HAL
open science

# Optimal discrete slicing

Marc Alexa, Kristian Hildebrand, Sylvain Lefebvre

## To cite this version:

Marc Alexa, Kristian Hildebrand, Sylvain Lefebvre. Optimal discrete slicing. ACM Transactions on Graphics, 2017, 36 (1), pp.1 - 16. 10.1145/2999536 . hal-01660773

**HAL Id: hal-01660773**

**https://inria.hal.science/hal-01660773**

Submitted on 12 Dec 2017

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Optimal discrete slicing

MARC ALEXA
Technische Universität Berlin
and
KRISTIAN HILDEBRAND
Beuth Hochschule für Technik Berlin
and
SYLVAIN LEFEBVRE
INRIA

Slicing is the procedure necessary to prepare a shape for layered manufacturing. There are degrees of freedom in this process, such as the starting point of the slicing sequence and the thickness of each slice. The choice of these parameters influences the manufacturing process and its result: the number of slices significantly affects the time needed for manufacturing, while their thickness affects the error. Assuming a discrete setting, we measure the error as the number of voxels that are incorrectly assigned due to slicing. We provide an algorithm that generates, for a given set of available slice heights and a shape, a slicing that is provably optimal. By optimal we mean that the algorithm generates sequences with minimal error for any possible number of slices. The algorithm is fast and flexible, i.e. it can accommodate a user driven importance modulation of the error function and allows the interactive exploration of the desired quality/time tradeoff. We demonstrate the practical importance of our optimization on several 3D-printed results.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*

General Terms: Algortihms

Additional Key Words and Phrases: computer numerical control, direct digital manufacturing, additive manufacturing, slicing, dynamic programming

## 1. INTRODUCTION

Layered Manufacturing requires a shape to be *sliced*. Together, the layers approximate the object. Each layer has a non-negligible thickness so that irrespective of the particular manufacturing technology used, slicing introduces an error – often referred to as "staircase" effect.

This error depends on exactly how the shape is sliced. Most layered manufacturing technologies offer variable slice thickness. Selectively using thicker slices reduces the overall number of slices in the production process and potentially saves time, while selectively using thinner slices could lead to better accuracy [Dolenc and Mäkelä 1994; Sabourin et al. 1996; Kulkarni and Dutta 1996]. This better allocation of resources applies to many different manufacturing technologies, including fused deposition modeling (FDM) [Tyberg and Bøhn 1999; Pandey et al. 2003a], stereolithography (SLA) [Xu et al. 1997], selective laser sintering (SLS) [Singhal et al. 2008], and fully dense freeform fabrication (FDFF) [Hayasi and Asiabanpour 2013]. It can also be used for inkjet-based 3D printing processes (e.g., PolyJet/MultiJet) where variable slice thickness can be achieved by modifying the droplet volume.

In this setting we approach the problem of finding sequences of slice heights that are *optimal*, in the sense of minimizing the error for a given number of slices or minimizing the number of slices for a bound on the error. A central assumption in our approach is that the shape is represented on a discrete grid. This grid is used to represent the shape, the slices, and compute the error — optimality has to be understood in this setting. This approach is supported in view of the mechanical nature of the output devices (we discuss this further in the context of related work in Section 2 and present the details of our setup in Section 3).

In this context, our central contributions are a data structure and algorithms that enable computing the sequence of slice heights that leads to the *smallest total error* for *all possible number of slices*. This computation is divided into two phases: first, the error is computed and minimized for each *potential* slice — in Section 4 we explain how to perform this computation efficiently. Second, we compute optimal sequences using dynamic programming parameterized over the number of slices, and also discuss how the same approach could be used to solve related optimization problems (see Section 5).

We experiment with this algorithm and potential alternative slicing strategies. A theoretical analysis based on analyzing the volumetric error for different slicing sequences demonstrates that optimal slicing is better than uniform or greedy adaptive slicing — and also explains why other techniques fail. Moreover, relating slicing to sampling illustrates that any technique that makes local decisions should be expected to provide non-optimal results, regardless of how the error is measured. The theoretical analysis is presented in Section 6.

The implementation of the algorithm shows that computations can be performed for practically relevant data sets and parameters in a few seconds on commodity hardware, thus not interfering with the direct digital manufacturing pipeline. We use the resulting slice sequences to generate control code for several recent manufacturing devices, in particular FDM and stereolithography. We show on several examples that, indeed, optimized slicing sequences can be used to significantly decrease the production time at the same qua-

Marc Alexa acknowledges an ERC Starting Grant ("XShape", ERC-2010-StG 259550). Sylvain Lefebvre acknowledges an ERC Starting Grant ("Shapeforge", StG-2012-307877) Authors' addresses: M. Alexa, TU Berlin, Sekretariat MAR 6-6, Marchstr. 23, 10587 Berlin, Germany; email: marc.alexa@tu-berlin.de; K. Hildebrand, Beuth Hochschule Berlin, FB VI, Luxemburger Str. 20a, 13353 Berlin, Germany; email: khildebrand@beuth-hochschule.de; S. Lefebvre, LORIA/INRIA Lorraine, Campus scientifique, 615, rue du Jardin Botanique, 54600, Villers les Nancy, France; email: sylvain.lefebvre@inria.fr

lity level, or get measurably higher accuracy in the same production time (Section 7).

We argue that the error introduced by the slicing process cannot be modeled in a way agnostic to the application and discuss the properties of measuring the volumetric error in this context. We suggest to adapt the error model to specific scenarios by weighting, and illustrate how the user could change the slicing by painting a weight function onto the shape. We discuss further trade-offs throughout the paper and present some open questions in Section 8.

## 2. RELATED WORK

Optimizing the parameters that control direct digital manufacturing is a vast field, and we provide only a partial overview relevant to our approach. Slicing is one of the central aspects for all layered manufacturing techniques and various strategies have been suggested [Pandey et al. 2003b]. The central theme of adaptive slicing is a better trade-off between time to generate the result and application specific quality of the result. The two most important aspects of adaptive slicing are how to model the error resulting from slicing and how to adjust the slice heights based on this error. Tangential to slicing but relevant in the context of optimized digital manufacturing are the aspects of decomposition of the shape and its orientation relative to the device.

### 2.1 Error models

Given that there is a choice of how to slice the object, it is natural to minimize *an* error relative to the number of slices. There exist different strategies to model the error, and any such strategy necessarily is based on some assumption on the geometry of the resulting slice. The majority of works assumes that the geometry of a slice is a contour extruded orthogonal to the layer, i.e. every slice has "vertical walls". This assumption makes particular sense for approaches that are supposed to be agnostic to the specific manufacturing technology. From our experience, the assumption of vertical walls is very realistic for SLS, and a good approximation for FDFF and SLA. In FDM, layers are generated by strands of plastic that have a rather round shape on the contour (and indeed this shape has been taken into consideration to model the error specific to FDM [Pandey et al. 2003a]). The consequence of these observations is that no single synthetic error model will be able to capture the properties of artifacts manufactured on different devices.

Based on the assumption of vertical walls, most error models measure the so-called "stair-cases". The standard is *cusp height* [Dolenc and Mäkelä 1994; Sabourin et al. 1996; Tyberg and Bøhn 1998; Xu et al. 1997]. It defines the local error as the distance from the surface of the input mesh to the connection point of consecutive slices. It has been improved by using an adaptive error [Cormier et al. 2000] or modulated using computational saliency models [Wang et al. 2015]. Another model with similar properties takes the area difference of consecutive contour polygons [Zhao and Luc 2000].

Consistent with other works on slicing [Tata et al. 1998; Masood et al. 2000], we measure the error as the *volume* that is incorrectly assigned: volume outside the shape that is covered by the sliced shape, as well as volume inside the shape that is not covered by the sliced shape. This would also allow considering contours that are not vertical within each slice; we however leave this generalization for future work. Note that the algorithm we develop for computing the optimal sequence is agnostic to the error model and can be used independent of how the error had been computed for each slice.

A central modeling assumption is that we consider the volume to be discretized. This is quite natural and widely adopted (e.g. [Vidimče et al. 2013]) in the setting of physical realization: all manufacturing processes have tolerances, and positioning is commonly done using stepper motors, whose step width or angle naturally provides a discretization along the axes. Recent SLA printers project digital images to cure the resin layer (e.g. Autodesk Ember, B9Creator) and natively require a discrete input.

### 2.2 Adaptive slicing strategies

The simplest way to adapt the local slice height is by directly mapping the local error measure to the slice height [Dolenc and Mäkelä 1994]. This method suffers from the inability to control the resulting number of slices.

There are two main strategies for varying slice thickness that enable control over the number of slices. Fine-to-coarse approaches start with the thinnest possible slicing configuration and merge consecutive slices depending on the error [Hayasi and Asiabanpour 2013]. Coarse-to-fine approaches have been preferred by many earlier works [Sabourin et al. 1996; Tyberg and Bøhn 1999; Kulkarni and Dutta 1996; Hope et al. 1997]. The main idea is to start with a coarse slicing sequence and refine it by subdivision. All of these techniques are essentially greedy – we demonstrate that they produce non-optimal results in Section 6.

A global approach to adaptive slicing has been introduced by Wang et al. [2015]. They formulate the goal of slicing as finding the smallest number of slices under the constraints of the slices having bounded thickness and the overall error bounded from above [Wang et al. 2015, Eq. 4]. As the slices may vary continuously within the allowed interval, the solution to the optimization can only be approximated. Our main observation is that by discretizing the available slice heights, a globally optimal solution *for the same general problem* becomes feasible. We slightly generalize the setting and provide sequences of all possible lengths. This avoids picking a bound on the error a priori. This more general solution is still simpler and faster than the approximate solution for the continuous setting.

### 2.3 Decomposition

Decomposing the shape can improve the manufacturing process, for instance by restricting the parts to pyramids [Hu et al. 2014] avoiding overhangs and thereby support material. Wang et al. [2015; 2016] directly target quality of the result by subdividing the shape into parts which are treated differently to either reduce manufacturing time but still maintain visual quality or optimize the visual appearance by better adapting each piece to the manufacturing process. These approaches where inspired by earlier works on local adaptive- [Tyberg and Bøhn 1998] and region-based slicing [Mani et al. 1999; Sabourin et al. 1997]. All of them relax the constraint that an entire layer should print with the same thickness, thus allowing more resolution in some areas. Our approach could work in conjunction with these techniques, for optimally slicing sub-parts after partitioning.

### 2.4 Orientation

The manufactured result depends on the orientation of the shape relative to the device. An important consideration for layered manufacturing is structural stability [Umetani and Schmidt 2013], as the material properties are highly anisotropic. Also, the visual appearance of the surface depends on the orientation: because the surface quality depends on the orientation of the surface relative to the

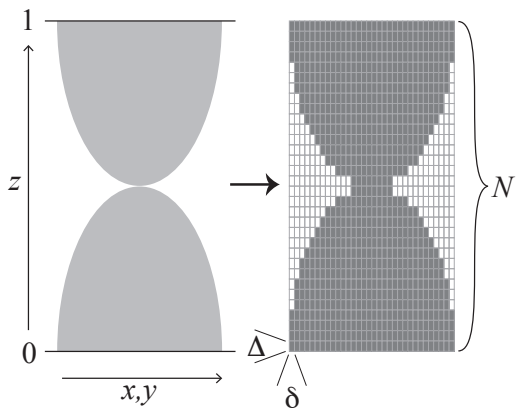


Fig. 1. The input shape $\mathcal{S}$ is assumed to cover $[0, 1]$ in direction $\mathbf{z}$. The shape is discretized and represented on an occupancy grid with grid constant $\Delta = 1/N$ in direction $\mathbf{z}$, where $N$ is the number of layers along $z$ covering the shape; and grid constant $\delta$ in the directions $\mathbf{x}$ and $\mathbf{y}$.

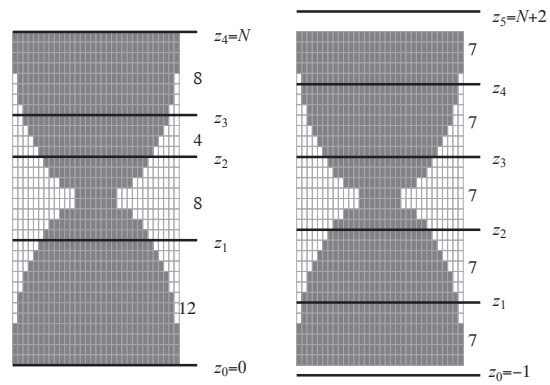


Fig. 2. A slicing is represented by the sequence $Z = \{z_0, z_1, \ldots, z_n\}$ along the $\mathbf{z}$-direction. The height of slices may vary (left). Admissible slice heights $\mathcal{T} = \{h_0, h_1, \ldots\} \in \Delta\mathbb{Z}^+$ are a subset of integer multiples of $\Delta$ so that all slice boundaries necessarily align with the grid. The slices cover the object. The first and last slice may extend beyond the shape to accommodate restrictions in the available slice heights, such as in the uniform slicing on the right.

build direction [Wang et al. 2016]; or because of the attachment of support material [Zhang et al. 2015]. Orientation has also been optimized in view of the error due to slicing, either standard uniform slicing [Thrimurthulu et al. 2004; Cheng et al. 1995], or in combination with variable slice thickness [Xu et al. 1997]. Printing directions have also been evaluated in terms of accuracy for a decomposed object such that each part is uniformly sliced in its optimal printing direction [Hildebrand et al. 2013].

We assume the orientation is fixed. This seems to be the case in most practical situations, where the orientation has already been chosen by the user or optimized (for one of the reasons mentioned above). We could also search over a set of discrete orientations to find the orientation that leads to the overall smallest slicing error, potentially using global optimization strategies [Danjou and Köhler 2009]. Our goal, however, is to trade between process time and accuracy. For the same orientation, the number of slices strongly correlates with manufacturing time (see Section 7). However, the same number of slices for different orientations results in very different manufacturing times, as the area and perimeter of the contours change completely. It is unclear whether it would be possible to reliably estimate the time in this more general setting.

## 3. SETUP

For ease of exposition, we describe the approach for a discrete grid aligned with the layer directions. As a running example we will use the 'hourglass'-shape depicted in Figure 1, on which we try to illustrate the main ideas and points of this work. Layers are assumed to be spanned by the first two canonical directions (i.e. "$x$" and "$y$"), and the slicing direction is along the third canonical direction (i.e. "$z$"). Moreover, we assume the grid constant $\delta$ to be the same for $x$ and $y$, while the grid constant $\Delta$ in $z$ may be different. This reflects that most practical devices have significantly different resolution in the layer directions and the slicing direction, while the resolution within each layers is mostly uniform. There is no technical reason for using the same grid constant for the two axes within the layers – we just do so for notational convenience. In this setting, the matrix $D = \mathrm{diag}(\delta, \delta, \Delta)$ connects integer coordinates $\mathbf{z} \in \mathbb{Z}^3$ with locations $D\mathbf{z}$ in $\mathbb{R}^3$.

### 3.1 Shape representation

Let $\mathcal{S}$ be a given shape (bounding a volume) in $\mathbb{R}^3$, conceptually described as an occupancy grid

$$S : \mathbb{Z}^3 \mapsto \{-1, 1\}, \tag{1}$$

where a value of 1 means the grid point is inside the shape. Note that most other representations of closed shapes could be easily converted into this format. This conversion may introduce geometric and topological error. Geometric or topological features that cannot be represented on the grid could not have been manufactured, so the discrete representation causes no loss relative to the physical artifact to be manufactured. The *reach* [Federer 1959] (which has been aptly called *local feature size* in computational geometry [Amenta and Bern 1999]) can be used to relate features to the grid resolution and decide whether a feature could be represented on the grid [Huang et al. 2013].

An important part of our implementation is that we never store the shape as a discrete binary grid in our implementation. Instead, we store sorted lists of boundary positions between inner/outer voxels along each $(x, y)$ column (see Section 4.2).

By an appropriate choice of uniform scaling, which we subsume into the grid constants $\mathbf{D}$, we align the extent of the shape in $z$-direction with the unit interval $[0, 1]$. This also means that the grid constant $\Delta$ along $z$-direction is of the form $N^{-1}, N \in \mathbb{N}$, because the interval boundaries need to be represented as multiples of $\Delta$. In the following, we will 'think' in integer values $z \in \mathbb{Z}$ that relate to positions in $\mathbb{R}$ by the map $D$, as mentioned above (see also Figures 1 and 2). Note that (integer) $z = 0$ maps to (real) 0 and $z = N$ maps to 1.

### 3.2 Slicing

A slicing of the shape $\mathcal{S}$ is an ordered sequence

$$\begin{aligned} &Z = \{z_0, z_1, \ldots, z_n\}, \quad z_i \in \mathbb{Z} \\ &z_0 < z_1 < \ldots < z_n, \quad z \leq 0, z_n \geq N \end{aligned} \tag{2}$$

representing the $n$ slices $[z_i, z_{i+1}]$ that together cover the shape, i.e. $[0, 1] \subseteq \cup_i [z_i, z_{i+1}]$ (see Figure 2). Note that we do *not* require the start of the first slice $z_0$ and the end of the last slice $z_n$ exactly

align with the boundaries of the shape (i.e. $z_0 = 0, z_n = N$). For one, given restricted slice thicknesses it may be impossible to align both the start and the end. More importantly, the exact starting and ending point should be the result of minimizing the error. We do require, however, that the whole shape is covered with slices. Allowing $z_0 > 0$ or $z_n < N$ would imply that the bottom and top of the shape might be not represented with slices, which would mean different parts of the shape are treated differently.

The slices must be elements of a set of available thicknesses $\mathcal{T}$ (which are necessarily multiples of $\Delta$):

$$z_{i+1} - z_i \in \mathcal{T} = \{t_{\min}, \ldots, t_{\max}\} \subseteq \mathbb{Z}^+. \tag{3}$$

The number of elements in $Z$ directly relates to the number of slices $n : |Z| = n + 1$. Given the minimal and maximal thickness of slices, we thus expect to have roughly at least $N/t_{\max}$ and at most $N/t_{\min}$ slices.

### 3.3 Error

Layered manufacturing builds an approximation of the geometry within an interval $[z_i, z_j]$. We assume this geometry is *extruded along the z-direction*. This allows to represent the geometry within a slice with a discrete binary image. We denote the image for an interval as $C_{z_i}^{z_j} : \mathbb{Z}^2 \mapsto \{-1, 1\}$. The approximation of the shape results from using the values $C_{z_i}^{z_j}(x, y)$ for the range $z \in [z_i, z_j]$. If needed, the image $C_{z_i}^{z_j}$ could be converted into a contour polygon using any contouring technique.

Now computing the error in each slice simply means counting the number of mismatches between $S$ restricted to the interval and the replacement geometry. Noting that the difference of the two values is zero if they are equal, while it is $\pm 2$ if they differ, we can measure the per-slice error as

$$E(z_i, z_j) = \frac{1}{2} \sum_{z=z_i}^{z_j - 1} \sum_{(x,y) \in \mathbb{Z}^2} |S(x, y, z) - C_{z_i}^{z_j}(x, y)|. \tag{4}$$

The definition of the slice geometry and the resulting error is illustrated in Figure 3 for two of the slices in the left slicing of Figure 2. Note how different choices for the slice geometry $C_{z_i}^{z_j}(x, y)$ lead to different errors and that using the contour of the slice at the boundary or the middle may not lead to the smallest error within a slice. We will explain how to systematically minimize the error within a slice in Section 4.

This way of defining the error means that the volume of a feature represents its importance. Depending on the application scenario, this may or may not be true. We note that some related work defines the slicing error differently, mostly to accommodate specific problems. For example, Wang et al. [2015] argue that features salient for the human observer should be reproduced more accurately — and therefore use mesh saliency [Lee et al. 2005] as a component of their error metric. This makes sense in the intended application scenario, yet would be inappropriate if shapes are intended to have some mechanical function where precision matters most [Coros et al. 2013; Koo et al. 2014; Bächer et al. 2012].

We believe it is better to start from considering each volumetric element equally and then offer to *weight* the error. The weights could be computed from the shape (such as in the scenarios described above) or be supplied by the user (we demonstrate this in Section 7.3). In either case we assume to have spatially varying weights $w(x, y, z)$ and then define the weighted error as

$$E(z_i, z_j) = \frac{1}{2} \sum_{z=z_i}^{z_j - 1} \sum_{(x,y) \in \mathbb{Z}^2} w(x, y, z) |S(x, y, z) - C_{z_i}^{z_j}(x, y)|. \tag{5}$$

The total error results from summing up over all slices:

$$E(Z) = \sum_{i=0}^{n-1} E(z_i, z_{i+1}). \tag{6}$$

In the following, we will first derive a data structure that allows us to quickly compute the optimal per-slice geometry $C_{z_i}^{z_j}$ and store all errors for any potential slice. Based on this information, we will then compute optimal sequences.

## 4. COMPUTING AND MINIMIZING SLICE ERRORS

In the following section we show how to compute the errors $E(z_i, z_j)$ for all pairs $z_i, z_j$ that represent an allowed thickness, i.e. $z_j - z_i = \tau_{ij} \in \mathcal{T}$. The error depends on the contour represented by $C_{z_i}^{z_j}$. As our goal is to *minimize* the overall error resulting from slicing, we also want to choose this contour such that the resulting error is minimized.

### 4.1 Locality of error and optimal contours

Our key observation is that the error within a slice from $z_i$ to $z_j$ can be computed for each $(x, y)$ position independently (see Eqs. 4, 5). Denote this error as

$$E(x, y, z_i, z_j) = \frac{1}{2} \sum_{z=z_i}^{z_j - 1} w(x, y, z) |S(x, y, z) - C_{z_i}^{z_j}(x, y)|. \tag{7}$$

A single column along $(x, y)$ within a slice is assigned either to the inside or to the outside: $C_{z_i}^{z_j}(x, y)$ remains constant in the summation, and is either 1 or $-1$. Inspecting the two cases and rewriting the result using the following abbreviations

$$S_{z_i}^{z_j}(x, y) = \sum_{z=z_i}^{z_j - 1} w(x, y, z) S(x, y, z) \tag{8}$$

$$W_{z_i}^{z_j}(x, y) = \sum_{z=z_i}^{z_j - 1} w(x, y, z) \tag{9}$$

shows that the error can be expressed as

$$E(x, y, z_i, z_j) = \frac{1}{2} (W_{z_i}^{z_j}(x, y) - C_{z_i}^{z_j}(x, y) S_{z_i}^{z_j}(x, y)). \tag{10}$$

The error is minimized when the product in the second term is positive. We thus set

$$C_{z_i}^{z_j}(x, y) = \operatorname{sgn} S_{z_i}^{z_j}(x, y). \tag{11}$$

This result is quite intuitive, especially in the unweighted case $w(x, y, z) = 1$: if more than half of the elements for one $(x, y)$ position in a slice are inside (resp. outside) then the optimal assignment for this position is inside (resp. outside). The resulting error equals the number of elements that differ from this assignment.

It is insightful to see when and how the resulting contours differ from the common approach of intersecting the shape against the slice boundaries or the mid-level of the slice (see Figure 3). Intuitively, if a slice contains small features, the contour at any fixed

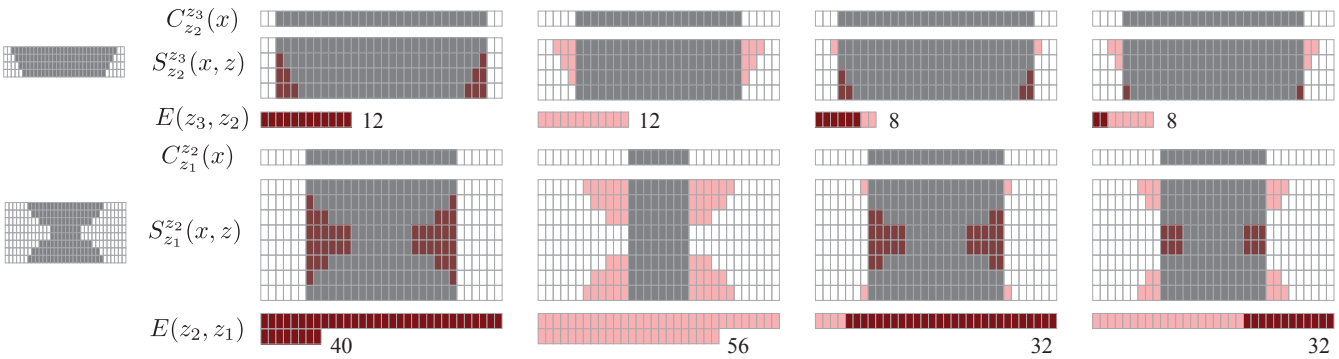


Fig. 3. The geometry of a slice is obtained by extruding its image along the **z**-axis, i.e. slices are approximated to have vertical walls. This results in an approximation error, as some grid cells that belong to the inside of the object are not part of a slice (light red) and some grid cells that belong to the exterior are part of a slice (dark red). The error of a slice is the number of wrongly classified grid cells. Using the midlevel contour for approximation is optimal only if a ray along the **z** direction intersects the geometry not more than once (compare upper vs. lower row). The optimal contour is not necessarily unique (slices on the right).

height may fail to adequately reflect this. The notion of *small feature* in this context relates to local feature size, however, measured only along the slicing direction and how it intersects the slice. We can distinguish different situations based on the number of intersections:

*No intersection:* This means the feature is large relative to the slice and all elements along the ray are inside or all are outside. Intersecting the slice at any position would generate optimal geometric and topological representation.

*1 intersection:* This means the boundary of one feature intersects the slice. As the feature is connected (there is only one intersection), the dominating part contains the center of the slice. Intersecting at the mid-level would generate the same assignment as minimizing the error (see Figure 3, upper row). This assignment would also locally capture the topology of the feature correctly.

*2 or more intersections:* Intersecting the slice at any level generally fails to minimize the volumetric error (Figure 3, lower row, illustrates how the mid-level contour fails to be optimal). Our approach generates assignments that minimize the volumetric error. Topology of the input is generally not captured correctly.

Note how the last case relates the number of intersections with topology. If we wanted to avoid topological problems resulting from slicing we could also assign $E(x, y, z_i, z_j) = \infty$ when more than two intersections occur. As such slices are unavailable this would effectively avoid topological inconsistencies resulting from selecting thick slices: for none or one intersection our assignment is identical to the intersection with the mid-level of a slice, so the topology would be preserved (compare [Huang et al. 2013]). We discuss the issue of geometric accuracy further by relating our approach, and slicing in general, to signal processing in Section 6.

Optimal contours also lead to an important property for the application of dynamic programming for finding optimal slicing sequences (see Section 5): the error of a slice can only increase when we make the slice thicker. More formally, we claim that

$$z'_i \leq z_i < z_j \leq z'_j \quad \Longrightarrow \quad E(z_i, z_j) \leq E(z'_i, z'_j) \qquad (12)$$

for optimal choices of $C_{z_i}^{z_j}(x, y)$ and $C_{z'_i}^{z'_j}(x, y)$. The claim directly follows from optimality. For the same assignment $C_{z_i}^{z_j}(x, y) = C_{z'_i}^{z'_j}(x, y)$ the statement is true because the wrongly assigned grid cells in the thinner slice $[z_i, z_j]$ are a subset of those in the thicker slice $[z'_i, z'_j]$. If this assignment was optimal we are done; if not, picking the optimal assignment will only further reduce the error.

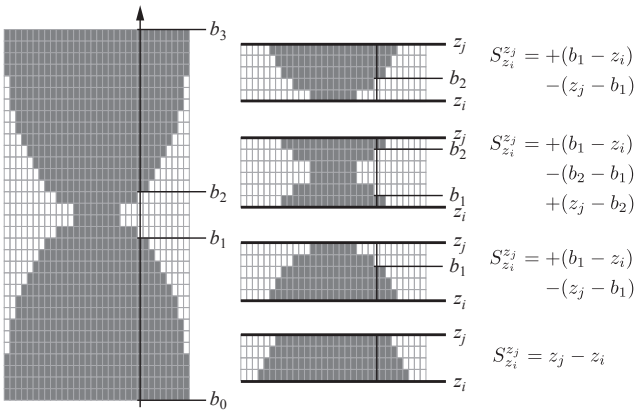


Fig. 4. Per-slice errors can be quickly updated for each location by conceptually intersecting a ray along slicing direction against the boundaries of the shape. Intersections $b_0, b_1, \ldots$ represent changes from outside to inside for even indices and from inside to outside for odd indices. The ray contributes to the error of a slice only if the slice contains any intersection. The computational complexity of computing the error is proportional to the number of intersections, and independent of the number of elements a slice contains.

## 4.2 Efficient evaluation

Our goal is to compute the errors $E(z_i, z_j)$ for all possible pairs $z_i, z_j$ quickly. As just explained, we can perform this computation independently for positions $(x, y)$ and do so in parallel. Accumulation in a global table is done using atomic add operations.

As mentioned before, the shape $S(x, y, z)$ is not available as a discrete grid, as the large memory footprint would significantly degrade performance. Fortunately we can generate information for

the slice errors *locally*, as the slice thicknesses are small compared to the height of the volume.

So consider a fixed position $(x, y)$ (we drop the index to $x, y$ for brevity in the following). We may think of a fixed position as a ray in slicing direction (see Figure 4). If the intersection of ray and a particular slice are completely outside the shape, we assign the $(x, y)$ position in this slice to the outside and the error resulting form this assignment is zero. Similarly, if the intersection is completely inside the shape, we assign the $(x, y)$ position in the slice to the inside and the error is also zero. This means we only need to consider a slice if it contains an intersection of the ray with the boundary.

We start by computing the boundary intersections

$$b_0 < b_1 < \ldots < b_{m-1} \tag{13}$$

for the ray at $(x, y)$ along $z$. Here, $b_0$ is the 'first' intersection of the ray along $(x, y)$ in slicing direction. At this point, the ray changes from being outside the shape to being inside. Consequently, at $b_1$ it changes from being inside to being outside again. Generally, intersection with even indices indicate changes from outside to inside, while odd indices indicate changes from inside to outside. As the ray starts on the outside and ends outside, the total number of events $m$ is even.

Intersections are handled consecutively; let the current intersection be $b_k$. This intersection affects the error $E(z_i, z_j)$ of all slices for which $z_i \leq b_k < z_j$. By indexing the error table based on the lower boundary $z_i$ and height $z_j - z_i$ of a slice and exploiting that heights are bounded by $t_{\max}$ we can directly loop over all candidate slices. It may seem natural to iterate through the intersection $b_k$, iterate through slices, account for the intersection and store the information in a temporary variable. Yet, we have found that is beneficial to avoid recording per-slice information and rather consider each slice only once. To do so, we handle a slice with the first intersection it contains. This just adds the additional constraint $z_i > b_{k-1}$. Together this leads to the loops in lines 2 and 3 of Algorithm 1 below.

So for each slice containing $b_k$, we collect any additional intersections. This means we check if $b_{k+1}$ is also contained in the slice, i.e. $b_{k+1} < z_j$. If it is, we repeat the procedure until we find $b_{k+l+1} > z_j$ or we have reached the last intersection, i.e. $k+l+1 > m-1$. After this step we have identified the intersection points $b_k, \ldots, b_{k+l}$ included in the interval $[z_i, z_j]$.

Note that for fixed $z_i$, looping over available slice heights means we are considering thicker and thicker slices, and each slice contains the previous slice. This means each slice in the loop contains at least the intersection points found for the previous one. So for each slice we only need to check if it contains the next intersection point, and if not we can move on, making this a very efficient procedure.

Based on the slice boundaries $z_i, z_j$ and the included intersection points $b_k, \ldots, b_{k+l}$ we could evaluate the sum $S_{z_i}^{z_j}(x, y)$ following Eq. 8 by summing up over the grid cells. However, we can reduce the complexity from being linear in the discrete height to linear in the number of intersections for uniform weights; and by using an approximation also for non-uniform weights. This is important as the dominant case by far is that the slice only contains one intersection point; more than one intersection would require the slice containing a very thin feature of the shape.

In the case of uniform weights, the value $S(x, y, z)$ along each of the intervals $[z_i, b_k], [b_k, b_{k+1}], \ldots, [b_{k+l}, z_j]$ are constant and either $-1$ or $+1$. This means their contribution to $S_{z_i}^{z_j}(x, y)$ is just plus or minus their thickness, e.g. the contribution of the first interval $[z_i, b_k]$ is $\pm(b_k - z_i)$. The sign depends on the type of intersection: if $b_k$ is a change from outside to inside the interval is outside so the contribution is $-(b_k - z_i)$; or $(b_k - z_i)$ if the interval is inside. Recall that the index $k$ is even for changes from outside to inside, and odd otherwise. This means we can write the contribution of the interval as $(-1)^k(z_i - b_k)$. Extending this idea to all intervals leads to the inner loop detailed in Algorithm 1.

**Algorithm 1:** Error per slice for fixed position $(x, y)$

**Data:** Position $(x, y)$, intersections $b_0, b_1, \ldots, b_{m-1}$ at this position
**Result:** Updated errors $E(z_i, z_j)$

1 **for** $k \in [0, \ldots, m-1]$ **do**
2  **for** $z_i \in [\max(b_k - t_{\max}, b_{k-1}), b_k]$ **do**
3   **for** $z_j \in [b_k, z_i + t_{\max}]$ **do**
4    $S \leftarrow (-1)^k(z_i - b_k)$
5    $l \leftarrow k + 1$
6    **while** $b_l < z_j$ **do**
7     $S \leftarrow S + (-1)^l(b_{l-1} - b_l)$
8     $l \leftarrow l + 1$
9    $S \leftarrow S + (-1)^l(b_{l-1} - z_j)$
10    $E(z_i, z_j) \leftarrow E(z_i, z_j) + (z_j - z_i) - (\operatorname{sgn} S)S$

For the weighted case we suggest to evaluate the weight function only in the heights $z_i, b_k, \ldots, b_{k+l}, z_j$ and use a piecewise linear approximation. This means each interval is weighted by the average weight values, e.g. the contribution of the first interval $[z_i, b_k]$ to $S_{z_i}^{z_j}(x, y)$ is $(-1)^k \frac{1}{2}(w(x, y, z_i) + w(x, y, b_k))(z_i - b_k)$, and so on for the following intervals. The computation of $W_{z_i}^{z_j}(x, y)$ also needs to sum up over intervals, similarly to the ones above except the sign is always positive.

## 5. OPTIMAL SEQUENCES

Based on the table of errors for each potential slice, we wish to compute sequences $Z$ that lead to smallest possible error. The ideal solution in practice may depend on many factors. In some scenarios would like to bound the resulting error; in others it one wants to bound the manufacturing time. We try to offer a convenient solution by computing sequences with minimal error for all possible number of slices.

A related but slightly more restrictive formulation would be to ask for the shortest sequence that satisfies a given error bound [Wang et al. 2015]. We have found two practical problems with this approach: 1) it is difficult to guess the error to achieve the desired effect; 2) even when continuously adjusting the error, not all sequence lengths are available. By providing the best sequence for all possible lengths, it is very easy to pick a sequence with desired error or length.

This problem in general appears similar in flavor to *weakly* NP-complete problems [Garey and Johnson 1979] such as knapsack. Rather than analyzing the complexity of the problem in detail, we show that, like knapsack, it has a pseudo-polynomial time solution using dynamic programming. For the instances we face it is very efficient in practice. Our main idea is to parameterize optimal solutions over the number of slices, i.e. instead of computing only one solution for a given number of slices, optimal solutions for all possible sequence lengths are generated in one pass. This is what sets

our approach apart from other common uses of dynamic programming in this context.

We first give an intuitive explanation of our approach and introduce the necessary notation. Then we provide a practical solution.

## 5.1 Idea and notation

Intuitively, we compute all optimal sequences leading to each $z$-value. The main idea is to iterate over the discrete set of $z$-values: given all optimal sequences reaching $z, z-1, z-2, \ldots$, we generate sequences reaching $z+1$ by considering all possible extensions of the existing sequences.

To make this concrete, let $\tau_m(z)$ contain the thickness of the last slice in the optimal sequence with $m$ elements ending at $z$. If no sequence of $m$ elements exists we set $\tau_m(z) = 0$. First note that this information suffices to reconstruct the whole sequence: starting from $z_m$ we can go backwards iterating

$$z_{i-1} = z_i - \tau_i(z_i) \tag{14}$$

until $\tau_i(z_i) = 0$. This observation means the set $\{\tau_m(z)\}$ for all possible $m$ and $z' < z$ encodes all optimal sequences leading to heights $z' < z$. How can we generate sequences that lead up to $z+1$? For a sequence to be optimal, have length $m$ and end in $z+1$ we need a thickness $t \in \mathcal{T}$ so that $\tau_{m-1}(z-t) > 0$. However, there is usually a choice of different values for $t$ and we wish to pick the one that leads to the *optimal* sequence, i.e. the sequence with the smallest error.

So for each sequence, we also need know its total error $E_m(z)$ up to $z$. Based on this information we choose the optimal thicknesses for each $m$ simply by considering the total error:

$$E_m(z) = E_{m-1}(z-t) + E(z-t, z). \tag{15}$$

Assuming there exists one or more thicknesses $t$ such that $\tau_{m-1}(z-t) > 0$ we pick based on the error above. That this update step indeed leads to optimal sequences requires the property that subsequences of optimal sequences are optimal. This is true whenever thicker slices won't have smaller error. We show this to be true for volumetric error – the approach would more generally work for any error measure satisfying this property.

The result of this approach is a table of thicknesses $\tau$, encoding for each height and sequence length the optimal choice of slice thickness. We have visualized this table by color coding the thickness in Figure 5. The left and right boundaries show the the limits of possible sequence lengths, i.e. the shortest and longest slicing sequence. We also see how different geometric features require different slice thicknesses for the sequence to be optimal.

## 5.2 Implementation

The basic idea explained above directly leads to an algorithm. Important details for the implementation are where to start and end the sequences, initialization, processing for the extension of sequences, and efficient storage of intermediate values. In the end we need to present the computed values to the user and retrieve the desired slicing sequence. We go through these items in order and briefly analyze the computational complexity.

*Limits.* It is possible to fix the starting point to $z_0 = 0$. However, it is not clear that this is the optimal choice, so we keep the starting point more flexible. Since the first slice should intersect the object, we restrict the starting point to $-t_{\max} < z_0 \leq 0$. For similar reasons we restrict the endpoint to $N \leq z_n < N + t_{\max}$.

The longest possible sequence results from using the thinnest slices in the range $0 < z < N$, and then one slice on each end

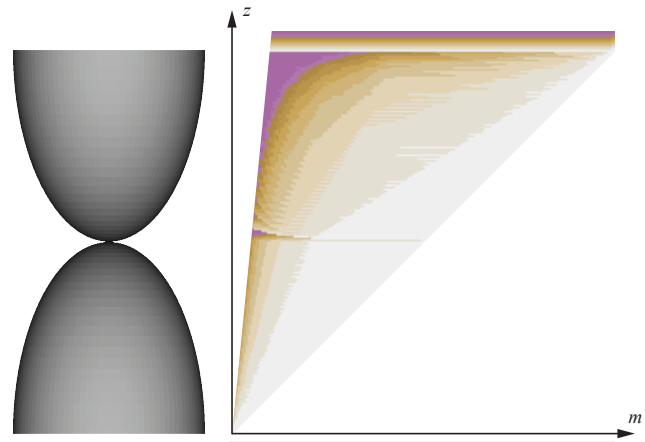


Fig. 5. The table of thicknesses generated by dynamic programming for an hourglass shape. Thickness is color-coded, starting with light gray for the thinnest available slice. The smaller slopes towards the center of the shape require thinner slices. This shows in the table as larger portions of each row being assigned to thinner slices, in particular the larger amount of light gray. The center itself, however, is better covered with one thicker slice.

extending the sequence to include the boundaries. This means $m \leq (N-2)/t_{\min} + 2$. While we could bound $m$ also from below and make the bounds depend on the current height $z$ it turns out that this will be unnecessary for our implementation.

*Initialization.* Given the bounds, we set $E_m(z) = \infty$ in the ranges $m \in [1, \lfloor (N-2)/t_{\min} \rfloor + 2]$ and $z \in [1, N + t_{\max} - 1]$. This means any sequence of length $m > 0$ ending in $z > 0$ that can be build from available slice thicknesses will be accepted. We will use the test $E_m(z) < \infty$ to check whether a sequence exists. If so, this sequence can be extended.

To be able to start the process, we initialize the range $z \leq 0$ with *empty* sequences (i.e. no slice). Since below $z = 0$ there is no shape, using no slice incurs zero error. In other words we set $E_0(z) = 0, z \in [-t_{\max} + 1, 0]$.

*Processing.* We iterate through heights $z > 0$. For each $z$, we try to extend an existing optimal sequence ending before $z$ by adding an admissible slice. This means for fixed $z$ we iterate through sequence lengths $m \in [1, \lfloor (N-2)/t_{\min} \rfloor + 2]$ and admissible slice heights $t \in \mathcal{T}$. For each combination we check

$$E_{m-1}(z-t) + E(z-t, z) < E_m(z). \tag{16}$$

If this test succeeds we set $\tau_m(z) = t$ and $E_m(z) = E_{m-1}(z-t) + E(z-t, z)$. The order and the test are based on the following observations:

—The per slice error $E(z-t, z)$ has been precomputed.

—If no sequence with length $m-1$ ending at $z-t$ exists, its error is infinity. This means $E_m(z)$ will be unchanged only if its value is infinity.

—No specific treatment for the first slice is necessary. If $z - t < 0$ we find $E_0(z-t) = 0$ so we can extend the empty sequence to a sequence consisting of the single first slice.

In this way we iterate as long as $z < N + t_{\max}$. Algorithm 2 provides pseudo code.

*Storage.* In practical tests we found that running times of this algorithm are fine, while storage might cause problems: $E_m(z)$ and

**Algorithm 2:** (Conceptual) Optimal discrete slicing algorithm.

**Data:** Set of admissible slice heights $\mathcal{T} \subseteq \mathbb{Z}^+$, errors $E(z_i, z_j)$ for all possible slices
**Result:** Optimal slicing sequences $Z_n = \{z_{n0}, z_{n1}, \ldots, z_{nn}\}$ for all possible $n$, implicitly stored in $\tau_n(z)$

```
for z ∈ [1 − t_max, ..., 0] do
    E_0(z) ← 0
for z ∈ [1, ..., N + t_max − 1] do
    for m ∈ [1, ..., ⌊(N − 2)/t_min⌋ + 1] do
        E_m(z) ← ∞
        for t ∈ T do
            if E_{m−1}(z − t) + E(z − t, z) < E_m(z) then
                τ_m(z) = t
                E_m(z) = E_{m−1}(z − t) + E(z − t, z)
```

$\tau_m(z)$ are potentially large 2D arrays storing respectively a float (error) and an integer (thickness index).

We make the following important observations that alleviate the storage problem and make the algorithm practical:

—At any $z$, information about the errors $E_m(z)$ is needed only in the range $z \in [z - t_{\max}, z]$. After processing we need to compare the errors for all $z \in [N, N + t_{\max}]$. So at any point we only need at most $t_{\max}$ consecutive rows of values $E_m$. Consequently, we store errors in a ring buffer of length $t_{\max}$. When we move to $z$, errors at height $z - t_{\max} - 1$ are discarded and new errors at level $z$ are initialized.

—We cannot discard thickness information $\tau_m(z)$ because it contains the information to reconstruct the slice sequences. On the other hand, we never need to access this information once it has been created at level $z$. This suggest to compress the vector $\tau_m(z)$ as we progress from $z$ to $z + 1$. Not surprisingly, $\tau_m(z)$ is mostly a function that slowly grows with $m$, meaning the dominating behavior is $\tau_m(z) = \tau_{m-1}(z)$ (see Figure 5). We use straightforward run length encoding to exploit it.

We provide information on timing and memory requirements for practical applications together with the resulting slicing information in the next section and in Table I.

*Consolidation.* After all heights have been processed, the range $z \in [N, N + t_{\max} - 1]$ contains optimal sequences that cover the whole shape. So the minimal error that can be achieved for a sequence with $m$ slices is $\min_{z \in [N, N+t_{\max}-1]} E_m(z)$ . We can present this information to the user in form of a curve (see for example Figures 13, 14, and 15). This immediately provides an overview over the different sequence lengths available and the resulting error. If necessary, we could also try to estimate the manufacturing time (which is based on a variety of factors, depending on manufacturing technology) and relate errors to time.

Once the user selects the sequence, it can be reconstructed by starting from $z_n = \arg\min_{z \in [N, N+t_{\max}-1]} E_m(z)$ and then visualized.

*Complexity.* The time complexity of this approach directly follows from the three loops: the outer loop visits $O(N)$ different discrete heights; for each height all sequence lengths are considered, so this loop is bounded by $O(N/t_{\min})$; and for each combination at most all $\|\mathcal{T}\| \leq t_{\max} - t_{\min}$ slice thicknesses are checked. Together this leads to an asymptotic worst case time complexity of $O(N^2 t_{\max})$, however, we note that the very premise of slicing is $t_{\max} \ll N$.

Whether or not the quadratic time complexity leads to a practical algorithm depends on the constant and realistic values for $N$. We show that the algorithm runs quite fast in practice in Section 7.

Space complexity for the values of $\tau$ is also clearly quadratic in the uncompressed form (just note that Figure 5 shows the stored data). The high coherency in the data makes RLE compression quite effective and we see very modest storage requirements in practice (see Section 7).

## 5.3 Variations and extensions

While we believe the basic version of the algorithm is quite practical, it can be varied or extended to match certain specific scenarios. Note that in any version the dynamic programming approach is *independent* of the error measure. One could simply replace the volumetric error measure with any other error measure – as long as the error is never decreasing with the slice thickness.

*Hard constraints.* It might be desirable that certain heights $z$ are exactly matched with a slice boundary. Examples that come to mind are mechanical pieces that need to match certain measurements; or topological features along the slicing direction that can be enforced by making sure that slices start or end at the boundaries of the feature. While it is possible to use weighting to achieve this goal, we wish to mention that hard constraints are easy to implement and even make the algorithm faster.

To illustrate this let us first consider the cases of enforcing that the first slice starts at the bottom of the shape ($z_0 = 0$) and ends at the top of the shape ($z_n = N$). This can be done by starting the first slice at $z = 0$. In our implementation we could enforce this by setting only $E_0(0) = 0$ and disable all other empty sequences, i.e. $E_0(z) = \infty, z < 0$. For enforcing the last slice to fit the top of the shape we simply stop the iteration at $z = N$, and then only consider the sequences ending at this height.

Any constraint $0 < z < N$ in between top and bottom leads to subdividing the slicing process into different parts: one parts has to end at $z$ and the next has to start at $z$.

*Local error bound.* We have considered global errors, i.e. the sum of per-slice errors. One can also make sure the error of each slice is bounded. The simplest way of doing so is during the error computation by simply discarding the slices that exceed the threshold. It also possible to add a check for the local error into the slicing procedure. We wish to stress that bounding only the local error is still a global problem.

*Inter-slice coherence.* In our approach we have considered slices to be independent – each slice can be used independent of the choice of other slices. There may be cases where this is not true. Consider, for example, that there are two (or more) ways to generate geometry for a slice, resulting in different topology. This would mean that adjacent slices need to fit this choice of topology.

We could implement this by also iterating over the different topological choices, just as we currently consider, at each level $z$, different lengths $m$. This would allow generating slice sequences that have different global topology, but each of them would be consistent along the sequence.

*Streaming.* The data in digital manufacturing can be massive and might be too large to fit into main memory. It is trivial to process the shape in the error tcomputation step in a streaming fashion: if streaming has to happen along slicing direction it should be pro-

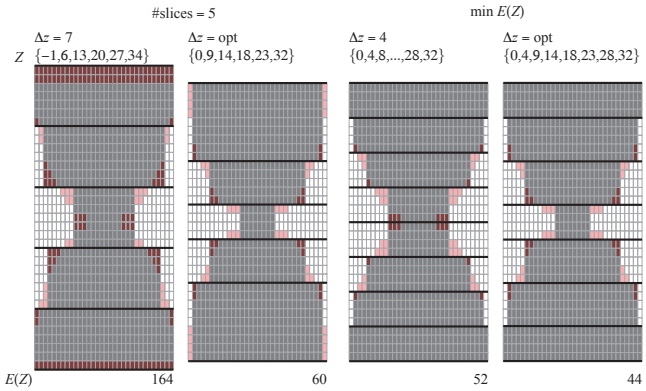


Fig. 6. Slicing the hourglass shape – uniform vs. optimal. The left results are generated with a target of 5 slices. Uniform slicing leads to overestimating the shape, resulting in large volumetric error. Optimal slicing provides a significantly better result using the same number of slices and a minimal thickness of 4. On the right we show the best uniform result for a minimum slice thickness of 4 as well as an optimal slicing having smaller error while using one slice less.

cessed in chunks. Otherwise it is better to process it along one of the other two directions.

Once the errors have been computed, generating the slicing sequence can be performed with rather low main memory requirements. As noted, the sequences $\tau_m(z)$ can be compressed, and since they are not needed during processing, they might as well be transferred to secondary storage. Once a sequence is selected, data is accessed in sequential order again.

## 6. ANALYTIC COMPARISONS

In this section we perform an analysis of slicing algorithms based on the volumetric error. Furthermore, we relate slicing to sampling in order to make statements that are independent of the type of error one considers.

As a running example for this section we use the small discrete hourglass shape discretized to 32 levels. For the available slice heights we assume $\mathcal{T} = \{4, \ldots, 16\}$, i.e. the smallest available thickness is $t_{\min} = 4$ and thickest slice is $t_{\max} = 16$, with all heights between the two extreme values being available. Our goal will be to generate good sequences with 5 slices, or the sequence with smallest error possible.

Later, in the Section 7 we manufacture common shapes on several devices, with the parameters derived from technical specifications. For some of these examples we provide plots of the discrete volumetric error against the number of slices (see Figures 13 and 15) and/or visualizations of the volumetric error resulting from the different slicing strategies. These illustrations show that the effects discussed here for toy examples do carry over to real cases.

### 6.1 Optimal vs. uniform slicing

Dividing the 32 levels into 5 uniform slices means the slice thickness should be 7. In addition, while it is common to align the first slice with the lower boundary of the shape and potentially compromise on the top, this choice seems arbitrary and the error could also be distributed to both the top and the bottom. Figure 6 shows the two best solutions for slices of uniform thickness 6 and 7. Note that for slices of thickness 6 the solution $Z = \{0, 6, 12, 18, 24, 30\}$, which does align the first slice with the bottom of the shape is among the optimal solutions, whereas for slices with thickness 7 this alignment has a larger error than the shown solution $Z = \{-1, 6, 13, 19, 27, 34\}$.

The optimal solution outperforms uniform slicing for this task by a large margin (see Figure 6, center). The reason for the bad performance of uniform slicing is mostly the misalignment of the boundaries of the shape with the slices. While this effect gets smaller as the shape becomes larger relative to the slice thickness, it may still be relevant: in Figure 10 we show the volumetric error for a gear model. The error mostly results from misalignment at the top and bottom. As a result, the error is wildly varying for different number of slices. The optimal result quickly achieves alignment and is then optimal for a wide range of slice heights.

It may seem from these examples that the non-optimal results of uniform slicing could be remedied by making sure that the height of the shape is multiple of the slice height, as this would avoid the large errors at the beginning and end of the sequence. Yet, the problem is more subtle: the problems of uniform slicing stem from misalignment of features that have small slope at any level. To see this, consider the best finest uniform slicing, i.e. using 8 slices of thickness of 4. This sequence perfectly aligns at the top and bottom. It may be surprising that it is still not the best possible solution and that, in fact, using less but thicker slices leads to smaller error (see Figure 6, right). The reason is that uniform slicing fails to align well to the middle part of the hourglass, where it is better to place the slice symmetrically over the thinnest elements.

### 6.2 Adaptive slicing

One might think it is natural that adaptive slicing outperforms uniform (although the finest uniform slicing is commonly claimed to be optimal in the literature). Other adaptive techniques have been proposed, some of which may appear simpler to understand or implement, and perhaps might be faster. We wish to compare to the two main classes of greedy adaptive techniques:

—Fine-to-coarse strategies start from the finest available slicing and selectively combine consecutive slices [Hayasi and Asiabanpour 2013]. Our version of this strategy combines the pair of consecutive slices that leads to the *smallest increase* in error – unlike most methods in the literature that are based on the errors of the slices to be combined. This strategy is illustrated for the example task in Figure 7, left. In practice, combination of consecutive slices would have to be restricted to the available slice thicknesses. For reference we denote this strategy as F2C.

—Coarse-to-fine strategies start from the coarsest possible slicing and then selectively refine slices [Sabourin et al. 1996; Tyberg and Bøhn 1999]. We have implemented two variants of this, both of which refine the slice that leads to largest *decrease* in error (and not the slice with the largest error regardless of the effect of the refinement). The simpler version only considers subdividing a slice into two slices of equal height (or differing by one unit if the slice height is odd) and is denoted C2F. The general and more costly version considers all possible subdivisions of a slice and is denoted C2F*. The latter version is illustrated on the right in Figure 7.

Both coarse-to-fine and fine-to-coarse can match a large range of sequence lengths exactly, and our analysis and experiments show that they significantly reduce the error compared to uniform slicing. Yet, the illustration in Figure 7 clearly shows that both of them suffer from any potential problem in the initial slicing sequence, which is commonly chosen to be uniform. For coarse-to-fine, the eventual adaptive slicing contains the initial sequence of slice boundaries as

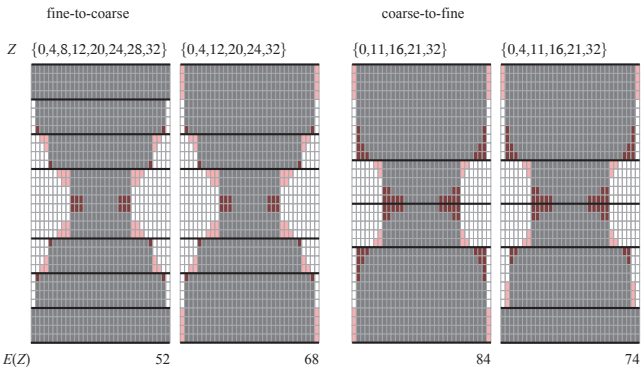


Fig. 7. Slicing the hourglass shape with adaptive techniques. For the fine-to-coarse strategy (left) we combine the pair of consecutive slices that leads to the smallest increase in error. For coarse-to-fine (right), we consider all possible subdivisions of each slice and pick the one that leads the largest decrease in error. Note that the results for 5 slices are not optimal, because both strategies are limited by the initial uniform sequences (consisting of 8 resp. 2 slices).

a subset, i.e. any slice boundary present in the initial sequence will also be present in the refined sequence. For fine-to-coarse, the resulting sequence is a subset of the initial sequence, i.e. adaptation is limited to choose among the initially available slice boundaries.

In both cases this limits the alignment to features (e.g. the small but symmetric neck of the hourglass). This means, if any of the slice boundaries in the initial slicing 'miss' a feature (for example a flat top such as in the Lego brick example), the subsequent refinement or coarsening can never recover and the feature will not be represented in an optimal way. More generally, making local and independent decisions throughout the process will fail to match slice boundaries to features (see also the discussion on sampling below).

While the effects are exaggerated in the toy example, they demonstrate our general findings with these strategies. Based on the larger scale examples in Section 7, we find that the greedy adaptive techniques are consistently outperformed by optimal slicing. Adaptive slicing is essentially limited by the available slice heights:

—Two slices, which are together thicker than the thickest available slice, cannot be merged. This makes it difficult for fine-to-coarse strategies to freely 'place' thick slices where possible.

—A slice less than twice the thinnest slice cannot be subdivided. This makes it difficult for coarse-to-fine strategies to match features with the thinnest slices.

—If slice boundaries in the initial slicing are less than the thinnest available slice from an important feature, this misalignment cannot be corrected throughout the process. This means, both strategies are effectively limited by the smallest available slice height $t_{\min}$.

These effects are clearly visible in the plots provided in Figures 13, 14 and 15. In contrast, optimal slicing can match features at the resolution of $\Delta$, as long as the features are separated. The fact that $\Delta$ is much smaller than $t_{\min}$ explains why optimal slicing can provide an advantage over greedy adaptive techniques. The experiments also show that optimal slicing is very fast in practice (we discuss this in more detail Section 7).

We also find that the optimal algorithm is not substantially more complicated than the greedy heuristics.

### 6.3 A signal processing view of slicing

A claim commonly found in the literature as well as apparently generally accepted in the community is that uniform slicing at the finest level (i.e. the thinnest available slice) would necessarily yield the best result [Pandey et al. 2003b]. While we have shown that this clearly not the case for volumetric error, one could still speculate that for other metrics this might be true. Here we relate slicing to sampling and, in this way, suggest why our findings should be expected, independent of the choice of metric.

We may interpret slicing as a *sampling and reconstruction* process, where the signal is given as the contour over $z$. Our assumption that the contour is constant along the $z$-axis within a slice implies that the *reconstruction filter* of this process is a *box function*. In this view, *uniform slicing is regular sampling*.

In particular, taking the intersection of the slice boundaries or mid-level of a slice with the shape to generate contour corresponds to *point sampling* of the signal. It is well understood that point sampling causes aliasing artifacts, which is what we see in Figure 3. Our approach of minimizing the volume error (Section 4) can be interpreted as using a box kernel for sampling, which optimally matches the reconstruction filter.

Regardless of the kernel used, uniform sampling causes aliasing if the frequency of the signal is higher than the sampling rate, which translates to the shape having features that are smaller than the minimal thickness of slices. Unless the slice could be made thinner than the smallest feature, thinner slices may well result in more aliasing artifacts. Note how our results for uniform slicing clearly demonstrate this phenomenon — the error may increase significantly for particular slice thicknesses, including the thinnest ones available.

One may view coarse-to-fine or fine-to-coarse strategies as dyadic (or generally n-adic) wavelet constructions. While this is indeed an optimal way to balance the number of coefficients (slices) and the quality of the approximation for *smooth* signals, it may be arbitrarily far from optimal in the presence of singularities (i.e. edges in the shape).

The only way to generate optimal results for arbitrary signals is to adapt the basis functions (i.e. the local widths of the analysis and reconstruction filters) to the input. Our approach could be related to approximating a (piecewise constant) function with another arbitrary piecewise constant function. For other error norms and without considering restrictions on the width, Konno & Kuno [1988] show that optimal solutions can be constructed using dynamic programming (in fact, we believe a greedy approach would have sufficed in the max-norm regime). One may view our algorithm as a generalization of their approach to the situation where the widths cannot be chosen arbitrarily: the maximal and minimal slice height are restricted.

## 7. EXPERIMENTS AND RESULTS

Now that we have shown that our approach has significant theoretical advantages, we focus on evaluating whether the gain in theoretical measures such as number of slices and volumetric error indeed translates into manufacturing time and quality of the manufactured shapes.

As manufacturing technologies we consider fused deposition modeling (FDM) and stereolithography (SLA). The particular devices being used are an Ultimaker2 for FDM and Autodesk Ember for SLA. Our approach for setting the discretization is to use manufacturer information on how the motors are controlled or what accuracy could possibly be achieved. Then we either match the in-

ternal discretization or simply use a fraction of the promised accuracy.

The height is controlled on both devices by stepper motors. For the Ultimaker we find that the motor makes a step of $1.875\mu m$, while the Autodesk Amber makes steps of $1\mu m$. We use these values to set the grid constant $\Delta$ in slicing direction. This means every height value we consider in our computations is at least theoretically matched by the hardware (the actual precision in height is usually less due to material properties and environment conditions). Using a smaller value for $\Delta$ would mean that we are considering and generating values that cannot be achieved by the devices and would be mapped to a coarse grid before actuation.

In addition, we have manufactured an artifact by cutting slices out of plywood and then glueing the slices together. Here the available slice thicknesses are given by the available plywood. We set $\Delta = 0.1mm$, as the inaccuracies in the plywood and the assembly process are likely an order of magnitude larger.

For $\delta$ we we use different strategies for the two devices: the Autodesk Amber projects a pattern onto the resin using a projector. Here we use the size of the pixels on the resin, which are reported to be 25 $\mu m$. For the Ultimaker we consider the size of the nozzle, which is 400 $\mu m$. By taking $\delta = 50$ $\mu m$ we make sure that the resolution of our computation is well below the resolution within the layer that could conceivably be achieved. We use a similar value for the plywood example for convenience.

Table I provides further details on the range of parameters in the examples, such as physical dimensions, number of slices, and computational resources. The performance numbers were generated using a 2,8 GHz Intel Core i7 processor with 16 GB RAM and a AMD Radeon graphics card with 2GB GPU memory. Error computation and slicing strategies have been implemented as part of IceSL, a GPU-based modeling and slicing tool [Lefebvre 2013]. It directly generates gcode for several common devices and gave us direct control over layer thickness and path generation from the contour (FDM) or slice image (SLA).

Figures 13, 14 and 15 provide the results of the practical evaluation: we start by uniformly slicing an object with thin slices (left column). Then we match the volumetric error using optimal slicing, with optimal slicing requiring much fewer slices for the same error. As can be seen, fewer slices indeed result in faster printing times. We try to match the faster printing time using uniform slices, which are necessarily much thicker and indeed lead to visible artifacts (right column).

Based on this set of examples we discuss the issues of computational resources (memory and time), the relation of number of slices and actual process times, and visual quality of the results relative to the volumetric error and how the error could be adapted.

## 7.1 Computational resources

Recall that the worst case complexity of the algorithm is quadratic in the number of discrete heights $N$. The important finding from practical scenarios is that the algorithm is quite fast and the compression of the table leads to a very small memory footprint.

Figure 8 shows computation times and memory consumption for a range of heights for fixed model (3DBENCHY) and FDM parameters. Memory consumption is shown in a log-log-plot – interestingly, compression not just drastically reduces memory consumption, it also reduces the growth rate from quadratic to linear.

Increasing the height for fixed parameters is roughly similar to decreasing $\Delta$, but not necessarily exactly identical: note that increasing the height leaves the size of $\mathcal{T}$ unchanged. If $\mathcal{T}$ changes when $\Delta$ changes depends on the type of manufacturing device, i.e. whether the number of available slices is limited only by resolution or by other factors.

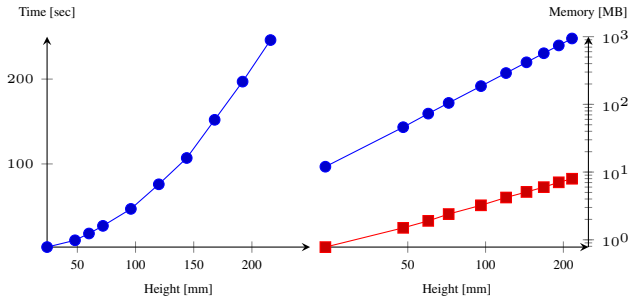


Fig. 8. Computational resources relative to the resolution for the 3DBENCHY. The left plot shows computation times and the right plots uncompressed and compressed data, both relative to the height of the shape for fixed FDM parameters. Adjusting the height for fixed $\Delta$ is roughly equivalent to varying $\Delta$ for fixed height. Note that storage requirements also change from quadratic in the uncompressed case to linear for the compressed version.

The error table computation is fast – on the order of seconds. As mentioned before, this part of the process can be easily performed offline and in a streaming fashion.

Overall computation and memory are not a limiting factor of our approach for the typical parameters of current 3D printing devices.

## 7.2 Manufacturing times

While the literature on adaptive slicing commonly states that fewer slices lead to reduced printing times, we are not aware of actual comparisons. We provide such numbers as part of our evaluation.

We find that, indeed, the number of slices generally influences the printing time: an increasing number of slices resulting in an increased printing time. This is due to the mechanical nature of the process. Each layer takes some time to process. The effect of the number of slices on the printing time largely depends on how the thickness of each layer affects the printing time, and this varies with the printing method.

In the case of FDM the speed is typically kept constant regardless of layer thickness, as it is chosen to avoid oscillations and positioning errors. This means the number of slices directly correlates with the production time. This is reflected in the times reported in Figures 13 and 15. The variation in times relative to the number slices is due to different paths of the nozzle in each layer, which is a factor that is independent of the thickness but has significant impact on the time. This factor would be very difficult to control.

Stereolithography (SLA) has certain exposure times for each slice that depend on the layer thickness and a constant time to get the next layer ready for production. The Autodesk Ember takes a constant time of $\sim 4$ seconds to adjust the system to process the next layer, with exposure times varying between 2.5 seconds for 25 $\mu m$ and 4.0 seconds for 100 $\mu m$. The expected improvement in processing can indeed be seen when comparing fine and coarse uniform slicing (see Figures 13 and 14). Exposure times are not linear, which explains why optimal slicing is faster for a slightly larger number of slices compared to coarse uniform slicing.

## 7.3 Volumetric error and resulting quality

As can be appreciated in the close-ups in Figure 13, 14 and 15 our approach results in better precision and better reproduction of surface detail than uniform slicing, for the same print time and same

Table I. Parameters and computational resources for the practical tests.

| Shape | Method | Z | slice heights $\mathcal{T}$ | | $\delta$ | $\Delta$ | error table computation | | | optimal slicing | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $[mm]$ | $\lvert\mathcal{T}\rvert$ | $[mm]$ | $[mm]$ | $[\mu m]$ | #slices | [MB] | $[sec]$ | [MB] | $[msec]$ |
| Robot | FDM | 29 | 107 | 0.1-0.3 | 0.05 | 1.875 | 1687500 | 12.8746 | 5 | 8/0.75 | 1838 |
| Gear | FDM | 13.5 | 107 | 0.1-0.3 | 0.05 | 1.875 | 797796 | 6.0867 | 17 | 2 0.634 | 411 |
| Tardis | FDM | 33.6 | 107 | 0.1-0.3 | 0.05 | 1.875 | 1954152 | 14.909 | 4 | 11 / 0.611 | 2513 |
| Lego | FDM | 11.3 | 80 | 0.05-0.2 | 0.05 | 1.875 | 501309 | 3.82468 | 1 | 2.73 / 0.18 | 420 |
| Knight | FDM | 100 | 107 | 0.1-0.3 | 0.05 | 1.875 | 5777352 | 44.0777 | 44 | 103/ 5.13 | 23155 |
| Castle | SLA | 16.3 | 75 | 0.025-0.1 | 0.025 | 1 | 1252784 | 9.55798 | 1 | 20 / 1.07 | 3215 |
| Octo | SLA | 11.3 | 74 | 0.026-0.1 | 0.025 | 1 | 866552 | 6.61127 | 1 | 10 / 0.9 | 1503 |
| 3dbenchy | SLA | 14.4 | 74 | 0.026-0.1 | 0.025 | 1 | 1102152 | 8.40875 | 1 | 16/0.7 | 2526 |
| Ghost | (FDM) | 28.5 | 80 | 0.05-0.2 | 0.05 | 1.875 | 1239948 | 9.46005 | 3 | 16/1.4 | 2725 |
| Venus | (FDM) | 193 | 107 | 0.1-0.3 | 0.1 | 1.875 | 11300796 | 86.2182 | 19 | 394/8.4 | 88178 |

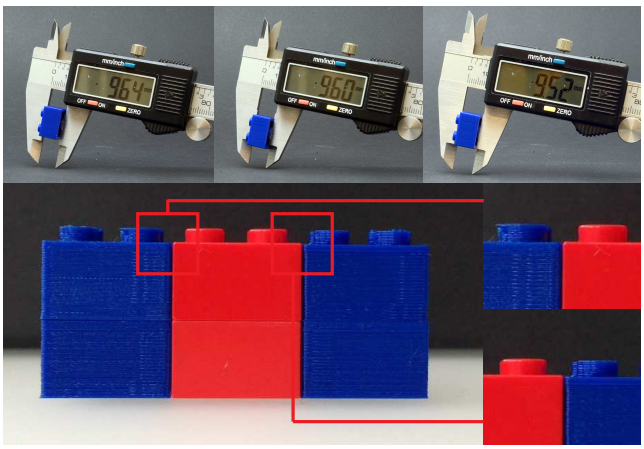

Fig. 9. Top: Uniformly sliced Lego brick with layer height of $0.099375mm$ (Left). Adaptively sliced brick using layer height between $0.05 - 0.2mm$ (Middle). Uniformly sliced brick with layer height of $0.19875mm$. Bottom: Optimal sliced and 3D printed Lego bricks (left) reproduces the height of real bricks (middle) more closely than uniformly sliced ones (right) printed in the same time.

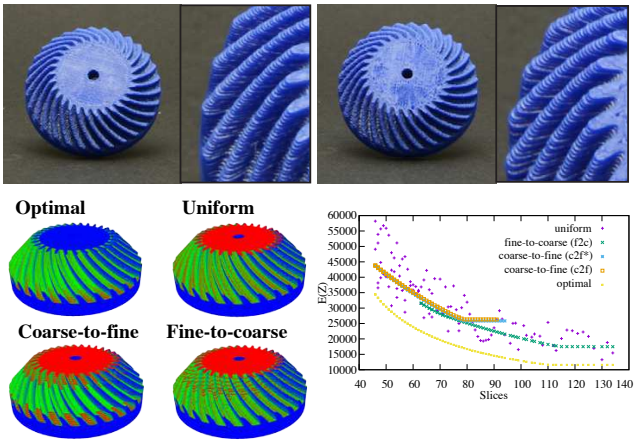


Fig. 10. Top: (Left) Optimally sliced gear model with layer heights between $0.1 - 0.3mm$ and $E(Z) = 22007.85$. (Right) Uniformly sliced with a layer height of $0.214$ and $E(Z) = 39122.30$. Both prints have 64 slices. Bottom: The error maps illustrate that optimal slicing does not generate errors in the top region of the model where other adaptive schemes fail to faithfully reproduce the correct height of the model.

print settings. The benefit is observable for both FDM and SLA prints.

Our approach focuses mainly on precision. In some cases, however, improved precision is uncorrelated with visual or perceptual properties. In particular, on FDM prints accumulations of thick slices can result in visible changes of the surface reflectance properties (see e.g. the close-ups for the Knight model). We speculate that manufacturers have meticulously optimized devices for the common uniform slicing and that some of the visual problems of adaptive slicing may well be remedied by proper adjustment of parameters inaccessible by the user.

Visual appearance of the shape is only one of several properties. Geometric precision is important in many areas where several shapes are stacked together [Luo et al. 2012; Chen et al. 2015; Song et al. 2016], or where they have functional purpose [Coros et al. 2013; Skouras et al. 2013; Koo et al. 2014]. We demonstrate this with the example of a manufactured Lego brick. Lego bricks have an exceptional production tolerance of $2\mu m$ around the raster height of $9.6mm$ (brick without the connectors). We show that by using optimal slicing it is possible to reproduce the production heights faithfully. Using an uniform slicing approach will give good results only if one chooses $\tau$ so that it adds up exactly to the correct height. Otherwise the manufacturing will introduce errors.

We show that the error is visible and significant by measuring the height of the manufactured Lego bricks (top row images of Figure 9). One can see that the uniformly sliced versions with layer heights of $0.099375mm$ (left) and $0.19875mm$ (right) significantly differ from the desired height while the optimal one matches the desired result. Similar outcomes are illustrated for the gear example in Figure 10. The error maps of the model show that other approaches fail to reproduce the correct height of the model and introduce large errors in the top part of the model.

In most cases, the gain of optimal slicing over the finest uniform sequence is small and may not be recognizable on manufactured shapes. We provide an example that demonstrates the advantage of optimal slicing (Figure 11): the shape is manufactured by laser cutting sheets of plywood, than glueing them together, and finally coating the object with spray paint to hide the layer structure. Plywood is available in only a few restricted heights – we used poplar in thicknesses of 4,6,8, and 10mm. Optimal slicing results in smaller error using fewer slices. The smaller number of slices indeed resulted in slightly faster assembly. More importantly, uniform slicing performs worse than optimal slicing in aligning the features with the slicing sequence – and this is clearly visible in the resulting shapes.

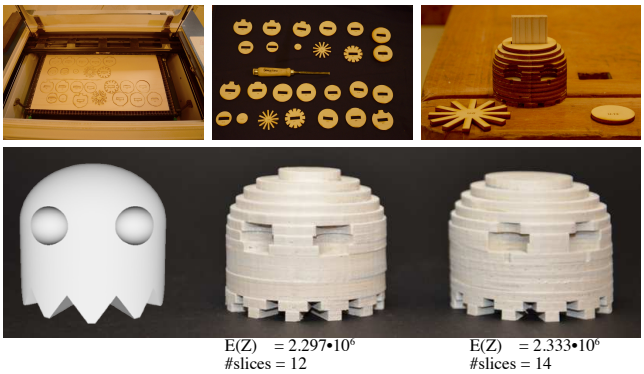


Fig. 11. Shape manufactured by cutting pieces from plywood that are stacked, glued, and spray painted to generate the physical artifact. The final result on the left shows optimal slicing, using 4,8 and 10mm sheets. Finest uniform slicing (final result on the right) only uses the 4mm plywood. Optimal slicing has slightly smaller volumetric error and better reproduces the desired height of the shape. We also prefer the visual appearance of optimal slicing, as it better reproduces the eyes and the round top.

Measuring the total error per slice, regardless of how the error is defined, has a fundamental problem: all features within a slice are treated equally, and the reproduction of details may depend on other parts of the shape present in the same slice.

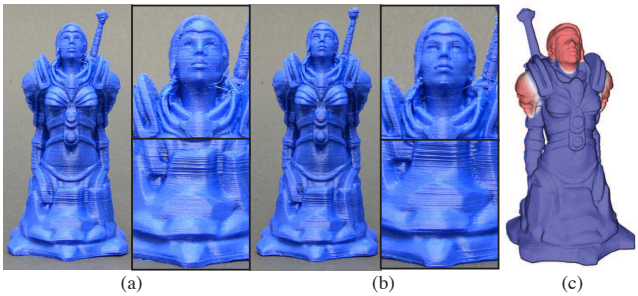


Fig. 12. Importance map specifies where the slicer has to use thinner slices to lower the error in these areas. This example consists of 550 slices with slice heights between 0.1mm to 0.3mm. (Left: Adaptive slicing without importance values, Middle: Adaptive slicing including importance map. Right: User-defined importance map.

We strongly believe that there is no error measure that will lead to the desired behavior in all scenarios. It seems equally flexible to adjust the volume error using the weights introduced in Section 3.

For generating this weight function we could use scenario-driven information such as saliency or mechanical analysis. As automatic generation of importance information may not always be possible, we show how importance defined by the user is incorporated in the slicing process: we offer a minimal brush tool for creating the importance map. One example is shown in Figure 12. This map is used as a simple error weight in the computation of $E(z_i, z_j)$. One can see how this weighting distributes the given set of 550 slices based on the indicated importance. More slices are allocated at the top of the shape, resulting in increased resolution on the head, while fewer slices are used at the bottom.

## 8. CONCLUSION AND OUTLOOK

By formulating the slicing problem in a discrete setting and measuring error by volume, we can provide optimal solutions for both, computing the contour in each slice and computing the sequence of slices. To our knowledge, this is the first approach to slicing that guarantees to provide optimal results in a clearly specified sense; in particular, it is the only approach to our knowledge that also considers the effect of the contour in each slice on the error. The resulting algorithms are natural and efficient in practice. As we have demonstrated, the theoretical advantages do have an impact also on practical results.

Depending on the manufacturing technology, volumetric error correlates more or less strongly with visual or more generally, perceptual quality. Establishing such connection for specific manufacturing technologies is an interesting research problem in its own. Our approach can conveniently adapt to steering the slicing with saliency data by supplying it as an importance map.

Variable slicing might be used in conjunction with techniques that use different materials in different layers, such as to control mechanical properties [Bickel et al. 2010] or color [Hergel and Lefebvre 2014; Reiner et al. 2014; Brunton et al. 2015]. In fact, varying the scale height in these approaches would make them considerably more flexible.

Finally we note that recent techniques that optimize support struts [Dumas et al. 2014; Schmidt and Umetani 2014] could benefit from optimal slicing, by making sure they attach at exactly the right height. Furthermore, printing accuracy would be important for prints that try to achieve optical goals, e.g. [Schwartzburg et al. 2014], or where several printed parts need to be assembled to a larger object [Luo et al. 2012; Chen et al. 2015; Song et al. 2016].

## REFERENCES

AMENTA, N. AND BERN, M. 1999. Surface reconstruction by voronoi filtering. *Discrete & Computational Geometry 22,* 4, 481–504.

BÄCHER, M., BICKEL, B., JAMES, D. L., AND PFISTER, H. 2012. Fabricating articulated characters from skinned meshes. *ACM Trans. Graph. 31,* 4 (July), 47:1–47:9.

BICKEL, B., BÄCHER, M., OTADUY, M. A., LEE, H. R., PFISTER, H., GROSS, M., AND MATUSIK, W. 2010. Design and Fabrication of Materials with Desired Deformation Behavior. *ACM Trans. Graph. (Proceedings of Siggraph 2010) 29,* 4 (July), 63:1–63:10.

BRUNTON, A., ARIKAN, C. A., AND URBAN, P. 2015. Pushing the limits of 3d color printing: Error diffusion with translucent materials. *ACM Trans. Graph. 35,* 1 (Dec.), 4:1–4:13.

CHEN, X., ZHANG, H., LIN, J., HU, R., LU, L., HUANG, Q., BENES, B., COHEN-OR, D., AND CHEN, B. 2015. Dapper: Decompose-and-pack for 3d printing. *ACM Trans. Graph. 34,* 6 (Oct.), 213:1–213:12.

CHENG, W., FUH, J., NEE, A., WONG, Y., LOH, H., AND MIYAZAWA, T. 1995. Multiobjective optimization of part building orientation in stereolithography. *Rapid Prototyping Journal 1,* 4, 12–23.

CORMIER, D., UNNANON, K., AND SANII, E. 2000. Specifying nonuniform cusp heights as a potential aid for adaptive slicing. *Rapid Prototyping Journal 6,* 3, 204–212.

COROS, S., THOMASZEWSKI, B., NORIS, G., SUEDA, S., FORBERG, M., SUMNER, R. W., MATUSIK, W., AND BICKEL, B. 2013. Computational design of mechanical characters. *ACM Trans. Graph. 32,* 4 (July), 83:1–83:12.

DANJOU, S. AND KÖHLER, P. 2009. Determination of optimal build direction for different rapid prototyping applications. In *Proceedings of the 14th European Forum on Rapid Prototyping*, A. Bernard, Ed. Ecole Centrale Paris.

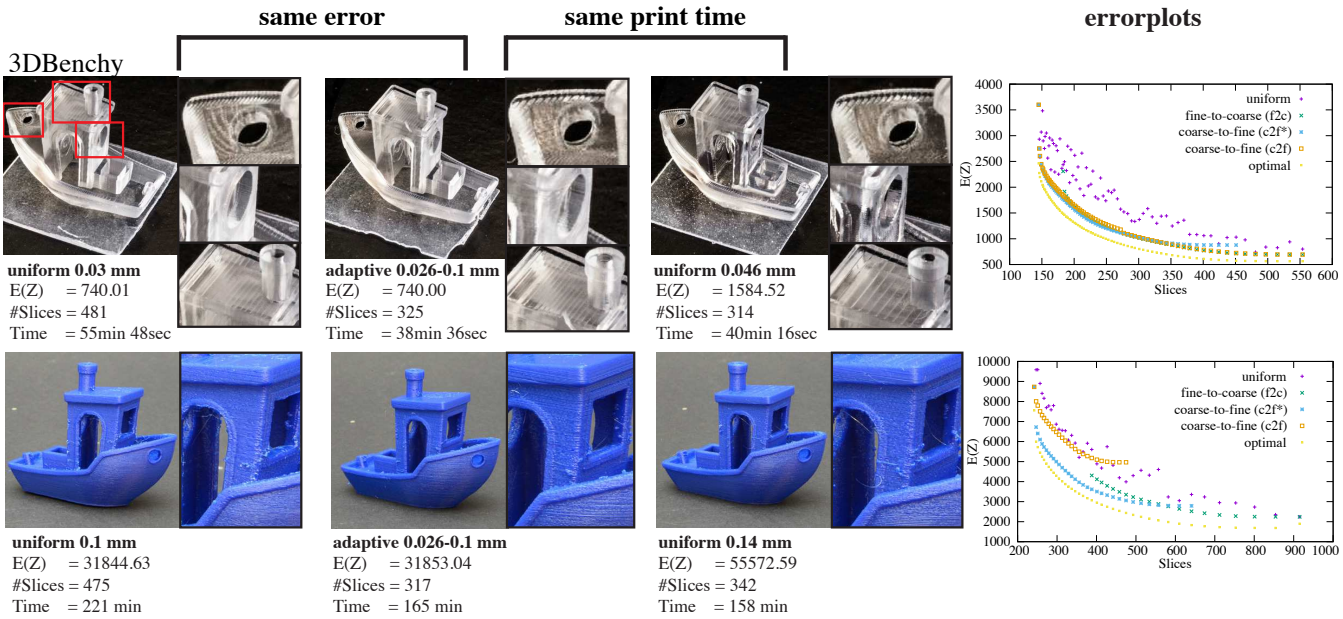


Fig. 13. We compare the same model using different 3D printing technologies. High-resolution uniform slicing (column 1), Optimal discrete slicing with the same error (column 2), Low-resolution uniform slicing with the same printing time (column 3), error plot shows different methods and their slicing errors over the number of slices. Note here, that if the number of possible slice heights is limited adaptive strategies such as c2f fail to generate longer and more accurate slice sequences.

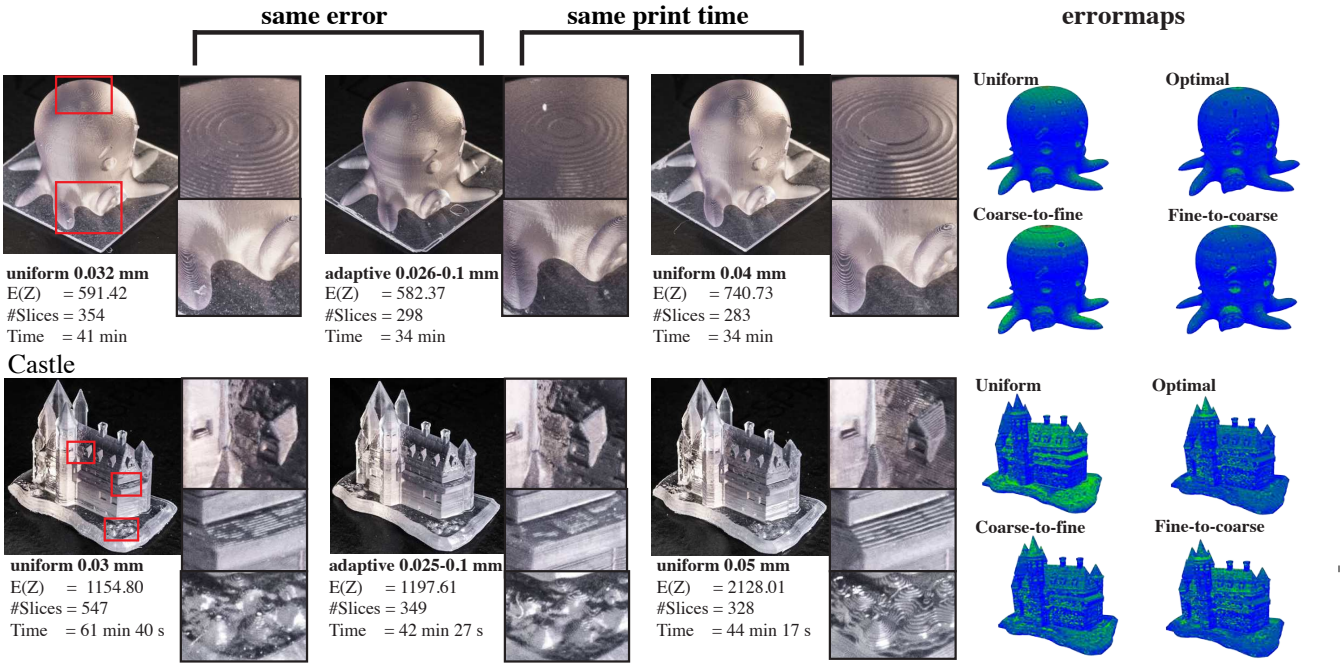


Fig. 14. 3D printed stereolithography results. High-resolution uniform slicing (column 1), Optimal discrete slicing with the same error (column 2), Low-resolution uniform slicing with the same printing time (column 3), error maps show the error for column 2 and 3 results and additionally for a coarse-to-fine and fine-to-coarse slicing approaches.

DOLENC, A. AND MÄKELÄ, I. 1994. Slicing procedures for layered manufacturing techniques. *Computer-Aided Design 26,* 2, 119–126.

DUMAS, J., HERGEL, J., AND LEFEBVRE, S. 2014. Bridging the gap: Automated steady scaffoldings for 3d printing. *ACM Trans. Graph. 33,* 4

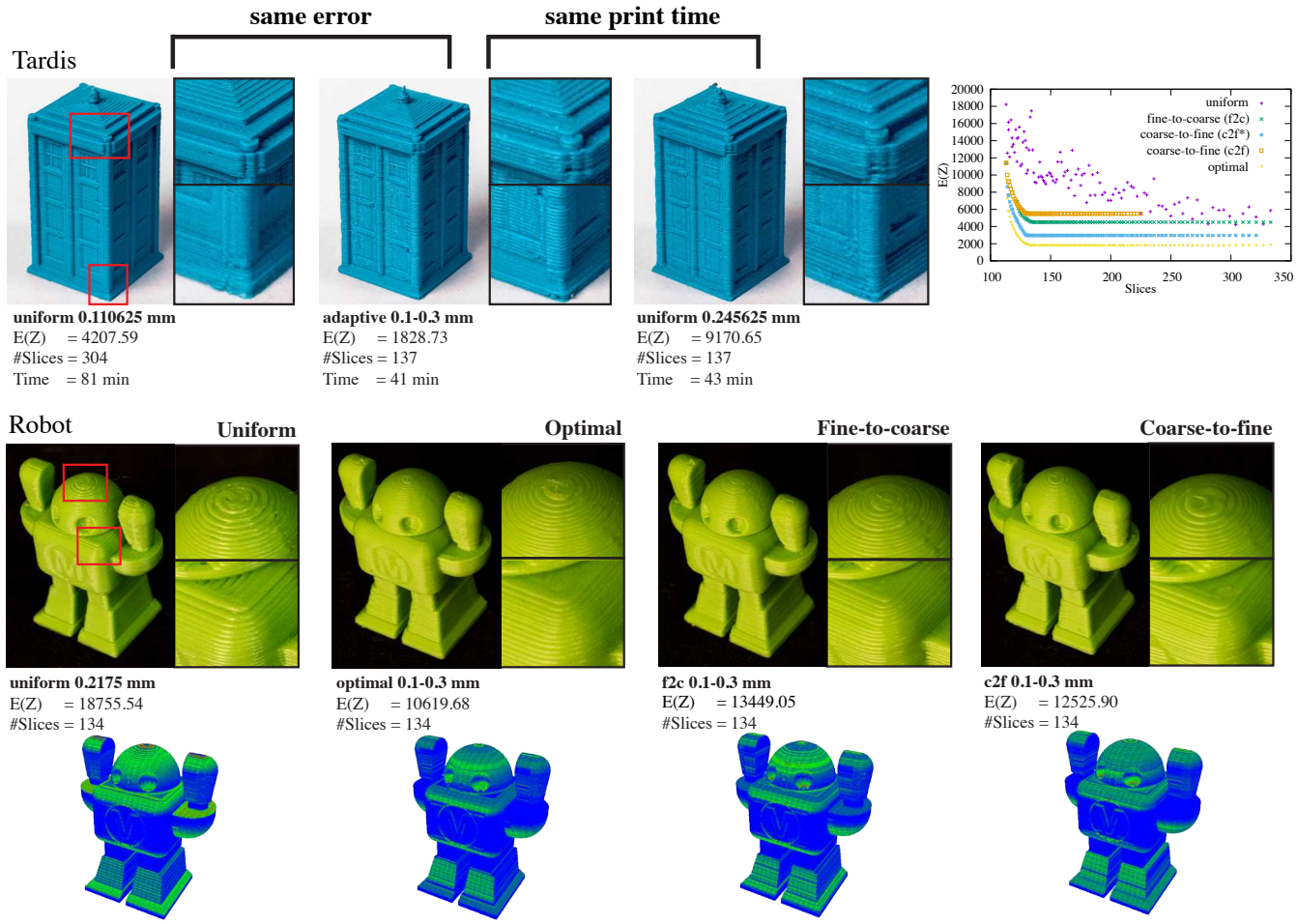


Fig. 15. 3D printed FDM results. Top row: High-resolution uniform slicing (column 1), Optimal discrete slicing with the same error (column 2), Low-resolution uniform slicing with the same printing time (column 3), error plot shows different methods and their slicing errors over the number of slices. Bottom row: Low-resolution uniform slicing (column 1), Optimal discrete slicing printed in the same time (column 2), Fine-to-coarse approach with larger overall error (column 3), Coarse-to-fine slicing (column 4) and their corresponding error maps below.

(July), 98:1–98:10.

FEDERER, H. 1959. Curvature measures. *Transactions of the American Mathematical Society 93,* 3, 418–491.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA.

HAYASI, M. T. AND ASIABANPOUR, B. 2013. A new adaptive slicing approach for the fully dense freeform fabrication (fdff) process. *Journal of Intelligent Manufacturing 24,* 4, 683–694.

HERGEL, J. AND LEFEBVRE, S. 2014. Clean color: Improving multi-filament 3d prints. *Computer Graphics Forum 33,* 2, 469–478.

HILDEBRAND, K., BICKEL, B., AND ALEXA, M. 2013. Orthogonal slicing for additive manufacturing. *Computers & Graphics 37,* 6, 669 – 675. Shape Modeling International (SMI) Conference 2013.

HOPE, R., ROTH, R., AND JACOBS, P. 1997. Adaptive slicing with sloping layer surfaces. *Rapid Prototyping Journal 3,* 3, 89–98.

HU, R., LI, H., ZHANG, H., AND COHEN-OR, D. 2014. Approximate pyramidal shape decomposition. *ACM Trans. Graph. 33,* 6 (Nov.), 213:1–213:12.

HUANG, P., WANG, C. C. L., AND CHEN, Y. 2013. Intersection-free and topologically faithful slicing of implicit solid. *Journal of Computing and Information Science in Engineering 13,* 2, 021009.

KONNO, H. AND KUNO, T. 1988. Best piecewise constant approximation of a function of single variable. *Operations Research Letters 7,* 4, 205–210.

KOO, B., LI, W., YAO, J., AGRAWALA, M., AND MITRA, N. J. 2014. Creating works-like prototypes of mechanical objects. *ACM Trans. Graph. 33,* 6 (Nov.), 217:1–217:9.

KULKARNI, P. AND DUTTA, D. 1996. An accurate slicing procedure for layered manufacturing. *Computer-Aided Design 28,* 9, 683 – 697.

LEE, C. H., VARSHNEY, A., AND JACOBS, D. W. 2005. Mesh saliency. *ACM Trans. Graph. 24,* 3 (July), 659–666.

LEFEBVRE, S. 2013. Icesl: A gpu accelerated csg modeler and slicer. In *AEFA'13, 18th European Forum on Additive Manufacturing*. http://shapeforge.loria.fr/icesl/.

LUO, L., BARAN, I., RUSINKIEWICZ, S., AND MATUSIK, W. 2012. Chopper: Partitioning Models into 3D-printable Parts. *ACM Trans. Graph. (Proceedings of Siggraph Asia 2012) 31,* 6 (Nov.), 129:1–129:9.

MANI, K., KULKARNI, P., AND DUTTA, D. 1999. Region-based adaptive slicing. *Computer-Aided Design 31,* 5, 317 – 333.

MASOOD, H. S., RATTANAWONG, W., AND IOVENITTI, P. 2000. Part build orientations based on volumetric error in fused deposition modelling. *The International Journal of Advanced Manufacturing Technology 16,* 3, 162–168.

PANDEY, P., REDDY, N. V., AND DHANDE, S. 2003a. Real time adaptive slicing for fused deposition modelling. *International Journal of Machine Tools and Manufacture 43,* 1, 61–71.

PANDEY, P. M., REDDY, N. V., AND DHANDE, S. G. 2003b. Slicing procedures in layered manufacturing: a review. *Rapid prototyping journal 9,* 5, 274–288.

REINER, T., CARR, N., MECH, R., STAVA, O., DACHSBACHER, C., AND MILLER, G. 2014. Dual-color mixing for fused deposition modeling printers. *Computer Graphics Forum 33,* 2, 479–486.

SABOURIN, E., HOUSER, S. A., AND BØHN, J. H. 1996. Adaptive slicing using stepwise uniform refinement. *Rapid Prototyping Journal 2,* 4, 20–26.

SABOURIN, E., HOUSER, S. A., AND BØHN, J. H. 1997. Accurate exterior, fast interior layered manufacturing. *Rapid Prototyping Journal 3,* 2, 44–52.

SCHMIDT, R. AND UMETANI, N. 2014. Branching support structures for 3d printing. In *ACM SIGGRAPH 2014 Studio*. SIGGRAPH '14. ACM, New York, NY, USA, 9:1–9:1.

SCHWARTZBURG, Y., TESTUZ, R., TAGLIASACCHI, A., AND PAULY, M. 2014. High-contrast computational caustic design. *ACM Trans. Graph. 33,* 4 (July), 74:1–74:11. Proc. SIGGRAPH 2014.

SINGHAL, S., JAIN, P. K., AND PANDEY, P. M. 2008. Adaptive slicing for sls prototyping. *Computer-Aided Design and Applications 5,* 1-4, 412–423.

SKOURAS, M., THOMASZEWSKI, B., COROS, S., BICKEL, B., AND GROSS, M. 2013. Computational design of actuated deformable characters. *ACM Trans. Graph. (Proceedings of Siggraph 2013) 32,* 4 (July), 82:1–82:10.

SONG, P., DENG, B., WANG, Z., DONG, Z., LI, W., FU, C.-W., AND LIU, L. 2016. Cofifab: Coarse-to-fine fabrication of large 3d objects. *ACM Trans. Graph. 35,* 4 (July), 45:1–45:11.

TATA, K., FADEL, G., BAGCHI, A., AND AZIZ, N. 1998. Efficient slicing for layered manufacturing. *Rapid Prototyping Journal 4,* 4, 151–167.

THRIMURTHULU, K., PANDEY, P. M., AND REDDY, N. V. 2004. Optimum part deposition orientation in fused deposition modeling. *International Journal of Machine Tools and Manufacture 44,* 6, 585 – 594.

TYBERG, J. AND BØHN, J. H. 1998. Local adaptive slicing. *Rapid Prototyping Journal 4,* 3, 118–127.

TYBERG, J. AND BØHN, J. H. 1999. Fdm systems and local adaptive slicing. *Materials & design 20,* 2, 77–82.

UMETANI, N. AND SCHMIDT, R. 2013. Cross-sectional Structural Analysis for 3D Printing Optimization. In *SIGGRAPH Asia 2013 Technical Briefs*. SA '13. 5:1–5:4.

VIDIMČE, K., WANG, S.-P., RAGAN-KELLEY, J., AND MATUSIK, W. 2013. Openfab: a programmable pipeline for multi-material fabrication. *ACM Trans. Graph. (Proceedings of Siggraph 2013) 32,* 4 (July), 136:1–136:12.

WANG, W., CHAO, H., TONG, J., YANG, Z., TONG, X., LI, H., LIU, X., AND LIU, L. 2015. Saliency-preserving slicing optimization for effective 3d printing. *Computer Graphics Forum 34,* 6, 148–160.

WANG, W. M., ZANNI, C., AND KOBBELT, L. 2016. Improved surface quality in 3d printing by optimizing the printing direction. *Computer Graphics Forum 35,* 2, 59–70.

XU, F., WONG, Y., LOH, H., FUH, J., AND MIYAZAWA, T. 1997. Optimal orientation with variable slicing in stereolithography. *Rapid Prototyping Journal 3,* 3, 76–88.

ZHANG, X., LE, X., PANOTOPOULOU, A., WHITING, E., AND WANG, C. C. L. 2015. Perceptual models of preference in 3d printing direction. *ACM Trans. Graph. 34,* 6 (Oct.), 215:1–215:12.

ZHAO, Z. AND LUC, Z. 2000. Adaptive direct slicing of the solid model for rapid prototyping. *International Journal of Production Research 38,* 1, 69–83.