *printability.*

CCS Concepts

•***Computing methodologies*** → *Shape modeling;* •***Applied computing*** → *Computer-aided design;*

1. Introduction

Additive manufacturing processes fabricate objects layerwise, from the bottom up, bonding each layer to the one below to form the final shape. This is a slow process and performance scales poorly since the volume grows to the cube of the object extent.

Several techniques have been proposed to reduce the time taken by the process, in particular by emptying the interior of the object (*i.e.*, by hollowing its erosion). However, the tops of the cavity cannot be printed directly as they are not supported from below. Thus, most software for additive manufacturing fill interiors with sparse fill patterns [LEM*17]. These patterns typically have a uniform density, and therefore take significant time and material to print within large volumes. Recent works define hierarchical, self-supporting fill patterns that print efficiently [Lef15] and can be subdivided following *e.g.* a mechanical criterion [WWZW16]. Lee *et al.* [LL16] propose to remove unnecessary facets in similar fill patterns so as to maximally empty a part.

Closer to our work, Hornus *et al.* [HLDC16] define large self-supporting cavities through morphological operations in the slices of an object. Our technique applies this construction iteratively in order to obtain printed objects that are almost completely empty, and further minimizes time and material by using only walls of the minimal printable thickness. While [HLDC16] relies on bitmaps, we discuss a robust *polygonal* implementation of the procedure and its iterative application, which is significantly faster on tall objects, and produces smoother paths.

† This work was supported by ERC grant ShapeForge (StG-2012-307877).

The idea of iterating self-supporting cavities has also been recently proposed in concurrent work [WLW*17]. This approach decomposes a volume into parts homeomorphic to cylinders, optimizes a cavity in each, and iterates on the remaining volumes. Cavities are optimized to consider objectives such as balance. While our work does not offer this capability, our approach generates more general cavities without the need for topological decomposition. It scales to large shapes and is orders of magnitude faster, while keeping only two slices in memory at any time. This is a crucial consideration when processing large objects. The present work is further detailed in our research report [HL17].

A polygonal implementation. One of the goals of minimizing the amount of material used is to afford for larger objects to be printed in reasonable time. For complex, detailed objects spanning a large number of slices (*e.g.*, several thousands) it is preferable to use a vectorial geometric description of the slices, for the complexity of a discrete, voxel-based representation grows too quickly. We represent a slice as a set of polygons with holes whose vertices lie on the integer grid with a 1 µm resolution: `typedef ClipperLib::Paths Slice;` In this setting, the *Clipper* library that we use is able to perform fast and robust boolean operations on the slices [Joh].

2. Carving cavities inside a shape

A single cavity would leave potentially large filled volumes between its walls and the actual surface. Thus, instead of producing a single cavity, we produce nested cavities: the remaining filled areas are iteratively analyzed and teared, spawning sub-cavities. Note that the iterations happen within a single slice, therefore the whole

process requires a single sweep from top to bottom and can be implemented out-of-core.

2.1. Working with slices.

Given a shape $X \subset \mathbb{R}^3$ we define the *slice of* X at height z, written $X_{|z}$, as the intersection of X with a horizontal plane at height z. Shapes are manipulated as a finite set of slices $\{X_{|i}, i = 1 \dots n\}$ ordered from the bottom up. The notation i is an index in the bottom up ordering instead of the actual height of the corresponding slice. If S is a planar shape and r a non-negative number, the *dilation* $S^{\uparrow r}$ is the set of points at distance r or less from S: $S^{\uparrow r} = \{p \mid \exists q \in S, |p-q| \leq r\}$. Similarly, the *erosion* of S of radius r is $S^{\downarrow r} = \overline{\overline{S}^{\uparrow r}}$ where $\overline{X}$ is the complement of X: $\overline{X} = \mathbb{R}^2 \setminus X$. The *opening* of S of radius r is $\textsc{Open}(S,r) = S^{\downarrow r \uparrow r}$. We implement these operations approximately using a Minkowski sum with a convex regular polygon D having 24 vertices with integer coordinates. The advantages are twofold: First, $S^{\uparrow r} \approx S \oplus D$ can be computed very quickly with contour convolutions [BL10]. Second, the operation is associative: $(S \oplus D) \oplus D = S \oplus (2D)$, so that the number of vertices on subsequent dilations or erosions never grows out of control. Implementation details for the Minkowski sum are provided in [HL17, Appendix A].

Let s be the radius of the extrusion nozzle (typically 0.2 mm). Let r be the distance by which we dilate each cavity when propagating it from one slice to the slice below (see below). The value of r is typically chosen so that the slope of the surface of a cavity is 45°, so that the resulting surfaces are self-supporting.

2.2. Modeling a single cavity

The single-cavity technique should model a large volume $C \subset O$ that can be completely carved out of a given object O, while still maintaining 3d-printability. We describe the CAV procedure from Hornus *et al.* [HLDC16] and then extend it with iteration in §2.3.

The idea for CAV is to sweep through the object slices from top to bottom and, at each slice, analyze whether a cavity can be started by "tearing" a filled area open. The tear is then propagated downwards through morphological dilation, producing a *self-supporting* surface. This is achieved by incorporating the shape $C^{\uparrow r}_{|i+1}$ into the slice $C_{|i}$ immediately below. The pseudocode for CAV is shown in Algorithm 1. Figure 1 gives a 2D-world example. Figure 2 illustrates the cavity modeling process in 3D on a selected number of slices. In this figure, the input is an L-shape of side length 12 mm extruded vertically to a height of 12 mm. The left column shows the seeding (green) and growth of the first cavity $C^1 = \textsc{Cav}(O)$ (red) where O is the input shape with some "cover" removed.

Seeding the growth of the cavity. In order to seed the growth of a cavity C inside O, we need a shape that can serve as topmost slice of C and bootstrap the growth of the cavity C in the slices below. [HLDC16] argues for the use of a filtered medial axis of each slice as the seed shape. We compute the *seed shape* by pruning the medial axis of the current slice using the *extended distance function* as defined and studied by Liu *et al.* [LCLJ11]. This particular pruned medial axis is particularly robust to small perturbations along the contours of the input polygons, contrary to the discrete

Algorithm 1 CAV computes a single cavity inside O.

```
1: function CAV(Shape O)
2:     C_{|n+1} ← empty slice                ▷ n is the number of slices of O
3:     for i = n downto 1 do
4:         G ← C^{↑r}_{|i+1}                 ▷ Dilate the upper slice
5:         S ← SKEL(O_{|i})                  ▷ Seed shape for that slice
6:         C_{|i} ← (G ∪ S) ∩ O_{|i}         ▷ C_{|i} has to stay in O_{|i}
7:     return cavity C
```

approach used in [HLDC16]. We thus obtain very clean seed shapes affording for fast and smooth motions of the 3D printer nozzle. We compute the medial axis of a slice using the *Segment Delaunay Graph* of the CGAL library [Kar16]. We prune it as detailed in [LCLJ11] and dilate the result by a disk of diameter equal to that of the printer nozzle. The contour of the dilation is now interpreted as a print-path and is ready to be used in the CAV procedure (Figure 2, top row).

Bridges. Once modeled using the CAV algorithm, the surface of the cavity may exhibit local minima. Cavity C^1 in Figure 1-middle has three such minima. Since the surface of the cavity is printed, additional support is required below each local minimum. We use *Clipper*'s hierarchical representation of the slice boundaries to detect the local minima as small empty holes that disappear in the slice below. We then replace each local minimum by a set of bridges that cover the local minimum and are anchored on the cavity boundary (Figure 3 top left). Dumas *et al.* propose a detailed analysis of these bridges [DHL14]. As argued in [HLDC16] using the medial-axis as seed shape tends to minimize the number of local minima.

2.3. Iterated carving

By observing Figure 1-middle, one can see that the complement of the cavity, $O \setminus C^1$, still makes for a significant volume. We thus propose to apply the CAV procedure again in order to model a second cavity in this region, and to iterate this carving process a number of times, each time carving a cavity in the volume that the previous cavity was not able to cover. The pseudocode for this procedure is shown in Algorithm 2. Figure 1-right demonstrates the modeling of a 4 iterated cavities in a 2D world. The second column of Figure 2 illustrates the 3D modeling of the second cavity C^2 (blue), including the seed-shape computed for $V^1 = O \setminus C^1$ (green).

Algorithm 2 The function ITERATIVECARVING.

```
1: function ITERATIVECARVING(Shape O, Integer k)
2:     V^0 ← O                  ▷ k is the number of iterations to perform.
3:     for i = 1 to k do
4:         C^i ← CAV(V^{i-1})
5:         V^i ← V^{i-1} \ C^i
6:     return (C^1, C^2, ..., C^k, V^k)
```

The number of iterations is either fixed (from 3 to 6 typically) or the iterative process stopped when no useful cavity can be created in the remaining volume V^i. Writing $C^0 = \mathbb{R}^3 \setminus O$, at the end of the call to ITERATIVECARVING one obtains a partition of $\mathbb{R}^3 = O \sqcup C^0 = V^k \sqcup \bigsqcup_{i=0}^{k} C^i$. (The symbol $\sqcup$ stands for the disjoint union.)

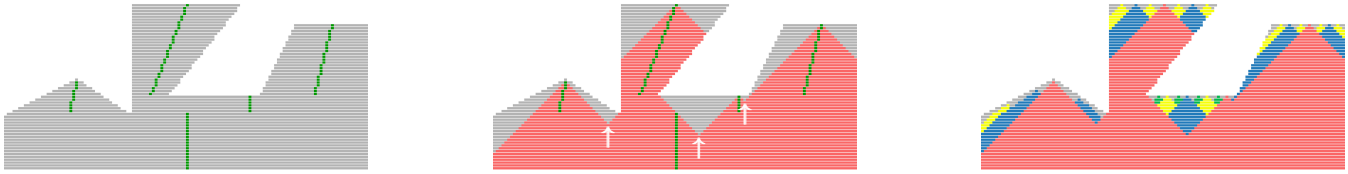

Figure 1: *In 2D, an object is represented as a stack of slices. Each slice is a set of collinear horizontal line segments.* Left: *The main object is O (gray). In 2D, the seed shape for a slice is simply the center of each segment (green).* Middle: *The cavity $C^1 = \textsc{Cav}(O)$ is drawn red. Each arrow points to a local minimum.* Right: *Iterated cavities. C^1 is drawn red, C^2 blue, C^3 yellow and C^4 green.*

Covers. Surfaces close to being horizontal are usually printed densely so as to avoid see-through holes. This dense part is known as the *cover* and is typically 2 to 6 layers thick. When modeling the cavities inside a shape, one should not touch the cover, so that the parameter O given to the IterativeCarving procedure should corresponds to the main object of interest with its cover subtracted. When enough iterations are used, the union of the boundaries of all the cavities just below the cover provides a good support for the cover (Figure 1-right). Alternatively, one may choose to fill the volume V^k with some common infill pattern.

3. From cavities to print-paths

The sliced representation of the cavities has to be turned into curves along which material should be solidified or deposited in a specific order. We call these curves *print-paths*. In the remainder we focus specifically on the case of widely available filament printers. The slices are processed independently of each other: we describe the process for a single slice. In a slice at height z, the object O and each cavity C^i are planar shapes $O_{|z}, C^i_{|z}$. Each boundary coincides locally with the boundary of another shape (a cavity C^i or V^k or O), so these boundaries can not be used directly as print-paths.

We print the boundaries of the cavities in the order they were modeled: $C^0, C^1, \ldots, C^k$. When extruding fused material along a print-paths p on the boundary curve of cavity C^i, we can thus assume that the print-paths for cavities $C^j, j < i$ have already been "printed." We then simply avoid extruding material on the parts of print-path p that coincide with a boundary of some cavity that has already been printed.

Recall that V^i is the volume out of which we model the cavity C^{i+1} using the Cav procedure. The boundaries of a slice $V^i_{|z}$ is composed of pieces of boundaries from $C^0, \ldots, C^i$ that can be thought of as print-paths, although they haven't been through smoothing and filtering of small components yet. The future print-paths for $C^{i+1}_{|z}$ must lie at a distance at least $\approx 2s$ away from these boundaries, *i.e.*, lie in an erosion of $V^i_{|z}$. We thus compute a *restriction shape* $W^i_{|z} \leftarrow V^{i\downarrow t}_{|z}$. The value of $t = 1.4s$ is explained below. The print-paths are obtained as follows: First, the cavity slices are smoothed and small components removed: $P^i_{|z} \leftarrow \textsc{Open}(C^i_{|z}, \frac{9s}{10})$. Then, we compute the intersection of both the cavity and its boundaries with the restriction shape: $Q^i_{|z} \leftarrow P^i_{|z} \cap W^i_{|z}$ and $\mathcal{R}^i_{|z} \leftarrow \partial P^i_{|z} \cap W^i_{|z}$.

$\mathcal{R}^i_{|z}$ is not a 2D planar shape but a set of open and closed curves in the plane. These curves form a subset of the boundaries of $Q^i_{|z}$: $\mathcal{R}^i_{|z} \subset \partial Q^i_{|z}$. The curves in $\partial Q^i_{|z} \setminus \mathcal{R}^i_{|z}$ are the precise locations where a print-path is not necessary because one will already be placed there in the processing of the slice of a cavity $C^j, j < i$.

To guarantee that the filaments along $\mathcal{R}^i_{|z}$ adhere well to the neighboring cavities $C^j, j < i$, we extend them along $\partial Q^i_{|z}$, over a distance of about 1 mm at both ends. The value of t is chosen a bit smaller than $2s$ to further increase adhesion along these extensions.

4. Results

Scalability. In this experiment, we process a single input model (`https://www.thingiverse.com/thing:12694`) at various scales, from 1 to 10. The number of iterations for modeling cavities is fixed at 6 so that the cavities (almost) completely cover the volume of the input model. Since the cavities are large and we only print their surface, we expect the quantity of material used to print the model to grow quadratically with the scaling factor. Figure 4 presents our measurements. When approximating the curves with a formula of type $R\sigma^\alpha$ where R is a constant and σ is the scaling factor (in abscissa). we obtain $\alpha \approx 2.1$ for the volume of extruded material. This indicates that the volume of our cavities walls scales more like a surface than like a volume.

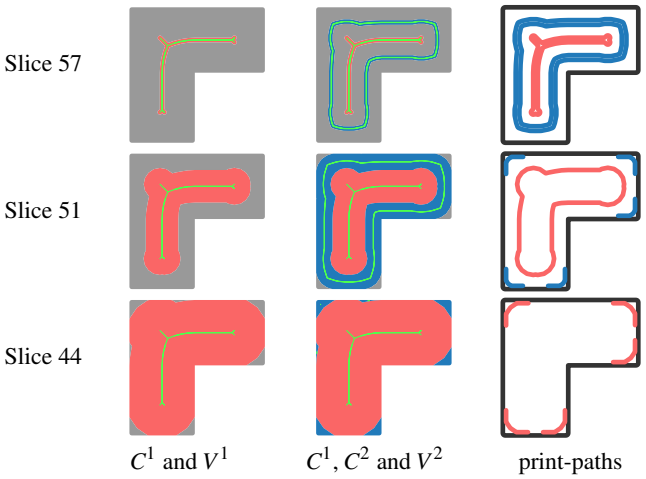

Figure 2: *An example of modeling cavities with two iterations. Same color scheme as Figure 1.*

Figure 3: Top left. *An unusually high number of bridges appears during the printing of the Fawn model.* Top right. *The models used for comparing with the work of Lee* et al. *[LL16].* Bottom. *Printing without inner shell makes the models transparent enough to let us see the inner cavities.*

Comparisons. We compare our technique to the recent work of Lee *et al.* [LL16], see Table 1. Seven simple but large models are used for this benchmark (Figure 3-top right). For all models but the *Fawn* our technique yields lower material usage and faster printing time. The Fawn model contains many bridges and we suspect this is the main reason why our technique is less efficient for this model. It should be possible however to improve our technique to remove many of these bridges which are not absolutely necessary.

We also compare to the concurrent work of [WLW*17], on a similar CPU. On their *Kitten* model we obtain 78 % material reduction vs. 75 % while being 273 times faster (6.9 s vs. 31.3 min). On their *Children* model we achieve 63 % vs. 53 %, while being 144 times faster (25 s vs. 59.7 min). Our approach scales trivially to very large models, while the number of variables to optimize in [WLW*17] grows quadratically. However, we cannot optimize for other objectives beyond maximizing cavity volumes.

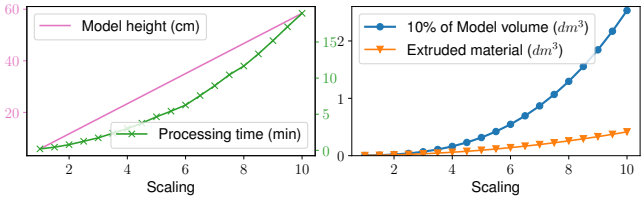

Figure 4: *Horizontal axis: scaling factor applied to the input model.* Left. *Model height and processing time, including loading the STL file, processing the slices and writing the GCode file.* Right. *Volume of the input model (**divided by 10** for clarity) and volume of the extruded material for printing the modeling with our cavities.*

Thin test. We have experimented with printing the models with a minimal amount of material, just 2 layers of "cover" and no shell, so that the surface of the model is printed with a single-filament thickness (only the perimeter is printed). Results are reported in columns "2c 0s" of Table 1. Because of their thinness a strong back light lets us see the cavities inside the object (Figure 3-bottom).

References

[BL10] BEHAR E., LIEN J.-M.: Extracting the Minkowski sum boundary from the reduced convolution. In *20th Annual Fall Workshop on Computational Geometry* (2010). 2

[DHL14] DUMAS J., HERGEL J., LEFEBVRE S.: Bridging the gap: automated steady scaffoldings for 3D printing. *ACM Transactions on Graphics 33*, 4 (2014), 98:1–98:10. 2

[HL17] HORNUS S., LEFEBVRE S.: *Iterative carving for self-supporting 3D printed cavities*. Tech. Rep. 9083, Inria, July 2017. 1, 2

[HLDC16] HORNUS S., LEFEBVRE S., DUMAS J., CLAUX F.: Tight printable enclosures and support structures for additive manufacturing. In *EG Workshop on Graphics for Digital Fabrication* (2016). 1, 2

[Joh] JOHNSON A.: Clipper. A C++ geometric library for boolean operations on polygons. http://angusj.com/. 1

[Kar16] KARAVELAS M.: 2D Segment Delaunay Graphs. In *CGAL User and Reference Manual*, 4.8.1 ed. CGAL Editorial Board, 2016. 2

[LCLJ11] LIU L., CHAMBERS E. W., LETSCHER D., JU T.: Extended grassfire transform on medial axes of 2D shapes. *Computer-Aided Design 43*, 11 (2011). 2

[Lef15] LEFEBVRE S.: 3D infilling: faster, stronger, simpler, 2015. http://sylefeb.blogspot.fr/2015/07/3dprint-3d-infilling-faster-stronger.html. 1

[LEM*17] LIVESU M., ELLERO S., MARTÍNEZ J., LEFEBVRE S., ATTENE M.: From 3D Models to 3D Prints: An Overview of the Processing Pipeline. *Computer Graphics Forum* (2017). 1

[LL16] LEE J., LEE K.: Block-based inner support structure generation algorithm for 3D printing using fused deposition modeling. *International Journal of Advanced Manufacturing Technology* (2016). 1, 4

[WLW*17] WANG W., LIU Y. J., WU J., TIAN S., WANG C. C. L., LIU L., LIU X.: Support-free hollowing. *IEEE Transactions on Visualization and Computer Graphics PP*, 99 (Oct. 2017). Accepted preprint. 1, 4

[WWZW16] WU J., WANG C. C. L., ZHANG X., WESTERMANN R.: Self-supporting rhombic infill structures for additive manufacturing. *Computer-Aided Design 80* (2016). 1

			Time (min.)			Volume (cm^3)		Weight (g)		
Model	Volume	k	2c 0s	3c 1s	M2	2c 0s	3c 1s	2c 0s	3c 1s	M2
Cat	261.1	5	183'	**223'**	240'	20.3 (7.77 %)	29.0 (11.11 %)	24.4	**35.5**	37.6
Fawn	387.6	6	249'	301'	**284'**	27.8 (7.17 %)	39.4 (10.17 %)	31.3	47.9	**45.6**
Fox	516.0	5	192'	**261'**	300'	22.0 (4.26 %)	36.3 (7.03 %)	26.4	**45.0**	52.7
Giraffe	355.5	5	203'	**258'**	301'	22.6 (6.36 %)	34.6 (9.73 %)	25.7	**42.1**	49.4
Moai	836.4	5	279'	**369'**	451'	32.4 (3.87 %)	52.4 (6.26 %)	36.6	**62.8**	77.1
Skull	1056.4	6	279'	**368'**	382'	35.4 (3.35 %)	53.5 (5.06 %)	42.6	**65.5**	68.9
Yoda	390.5	5	222'	**261'**	311'	21.5 (5.51 %)	32.7 (8.37 %)	24.6	**38.9**	51.3

Table 1: *Comparisons with Lee* et al. *[LL16]. "k" is the number of iterations used to model the cavities. "Volume" (first) is the volume of the input object. "Volume" (second) is the material used for printing. (In parenthesis: percentage of the total volume of the object.) The column "M2" shows the values of the best technique in the work of Lee* et al. *The column "3c 1s" uses a cover 3 layers thick and 1 shell (in addition to the perimeter). This is the same configuration used by Lee* et al. *so that this column is directly comparable with column "M2". Better values are typed in boldface.*