A slice-traversal algorithm for very large mapped volumetric models

Jeremy Youngquist, Meera Sitharam, Jörg Peters*

Computer and Information Sciences and Engineering, University of Florida

Abstract

When the full-scale storing and retrieving of volumetric models is cost prohibitive, intersection queries require intelligent access to pieces generated on demand. To conform to a given curved outer shape without clipping, such models are often the result of a non-linear free-form deformation applied to a geometrically simpler, canonical model. The additional challenge is then to relate the intersection query back to the pieces of the pre-image of the conforming curved model.

Motivated by 3D print slice generation of massive mapped material micro-structure, this paper presents an algorithm to traverse a planar slice of a volumetric mapped model that is too large to be treated as an explicit model. The algorithm safely reduces the large-scale slicing problem to many small-scale problems that can be solved by existing slicing techniques. With the pre-image partitioned into boxes, the algorithm activates boxes to generate micro-structure only where the non-linear image of the box can intersect the given plane. Active boxes are intelligently traversed to guarantee both full coverage and minimize a front of active boxes.

Keywords: very large volumetric models, free-form deformation, slice-traversal, on-demand mesh processing.

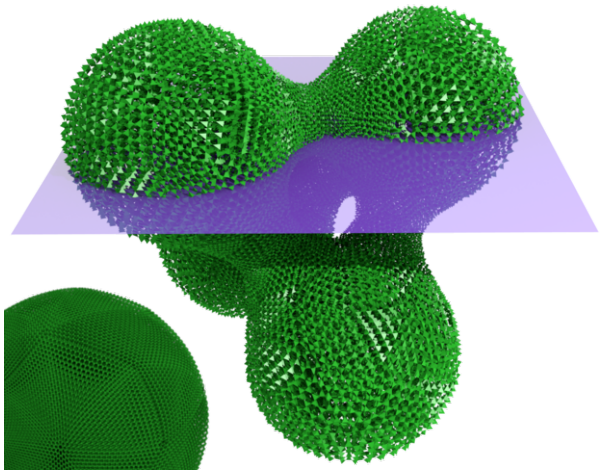

Figure 1: Mapped micro-structure and slice plane. A non-linear free-form deformation, assembled from 150 maps deforming simplicial domains, bends the micro-structure to follow a curved outer bubble surface. The purple slicing plane selects pieces to be activated/generated. For ease of exposition, the example is low-resolution. The *lower left* illustrates a more typical finer resolution.

1. Motivation

State-of-the-art 3D printers are capable of printing structure at the micrometer [1] and even nanometer [2] scale. Additive manufacturing therefore, in principle, allows designs that spatially vary and optimize the material properties of objects via their micro-structure [3, 4, 5, 6]. However, representing a one meter cube with micrometer structures challenges existing storage capacities and access: the natural response, to generate the micro-structure on demand has to cope with fine-scale micro-structures over regions up to one million times larger than the feature size [7, 8]. Therefore local micro-structure generation must be activated with a tight focus on the query. For 3D printing of very large models with micro-structure, only the immediate neighborhood of the material deposition plane, called slicing plane, should be generated in chunks that are each small enough to be treated by existing slicing algorithms.

The enumeration of all and only those chunks that can straddle the slicing plane is the *algorithmic contribution* of this paper. Moreover the traversal by enumeration is coherent and jump-minimizing (to reduce printer head movement) and applies when the micro-structure is non-linearly mapped to conform to an outer boundary shape. We also minimize the storage cost by avoiding storing all chunks that straddle the slicing plane, storing only a minimal subset at any given time.

One can make micro-structure conform to a complex curved outer boundary by subjecting the material to a collection of free-form deformation maps $\mathbf{g}^\gamma : \mathbb{R}^3 \to \mathbb{R}^3$, $\gamma = 1, \ldots, N$ [9, 10].

This approach allows separating reasoning about the micro-structure and its embedding into physical space. By contrast, clipping the un-mapped micro-structure to match the shape can easily compromise structural integrity along the boundary surfaces as supporting struts are chopped off and left dangling. As this paper's running example of a free-form deformed micro-structure, Figure 1 shows a curved union of bubbles filled with a mapped Kagome micro-structure, adapted from [11, 12]. Figure 2 shows one printed piece, a box filled with the characteristic interleaved tri-hex planes of the 3D Kagome lattice. (The insets show the basic structural piece, each node connected to six neighbors to guarantee minimally rigidity, and an intentionally constructed hole retaining the connectivity.) More generally, the micro-structure can be any that fills on-demand and deterministically so that chunks fit together without unwanted gaps or overlap.

While effective for modelling, on-demand generation of mapped micro-structure presents a challenge. One would like to only generate those pieces whose images straddle a given slic-

*Corresponding author
Email address: {jyoungquist,sitharam,jorggato}@ufl.edu (Jeremy Youngquist, Meera Sitharam, Jörg Peters*)

ing plane since generating and testing the entire micro-structure is too expensive. Pulling back the interrogation from the mapped image to the original model turns the slicing plane into a complex shape even when we restrict $\mathbf{g}^\gamma$ to be close to the identity: the pre-image of the plane is the result of applying the inverse of the free-form deformation, e.g. the inverse of a polynomial map.

This paper presents an algorithm that organizes slicing of very large mapped micro-structures at the meta level, i.e. identifies the pieces of the un-mapped micro-structure that need to be activated for localized, on-demand micro-structure generation. This reduces the slicing challenge to chunks of the mapped micro-structure that are small enough to be treated by existing slicing algorithms. For these very large micro-structures the slice-traversal algorithm

- efficiently predicts what pieces of the original un-deformed micro-structure need to be generated and mapped to guarantee complete coverage of the slicing plane;
- organizes the traversal of the pieces to hold in memory only pointers to a subset of these pieces;
- is agnostic to the choice of micro-structure but leverages the advantages of local, on-demand micro-structure generation schemes; and
- reduces the large-scale slicing problem to smaller-scale slicing, permitting the use of existing slicing algorithms for smaller pieces.

This paper focuses purely on the challenge of traversal of boxes that reduce the printing problem to manageable chunks. The paper does not aim to add insight into other topics of additive manufacturing such as support-structures[13], under- or overfill [14] or mechanical or multi-material properties of specific micro-structures. Local chunks are treated as black boxes to be sliced by existing algorithms.

Since the algorithm delegates the slicing at the level of the micro-structure as a black box, the actual micro-structure can be any that fills on-demand and deterministically so that chunks fit together without unwanted gaps or overlap. The algorithm requires that the free-form deformation maps $\{\mathbf{g}^\gamma : \gamma = 1, \ldots, N\}$ have a bounded Jacobian and are globally injective. This avoids extreme distortions and multiple cover that micro-structures should avoid. Tight bounds on the maps are based on second derivatives of the $\mathbf{g}^\gamma$. Figure 1 shows a deformation map of piecewise total degree 3. The partition of the macro-shape into curved simplices is taken from [12] but can be generated by any simplicial decomposition algorithm. The algorithm partitions the pre-image domain into box-shaped chunks.

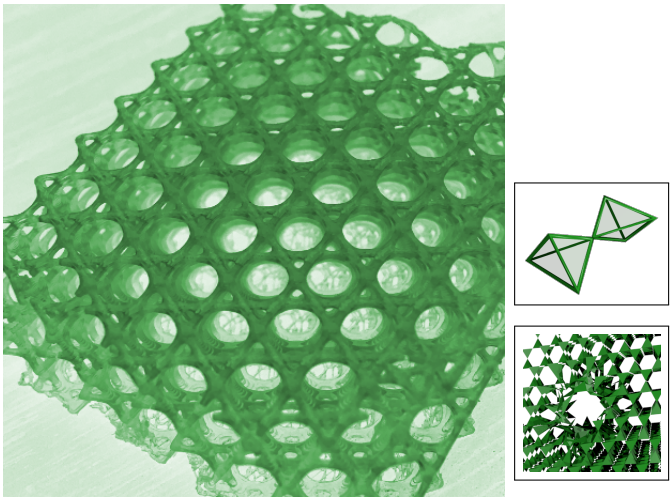

Figure 2: Example of a box, here filled with a printed 3D spline-modelled Kagome lattice and variants such as *upper right inset* a hole irregularity. The *lower right inset* illustrates how 6 struts join at every internal point and form the micro-structure of Figure 1.

Overview. Section 2 reviews prior work on slicing very large mapped volumetric models. Section 3 defines the slicing problem and the partition into boxes. Section 4 establishes a close numerical bound for the image of a box under a polynomial map. Section 5 presents the fat front algorithm for efficient, low memory and jump-avoiding traversal of all intersections between a plane and (the mapped boxes partitioning) a polytope. Section 6 analyzes the complexity of the fat front algorithm. Section 7 compares run time, storage, printer head movement, and time per box of the fat front algorithm to three alternative graph traversal algorithms.

2. Slicing Very Large Volumetric and Mapped Models

Models of volumetric micro-structure are very large if they can not be stored explicitly but have to be generated in chunks, on demand. Models are mapped if they are not originally defined in physical space but are subjected to a free-form deformation map from $\mathbb{R}^3$ to $\mathbb{R}^3$. There are a number of options for representing very large micro-structure models and mapped models.

Besides data-driven solid texture synthesis, see e.g. [15, 16], procedural micro-structures provide a rich source of on-demand volumetric data, see eg. [17, 18] for a programmable pipeline that synthesizes multi-material 3D printed objects. A Function Representation (FRep) defines a geometric object implicitly by a real continuous function p. The scalar inequality $x \in \mathbb{R}^3 : p(x) \geq 0$ identifies regions that belong to the micro-structure [19, 20, 21, 22, 23, 24]. One option of a scalar field is the voxelized signed-distance map in physical space [25]. A free-form deformation or mapping can additionally be applied [26]. Contours of curved FReps can be extracted, e.g. using marching cubes [24].

A common modeling approach is to generate a base shape and to decorate it with repeating micro-structure in a base cell, possibly with variation [6, 27, 28]. By constrast, Voronoi foam-like structures [29, 30] are aperiodic micro-structures generated directly in physical space.

While micro-structures generated explicitly in physical space can be easily tiled, inducing curved shape on the micro-structure by curved clipping can destabilize the now-exposed partial structure. Using the signed distance to the bounding surface [17, 18] an additional conforming wall at the boundary can be generated. An alternative is to subject a given micro-structure to a free-form deformation. A micro-structure conforms when shape and size of the unit cells can adapt to the macro shape of its design boundary. Shape conforming lattices avoid having a partial cell located on boundary. For micro-structure to be used with state-of-the-art modeling and analysis tools, they should be defined in the parametric space of the elements. For example, isogeometric analysis [31, 32, 10] uses parametric maps, splines or Bézier functions to define geometry. Defining micro-structure in the domain of a free-form deformation is also necessary to leverage the standard tools of computer-aided geometric design [11, 27, 26].

Many classes of sufficiently fine-scale micro-structures behave as if they were a continuum, with the continuum behavior increasing in accuracy as the micro-structure is made finer [33, 21, 22]. This allows for specification of functionally graded material properties of procedural models while still treating

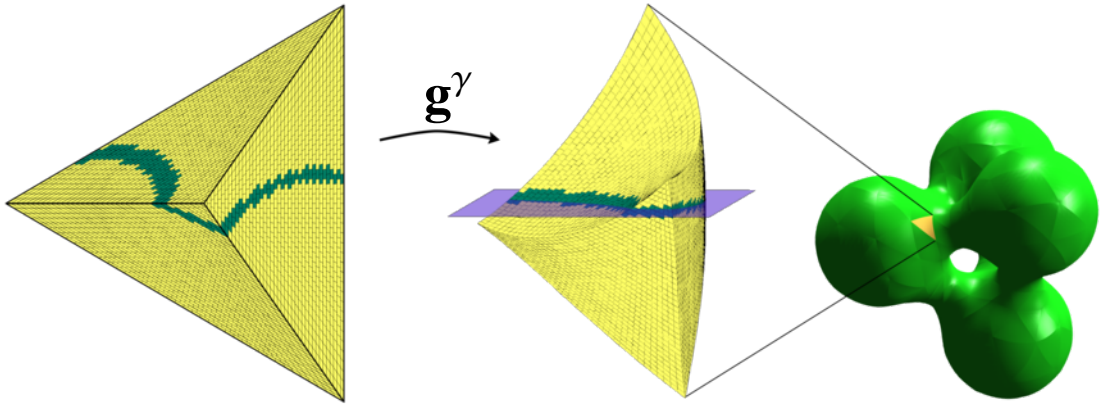

Figure 3: Non-linearity of the pre-image of the slicing plane. *Left*: Simplex Δ partitioned into boxes $\square^\beta$. The green boxes map to hexahedra that straddle the slicing plane. *Middle*: $\mathbf{g}^\gamma(\Delta)$ covered by overlapping hexahedra that each enclose one $\mathbf{g}^\gamma(\square^\beta)$. The slicing plane is purple. *Right*: The full set of maps $\{\mathbf{g}^\gamma : \gamma = 1, \ldots, N\}$ with the exterior face of the specific $\mathbf{g}^\gamma(\Delta)$ marked yellow.

them with continuum analysis, and has been successfully integrated into analysis tools to optimize micro-structure to match a spatially-varying compliance matrix [3, 4, 5, 6].

Our algorithm applies to any of these micro-structures placed in the domain of a free-form deformation map $\mathbf{g}$ as long as a construction deterministically fills each box on demand. Periodic micro-structures qualify. Aperiodic and randomized micro-structures qualify if, after fixing a random seed, repeated calls on the same box return the same micro-structure.

Surface slicing algorithms take as input a triangulated surface and return a set of oriented curves representing the intersection of that surface with a set of slice planes. These curves are used to generate infill and machine instructions for the printer. Our focus is on selecting relevant subsets (boxes) of micro-structure. We treat the final slicing operation as a black box.

For accelerating the slicing of volumetric structures, ray casting and bitmaps can be leveraged, e.g. to optimize slicing for slender truss-based structures where manifold .stl representations are highly inefficient [34]. Hierarchical regular lattice representations consisting of straight beams and circular balls transformed by translation or dilation to accelerate ball localization queries can be taken advantage of [35]. Both approaches depend on a specific representation.

Leveraging massively parallel architecture, GPU-based methods for model slicing can provide up to a 30x speedup over similar CPU-based algorithms [36, 37, 38] and out-of-core slicing algorithms reduce the memory burden for larger meshes by keeping the mesh on the hard drive rather than the memory of the computer[39, 40]. However, these out-of-core methods have quadratic complexity in the number of triangles and so are cost-prohibitive for very large volumetric models. An optimal slicing algorithm [41] can be designed to have complexity linear in the number of triangles, the number of slice planes, and the number of triangle-plane intersections – provided the slices have uniform thickness. However, this algorithm takes as input the entire mesh, and is therefore impractical for very large mapped volumetric models.

The algorithm to be presented treats very large, mapped micro-structures by traversing sufficiently small chunks, generated on-demand.

3. The slicing problem for mapped micro-structure

One way to make the micro-structure conform to the outer hull of an object, is to clip it. However clipping can destabilize the structure at the boundary, leaving dangling pieces of edges. Instead, in the following, the micro-structure is made to conform by applying a family of local free-form deformations $\mathbf{g} = \{\mathbf{g}^\gamma : \gamma = 1, \ldots, N\}$, see Figure 3, that preserve its combinatorial and topological structure.

When slicing the deformed curved shape, the traversal algorithm must *activate all chunks of the domain that contain original structure whose on-demand generated image straddles the slicing plane*. The chunks must be sufficiently small so that existing slicing algorithms can be applied as a black box. The challenge in this setting is to pull the interrogation back from the plane intersection of the mapped image to the original model – without truly inverting the trivariate (piecewise polynomial) map (see Figure 3). Given the number of samples needed, a numerical inversion to trigger generation of the domain boxes is neither practical nor, due to gaps and overlap, safe; and also encounters convergence challenges [42].

The 3D printing of objects by layers requires slicing the images of one or more canonical domains Δ mapped by a collection of free-form deformations $\mathbf{g} = \{\mathbf{g}^\gamma : \gamma = 1, \ldots, N\}$ by a sequence of planes p^λ, $\lambda = 1, 2, \ldots, L$. Let $n \in \mathbb{N}$ be a tessellation number that guarantees that splitting Δ into boxes $\square^\beta$ with indices β of side length at most $1/n$ can be handled by a black box slicing algorithm from the literature suitable for the micro-structure. Then the slicing problem can be stated as follows.

Definition 1 (Slicing Problem). *Given a set G of maps $\mathbf{g}^\gamma : \Delta \to \mathbb{R}^3$, $\gamma = 1, \ldots, N$, a set of planes $p^\lambda \in P$, $\lambda = 1, 2, \ldots, L$, and a constant n, enumerate for each λ and for each γ the set of indices β of all boxes $\square^\beta \subset \Delta$ of side length at most $1/n$ such that $\mathbf{g}^\gamma(\square^\beta) \cap p^\lambda \neq \emptyset$.*

The key to efficiency is to activate only those boxes whose image can overlap a current slice plane p^λ. To this end, we specify the maps $\mathbf{g}^\gamma$ and the boxes $\square^\beta$ in more detail.

3.1. 3D free-form deformations $\mathbf{g}^\gamma$

The algorithm applies to a family of piecewise smooth free-form deformations $\mathbf{g}^\gamma$; for example piecewise polynomial mappings such as trivariate tensor-product B-splines. Here we consider polynomial pieces $\mathbf{g}^\gamma$ of the 3D deformation [43] in *total*

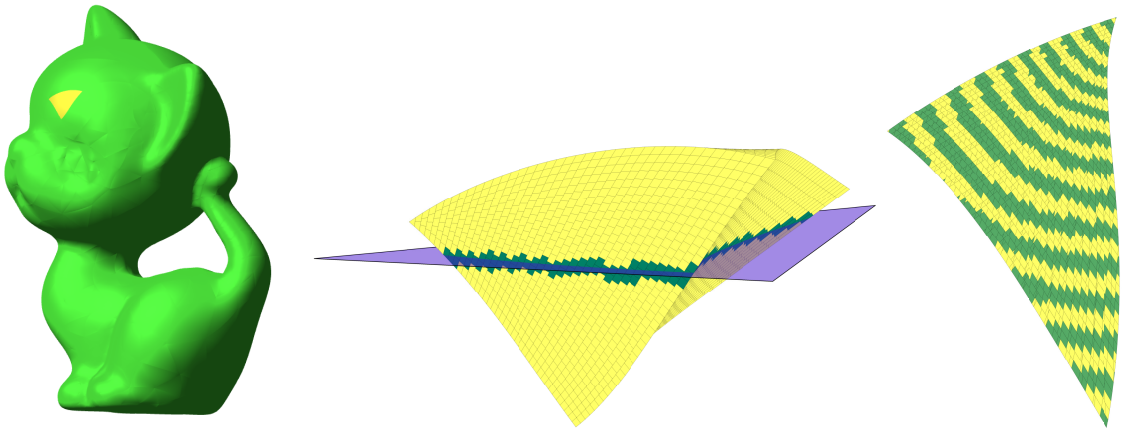

Figure 4: Slice through a mesh of a kitten. *Left*: The object in 3-space consisting of curved images of simplices. *Middle*: Focus on a single mapped simplex intersection with one slicing plane. One face of this mapped simplex is shown in yellow on the kitten. *Right*: The partition of the intersection into fat fronts, alternately color-coded.

degree k Bernstein-Bézier form (BB-form), see e.g. [44]:

$$\mathbf{g}^\gamma : \Delta \subset \mathbb{R}^3 \to \mathbb{R}^3 \qquad \sum \alpha_i = k,\ \alpha_i \geq 0, \tag{1}$$

$$\mathbf{g}^\gamma(\mathbf{u}) := \sum_\alpha \mathbf{g}^\gamma_\alpha B_\alpha(\mathbf{u}), \quad B_\alpha(\mathbf{u}) := \frac{k!}{\alpha_0!\alpha_1!\alpha_2!\alpha_3!} \prod_{i=0}^{3} \mathbf{u}_i^{\alpha_i}.$$

The domain Δ is a simplex and the BB-coefficients $\mathbf{g}^\gamma_\alpha$ form a lattice, called the BB-net, that connects coefficients whenever one index coordinate is increased (and hence another decreased) by 1. The BB-net outlines the the image and $\mathbf{g}^\gamma(\Delta)$ must lie in the convex hull of the BB-coefficients.

Figure 3 illustrates the challenge of finding the pre-image of the slicing plane. From right to left, on the right, the map consists of images of simplices under maps $\mathbf{g}^\gamma$ in total-degree BB-form. The image of a one map is highlighted in yellow and shown enlarged in the middle of Figure 3 covered by hexahedra. Each hexahedron encloses the image under $\mathbf{g}^\gamma$ of one box of the domain partition, Figure 3, left. (Section 3.3 explains the partition of the domain in detail.) The pre-image of the slice plane (purple) is highly non-linear; the green boxes in the domain (Figure 3, left) are the candidates to be sliced.

Injectivity of each $\mathbf{g}^\gamma$ and indeed of the whole collection of maps $\mathbf{g}$ is mandatory when generating micro-structure since we cannot deposit material twice in the same location. In any case, large deformations are impractical when they stretch or squeeze the micro-structure beyond its free space. We therefore make the following **assumptions**.

- a1 The Jacobian determinant $\det \nabla \mathbf{g}^\gamma$ of $\mathbf{g}^\gamma$ is nonzero and bounded: $0 < m_0 \leq \det \nabla \mathbf{g}^\gamma \leq m_1$.
- a2 The number of boxes n per edge of the domain Δ is sufficiently large so that the pre-image of p^λ does not intersect any box more than once.
- a3 Pre-images of curves have dimension 1 (are not fractal).

Standard slicing places the print bed parallel to the xy plane at $z = z_0$

3.2. Repeated slicing

Since we assume that we cannot store the micro-structure of the $O(n^2)$ boxes that intersect a slice plane, we must regenerate box micro-structure for the adjacent slice.

Denote by $\underline{z}^\gamma$ the minimal z coordinate of all BB-coefficients of $\mathbf{g}^\gamma$ and $\overline{z}^\gamma$ the maximal z coordinate of all BB-coefficients. By the convex hull property of the BB-form, p^λ can intersect $\mathbf{g}^\gamma$ only if $\underline{z}^\gamma \leq z(p^\lambda) \leq \overline{z}^\gamma$. The testing can be sped up by sorting the $\mathbf{g}^\gamma$ by least $\underline{z}^\gamma$ for a plane sweep [41].

This reduces the Slicing Problem to one plane and one map. Figure 5 shows how multiple map-slices join to form a global slice. Figures 3 and 4 focus on two aspects, non-linearity of the pre-image and efficient traversal, of a single map $\mathbf{g}^\gamma$ of the family of maps.)

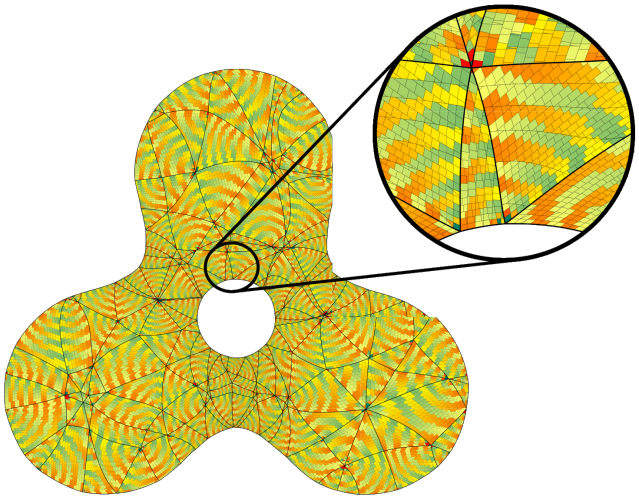

Figure 5: 2D slice of Figure 1 to show how multiple sliced pieces join. Note that the unequal size of the black bounded regions does not indicate distortion but slicing of the deformed simplices at different depths. Each facet records one of 594,641 intersections between a hexahedron and the slice plane. Each mapped Δ is sliced independently of the others. The color pattern stems from the traversal via the fat front (FF) algorithm introduced in Section 5.

3.3. Partitioning the domain Δ into boxes $\square^\beta$

Since we assume that $O(n^2)$ storage is excessive and only $O(n)$ storage is permissible, we cannot explicitly subdivide at the outset to generate a $1/n$ partition and then apply the convex hull test to obtain maps and bounds on each piece. Nor can we sort the labels of $O(n^2)$ pieces whose convex hull intersects into a range list. Trading work for storage, a generate-and-test regimen using on-the-fly subdivision is also not possible since that has work complexity $O(n^2)$.

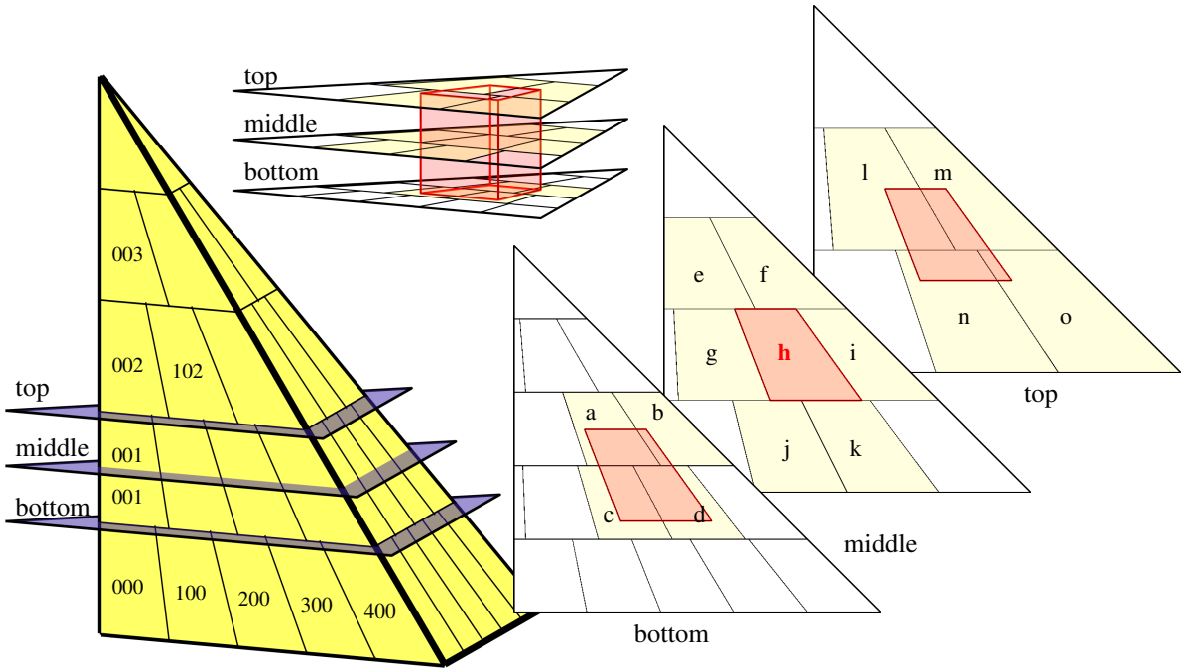

Figure 6: The brick layer's partition of a simplex Δ with boxes for resolution $n = 5$. The box highlighted in red and strictly inside Δ has a full neighborhood. The slices show that an internal box, h, has at most 4+6+4 neighbors in $a, b, \ldots, o$. The relative indices of the neighborhood are enumerated on the *right* and are the same regardless of how fine a partition is chosen.

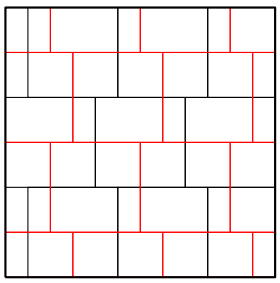

Figure 7: Top view of two layers (black, red) of the brick layer's partition of the cube domain $[0,1]^3$. Every second row of the black layer is shifted by 1/4 of a box width. In the red layer, both x and y are shifted by 1/2 in addition to 1/4 shifting in alternating rows. Each vertex has at most 4 incident boxes: three in one layer plus one in the adjacent layer.

Instead, we partition the domain Δ into boxes $\square^\beta$, where β is the triple index illustrated in Figure 6. The boxes fill the domain Δ without overlap so that, unlike when superimposing a uniform grid on Δ, we do not encounter partially filled boxes or clipped boxes.

The chosen brick layer's partition is motivated by the need to avoid processing the same micro-structure neighborhood repeatedly at very different times. That is, we want to minimize the number of $\square^\beta$ joining at a common point. The most efficient way to partition a polynomial in BB-form via de Casteljau's algorithm into constituent pieces, by always halving a longest edge [45]. This results in up to 48 sub-simplices meeting at a vertex. The more labor-intensive partition of [46], splitting the difficult octahedral domains into tetrahdra, yields locations where 32 pieces meet. By comparison, a regular grid-like partition into cubes has 8 $\square^\beta$ meet at a vertex, as do various oct-tree refinements.

Another goal is to avoid extended fault lines through the material. Unlike for other tasks, such as creating spline surfaces, here T-junctions not only pose no problem but they are beneficial to prevent too many shear planes meeting – as brick layers and Lego enthusiasts have long discovered. We therefore partition so that at most four boxes, the minimal number, meet at a vertex.

The brick layer's partition of a cube domain (see Figure 7) shifts alternating layers in z of a regular $n \times n \times n$ grid partition. Every second layer is shifted in both x and y by one half of ℓ, the side-length of a box. Within each layer, alternating rows are additionally shifted in x by $\ell/4$. This reduces the number of boxes incident on any vertex to at most 4.

The brick layer's partition of a simplex Δ with n layers is alike except that layer k is partitioned into $n-k$ rows and row j is split into $n-k-j$ boxes. Every even row j is shifted by $\ell/2\pi$ to break symmetries. Then only 4 boxes are incident on any vertex. The shift yields one partial box on the left and another on the right. At the apex, the paving degenerates into a tetrahedron but this is not a problem since the goal is only to identify neighbor boxes in Δ. Each box is identified with the indices $(i, j, k) =$ (box, row, layer) as illustrated in Figure 6. Each box has at most 14 neighbors (4 above, 6 in the same layer and 4 below) explicitly listed in Figure 6, *right*.

The restriction of $\mathbf{g}^\gamma$ to a skew box $\square^\beta$ with horizontal top and bottom planes ($z = const$ in the domain) is exported to the black box slicing algorithm as $\mathbf{g}^\gamma \circ \square^\beta$.

4. Bounding the image of a box $\square^\beta$

This section shows how to tightly bound the image of a box $\square^\beta$ to quickly test whether a box should be tagged for micro-structure generation and slicing. The approach first bounds the whole map $\mathbf{g}^\gamma$ and then proposes a fast, tight estimate that requires only a few evaluations.

4.1. A tight enclosure of $\mathbf{g}^\gamma(\Delta)$

The key to efficiently activating polyhedral sub-regions $\square^\beta$ of Δ is to determine whether $\mathbf{g}^\gamma(\square^\beta)$ intersects the slice plane. We reduce this query to a query on the simpler BB-net – without subdivision! We will only consider tight enclosures of the curved images $\mathbf{g}^\gamma(\square^\beta)$. Due to their conservative, outer approximation, neighboring enclosures will typically overlap – but this is of little

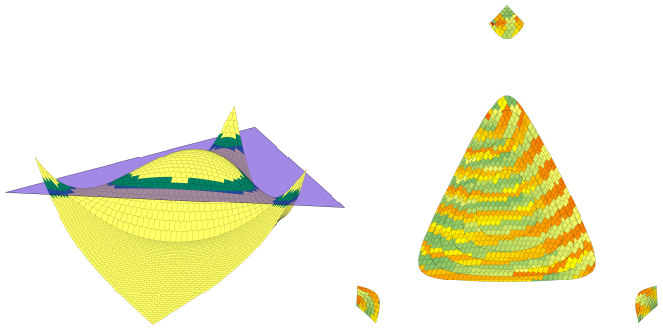

Figure 8: The traversal of a slice of a map $\mathbf{g}^\gamma$ of total degree 4.

consequence since the enclosures are never explicitly computed but only serve to identify their domain boxes.

To efficiently compute the tight enclosures, consider the four corner vertices $\mathbf{v}_i$ of the domain Δ and the (trivial) linear interpolant $\ell : \Delta \to \mathbb{R}^3$ of the simplex T spanned by the images of corner vertices $\mathbf{g}^\gamma(\mathbf{v}_i) = \mathbf{g}^\gamma_\alpha$, $\alpha := 3\mathbf{e}_i$ in physical space. The necessary enlargement of T depends on how much $\mathbf{g}^\gamma(\Delta)$ differs from ℓ. Generalizing an estimate in [47] to the trivariate total degree case we have

$$\sup_{\mathbf{u}\in\Delta} \|\mathbf{g}^\gamma - \ell\| \le \frac{1}{8} \sum_{i,j\in\{1,2,3\}} l_i l_j \sup_{\mathbf{u}\in\Delta} \|\partial_{\mathbf{u}_i\mathbf{u}_j}\mathbf{g}^\gamma\|, \tag{2}$$

where l_i is initially the unit length of the domain with respect to variable $\mathbf{u}_i$. Here the mixed partials $i \neq j$ are counted twice for a total of 9 terms of which only 6 are distinct.

In our example $\mathbf{g}^\gamma$ is of total degree 3. Therefore $\partial_{\mathbf{u}_i\mathbf{u}_j}\mathbf{g}^\gamma$ is linear, and each of the six distinct mixed partial derivatives attains its maximum at one of its four vertices. As is well-known for polynomials in BB-form [48], the values of $\partial_{\mathbf{u}_i\mathbf{u}_j}\mathbf{g}^\gamma$ at each of its four vertices are a multiple (here $3 \cdot 2$) times the second differences at the vertex. For $i \neq k \neq j$ and $k \in \{1, 2, 3, 4\}$, we compute the second differences of the BB-coefficients

$$\mathbf{d}_{ijk} := \mathbf{g}^\gamma_{3\mathbf{e}_k} - \mathbf{g}^\gamma_{2\mathbf{e}_k+\mathbf{e}_i} - \mathbf{g}^\gamma_{2\mathbf{e}_k+\mathbf{e}_j} + \mathbf{g}^\gamma_{\mathbf{e}_k+\mathbf{e}_i+\mathbf{e}_j}. \tag{3}$$

Denote the z-coordinate of $\mathbf{d}_{ijk}$ by $z(\mathbf{d}_{ijk})$. Then $\mathbf{g}^\gamma(\Delta)$ is enclosed by offsetting T in z by

$$\mu^z := \frac{6}{8} \sum_{i,j\in\{1,2,3\}} l_i l_j \max_k |z(\mathbf{d}_{ijk})|, \tag{4}$$

(and alike in x and y, but we will only need the z estimate). Note that this computation is done only once for each $\mathbf{g}^\gamma$ and not repeated for each $\square^\beta$.

4.2. A tight hexahedron enclosure $\diamondsuit^\beta$ of $\mathbf{g}^\gamma(\square^\beta)$

If we split each edge of the domain Δ into $n = 2^\nu$ pieces then $l_i l_j$ in (4) becomes $4^{-\nu}$ times the initial unit edge length. Let $\mathbf{v}^\beta_i$ be the vertices of a box $\square^\beta$ and let $\diamondsuit^\beta$ be the hexahedron in the physical space spanned by the 8 points $\mathbf{g}^\gamma(\mathbf{v}^\beta_i)$. Note that in general $\diamondsuit^\beta \neq \mathbf{g}^\gamma(\square^\beta)$ since $\diamondsuit^\beta$ is a simpler object than the curved $\mathbf{g}^\gamma(\square^\beta)$ and that $\diamondsuit^\beta$ is easy to compute. Since any hexahedron $\diamondsuit^\beta$ can be split into tetrahedra, $\diamondsuit^\beta$ enlarged by $\mu/4^\nu$ in x, y, z tightly encloses $\mathbf{g}^\gamma(\square^\beta)$. To guarantee that all boxes are tagged whose non-linear images can intersect the slicing plane, it suffices to

- compute the images $\mathbf{g}^\gamma(\mathbf{v}^\beta_i)$ of the box corners,
- test for intersection the *edges* of $\diamondsuit^\beta$ with the planar slice with a tolerance of $\mu/4^\nu$.

A positive outcome in the second step then identifies any $\square^\beta$ whose image can intersect the slicing plane. Note that the testing against the 12 edges of $\diamondsuit^\beta$ can stop after the first positive test.

While tighter bounds of the images of the $\square^\beta$ can be obtained, e.g. by explicit subdivision of $\mathbf{g}^\gamma$ or by sleves [49], we expect the number of boxes $\square^\beta$ to be very large so that $4^{-\nu}$ is the dominant factor and so the cheapest test against 12 edges of $\diamondsuit^\beta$, based on a single initial estimate (4), is most efficient.

Let z_i be the z-coordinates of the corners $\mathbf{g}^\gamma(\mathbf{v}^\beta_i)$ of $\diamondsuit^\beta$, $i = 1, \ldots, 8$. After subtracting a constant, the slice plane is $z \equiv 0$. The intersection test can therefore be further simplified to checking whether one point is close to zero, or the interval between two z_i straddles zero.

Intersection Test, degree 3:
Tag β as an index of a box to be activated if

- for some i, $|z_i| < \mu/4^\nu$ or
- for some $i \neq j$, $z_i z_j < 0$.

Our typical examples show total degree 3 Bézier functions where the simple bounds Equation 4 apply directly. Using Equation 2, we can bound images of higher degree maps: Figure 8 illustrates multiple components for one piece $\mathbf{g}^\gamma$ of total degree 4.

5. Slice-traversal algorithm

Even when the only one slice plane and one map $\mathbf{g}^\gamma$ interact, the traversal of all boxes cannot be stored since it consists of $O(n^2)$ indices. We would like to limit storage of box IDs to $O(n)$ and minimize jumps from one hexahedron to the next since this would correspond to unproductive printer head movement.

Definition 2 (Slice Traversal Problem). *Given a map $\mathbf{g}^\gamma : \Delta \to \mathbb{R}^3$, a plane p, and a constant n, discover the set of all indices β of boxes $\square^\beta \subset \Delta$ of side length at most $1/n$ such that $\mathbf{g}^\gamma(\square^\beta) \cap p \neq \emptyset$ and such that no more than $O(n)$ indices need to be stored at any time.*

Definition 3 (jump length). *The* jump length *between two nodes is the minimal distance between their hexahedron slices, i.e. zero where the two $\mathbf{g}^\gamma(\square^\beta) \cap p$ touch. The* total jump length *is the sum (1-norm) of the jump lengths.*

As before, we illustrate the algorithm with a polynomial map $\mathbf{g}^\gamma$ of total degree 3 in Bernstein-Bézier form. Δ is a simplex, and p a single plane with normal $(0, 0, 1)$.

5.1. Map-plane intersection

Let each box be represented by a node with an edge connecting it to its neighbor boxes. The nodes are the centers of its hexahedron's intersection with the plane, see • in Figure 9. Then the traversal can be formulated as a graph traversal problem on an unknown graph.

One approach is to lexicographically search through all box IDs to discover those whose images intersect the plane, which we refer to as the *brute force* algorithm. This brute force algorithm requires no storage of IDs but gives poor bounds on the total jump length and has a complexity of testing against all $O(n^3)$ hexahedra. Depth-first search (DFS) on the graph reduces the tests to

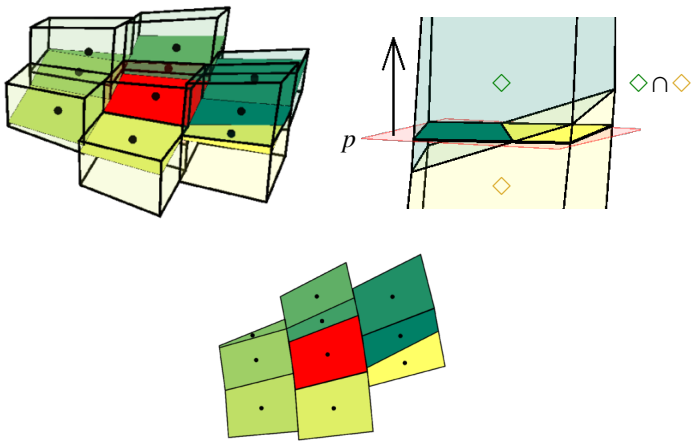

Figure 9: Planar cut through hexahedra $\diamond^\beta$. *left*: a 3D view of the active hexahedra, *bottom*: a top view of their intersections with the slice plane. *right*: Oblique cuts yield slanted line intersections that give the appearance of split quadrilaterals, here illustrated for the yellow-green pair. The current $\diamond^\beta$ is red and its neighbors are colored in counter-clockwise order from green to yellow. Each *node* • represents a box.

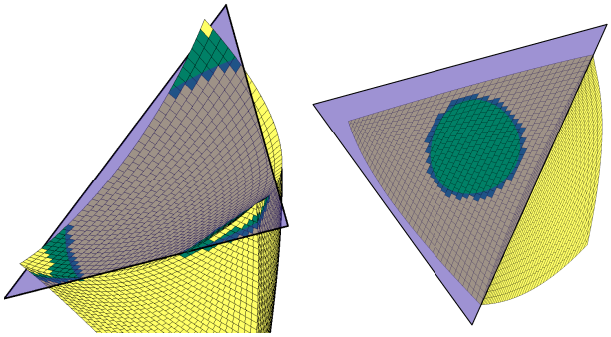

Figure 10: *Left*: A single slice with multiple components due to intersecting edges. *Right*: A closed loop intersection on a face (does not intersect any edges).

only the $O(n^2)$ hexahedra which intersect the plane and reduces the number of jumps since it prioritizes visiting a neighbor of the current box– but DFS jumps can require extensive backtracking resulting in long jumps and requires storing visited box IDs to ensure they are not visited twice.

We therefore use a variant of Breadth-First search (BFS) that visits nodes by Euclidean distance in the slice plane to an initially discovered node. The ordering by distance prevents backtracking but may cause incessant jumping as the front advances like a prairie fire to form (the arcs of) a circle and always picks the next with least distance. Our variant of the BFS therefore alternates between collecting a set, called a 'fat front', of boxes and then traversing the set while minimizing the jump length.

We note that to eliminate jumps altogether, we would have to construct a Hamiltonian path, i.e. first collect the $O(n^2)$ nodes and then solve an NP complete graph problem.

By assumption a1 and the Pre-Image Theorem, pre-images of slices through $\mathbf{g}^\gamma(\Delta)$ are surfaces and are disconnected only by slicing through the boundary. Due to the curvature of $\mathbf{g}^\gamma$, near the boundary, the intersection $\mathbf{g}^\gamma(\Delta) \cap p$ can have multiple **components** (see Figure 10). The challenge is to ensure that all these components are visited.

We find all components by testing the hexahedra incident on the edges of T and then testing the faces of T for closed loops. A

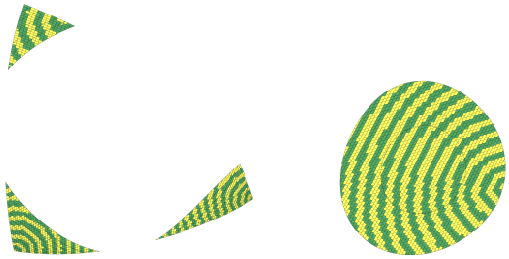

Figure 11: *Left*: Fat front (FF) traversal pattern on the slice from Figure 10, *left*. Each component is traversed independently of the others. The algorithm restarts on each component. *Right*: FF traversal pattern on the slice from Figure 10, *right*.

closed loop intersection can only occur if a face surface $\mathbf{g}^\gamma(t_1, t_2)$, $\gamma = 0, 1, 2, 3$ has a normal $\mathbf{n}^\gamma$ orthogonal to the slicing plane [50], i.e. pointing in the z direction. If the BB-coefficients of $\det(\mathbf{n}, \partial_{t_1}\mathbf{g}^\gamma, \partial_{t_2}\mathbf{g}^\gamma)$ are of one sign there is no loop. If the criterion fails to rule out an intersection, we find any intersection by traversing and testing in lexicographic order all $\diamond^\beta$ whose pre-image $\square^\beta$ is on the face of Δ.

We form a list of all hexahedra that lie on the boundary of T and use them to seed the fat front algorithm.

5.2. *The 'fat front' (FF) Euclidean distance BFS traversal of the graph in the slice plane*

A *fat front* is a set of box IDs collected during several consecutive BFS iterations. For each component, the BFS access is ordered by radius.

Definition 4 (node, radius, angle, front). *The* node *of a box ID is the center of the non-empty intersection of the hexahedron with the slicing plane. The* radius *of a node is the Euclidean distance of the node to the starting node b_0 of the component. The* angle *of a node is the angle (with foot point b_0) between the node and a fixed point in the slicing plane outside the convex hull of $\mathbf{g}^\gamma(\Delta)$. A* front *is a collection of box IDs that have both visited and un-visited neighbors.*

We refer to the new algorithm as the *fat front* algorithm (FF). FF differs from the standard BFS approach in that it accumulates box IDs whose radii lie in a small interval that is sufficiently large to be able to join the nodes with small jump length and then advance the front.

Denote the current fat front by F. A queue Q, sorted by radius, and two sets S^- and S accessed as hash tables aid in building F. We initialize Q and S^- with b_0, the starting node of the component, and both S and F to empty. We populate S and F by popping box IDs off Q until S^- is empty. For each popped box ID all un-visited neighbors are added to Q, S, and F and marked visited in S and S^-. When all neighbors of a box ID in S^- have been visited, the box ID is removed from S^-. S is considered populated when S^- is empty. Since $Q \subseteq S \cup S^-$, Q is no larger than the union of two consecutive fronts. Repeatedly computing fat fronts in this manner partitions $\mathbf{g}^\gamma(\Delta) \cap p$ as illustrated by the stripes of Figure 13 (as well as Figs. 5, 8, 11) .

Once S^- is empty, we traverse the box IDs in F starting with the box ID in F closest to the last box of the previous fat front. We iterate through F in a generally clockwise (resp. ccw) order, prioritizing neighbors of the current box to minimize jumps in the path. To this end we associate with F a hash map $M^\angle$, with expected $O(1)$ lookup cost, that maps a box ID to its angle. A

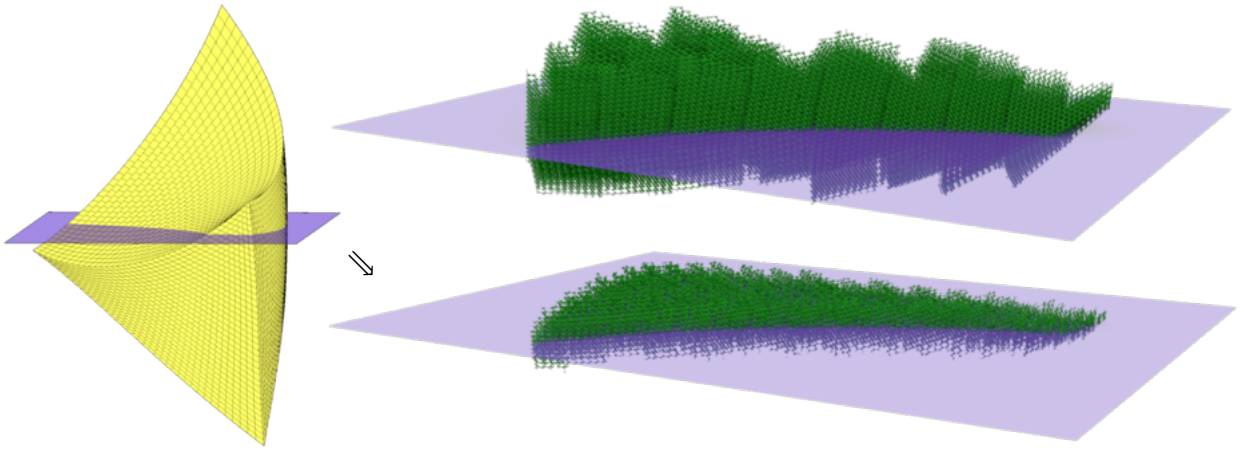

Figure 12: As the number of boxes increases, a smaller portion of the micro-structure (*bottom vs. top*) needs to be generated. (cf. green hexahedra in Figure 3.)

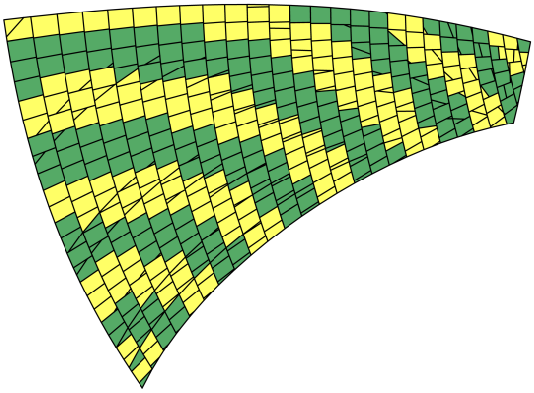

Figure 13: Partition of the slice of Figure 12 into fronts. Slanted line intersections stem from the plane slicing adjacent boxes obliquely, see the yellow-green quad in Figure 9, *right*.

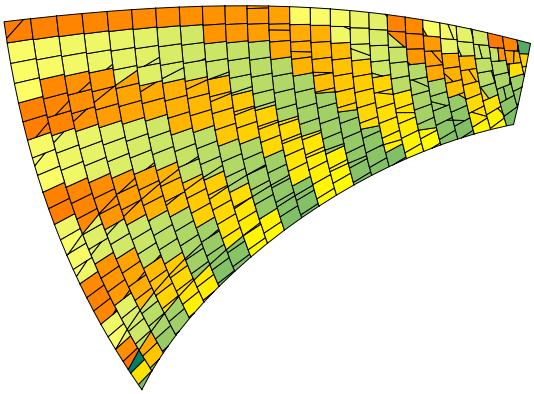

Figure 14: Traversal across fronts alternates between counter-clockwise, shown by the green to light yellow progression, and clockwise, illustrated by the orange to yellow color progression.

priority queue $Q^{\angle}$ orders the box IDs by angle for $O(1)$ selection of the box ID with minimal angle. Inserting a box ID and its corresponding angle into F inserts it into both data structures.

From the list of all box IDs in F that are neighbors of the current box, we return the box ID with minimum angle and remove it from $M^{\angle}$. We avoid the high cost of directly removing it from $Q^{\angle}$ by waiting until a current box ID has no neighbor in F, at which point we must pop box IDs off $Q^{\angle}$ until we find the next in $M^{\angle}$(to be returned and removed from $M^{\angle}$).

Visiting a box ID without a remaining neighbor in F causes a jump. The algorithm looks ahead whether the currently selected box ID b_i would create a box ID $b_j$without neighbors and if so, returns b_j instead and reinserts b_i into F. In practice, this reduces the total jump length.

Finally, we set S^- to S and clear S to build the next front. When we build the next front, we change $Q^{\angle}$ from a min priority queue to a max priority queue or vice versa to snake back and forth and so avoid large jumps at the ends, see Figure 14. An example of total jump length tracing is shown in Figure 18.

In the case where the $\mathbf{g}^{\gamma}(\Delta) \cap p$ has more than one connected component, the algorithm runs on each component (see Figure 8 and Figure 11).

6. Complexity Analysis

We analyze the complexity of the FF algorithm for slice-traversal of a single map-plane pair and treat the actual slicing of the micro-structure in the mapped box as a black box. Let n be the number of boxes along one axis of Δ, i.e. any algorithm has to output $O(n^2)$ of a total of $O(n^3)$ boxes.

Proposition 1. *The FF algorithm's worst-case run time complexity for an $O(n^2)$ output size is $O(n^2 \log n)$. The FF (index) storage complexity is at most $O(n)$.*

Proof Due to assumption a2 and the monotone increase in distance no box in the pre-image of the slice is visited more than once. Denote the z-coordinate of $\mathbf{g}^{\gamma}$ by $z(\mathbf{g}^{\gamma})$. By assumption a1, the Pre-image Theorem certifies that the pre-image $\{\mathbf{u} : z(\mathbf{g}^{\gamma})(\mathbf{u}) = z_0\}$ is bi-variate without jumps, and with holes only due to slicing the boundaries of Δ (see Figure 10 *left* for the case of multiple or isolated components). There are kn^2 hexahedra $\diamondsuit^{\beta}$ straddling the slice where the constant k depends on the tightness of the estimate μ^x of (4), i.e. the overlap of the hexahedra.

By ordering the BFS traversal to always choose the node of minimal Euclidean distance in the range, the fat front is as close as possible to (arcs of) a circular annulus. By assumption a3, its pre-image therefore has $O(n)$ elements. $O(n)$ fronts cover the slice.

The traversal cost of a single front F is bounded by $O(n \log n)$:

- insertions into and removal from the priority queue Q is $O(n \log n)$ since Q is of size at most $O(n)$ and insertion and removal per box ID cost $O(\log n)$ each;
- fill and empty the hash tables of the sets S^- and S is $O(n)$;
- construction of the hash map $M^\angle$ is $O(n)$;
- building the priority queue $Q^\angle$ is $O(n \log n)$;
- building the list of in-front neighbor box IDs of the current box ID is constant time since there are a constant number of neighbors and hash map lookup is $O(1)$ expected time;
- popping all box IDs off the priority queue $Q^\angle$ is $O(n \log n)$ since each box ID is popped exactly once in $O(\log n)$.

The total cost of iterating through $O(n)$ fronts per slice is therefore $O(n^2 \log n)$. **end of proof**

Let $O(\mathbf{m}(n))$ be the cost of generating micro-structure in a single box for streamed printing, where $\mathbf{m}$ depends on the type of micro-structure used. Then the complexity for generating micro-structure near the slice plane is $O(\mathbf{m}(n)n^2)$. As n increases, the extent of the boxes decreases in all three dimensions, and $O(n^2\mathbf{m}(n))$ can decrease as n increases due to smaller regions of micro-structure being generated, see Figure 12, and the optimal n depends on the detailed cost of $\mathbf{m}(n)$.

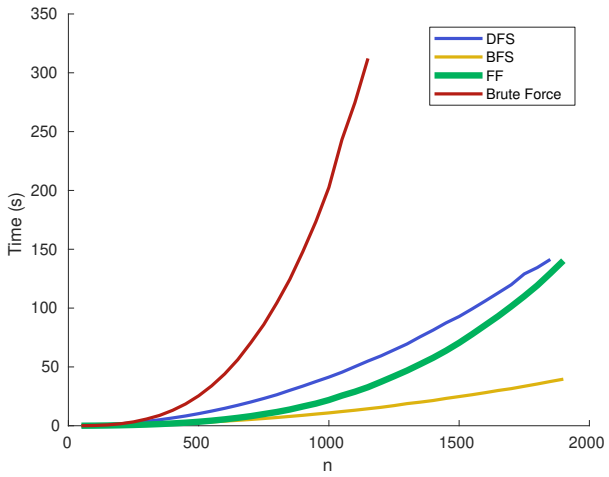

Figure 15: Run time comparison between the four algorithms discussed in Section 5.1. Note that the number of boxes enumerated is $O(n^2)$.

7. Implementation and Example

There do not exist prior implementations of slice-traversal algorithms for very large mapped volumetric models. We therefore compare the FF algorithm to the standard algorithms for graph traversal adjusted to the specific problem: brute-force, depth first search (DFS), and breadth first search (BFS).

For the DFS implementation, rather than visiting an arbitrary neighbor of the current box, we sort the un-visited neighbor nodes by angle, see Figure 9, and choose the one with the maximal angle so that the DFS tends to follow the boundary of the un-visited graph, spiraling inwards. Compared to random choice DFS, this reduces overall backtracking but does not eliminate backtracking for isolated unvisited boxes. Therefore visited boxes need to be stored leading to excessive storage complexity.

The BFS algorithm processes a box as soon as it is discovered and is placed into a list of visited boxes (a 'thin front'). Once all

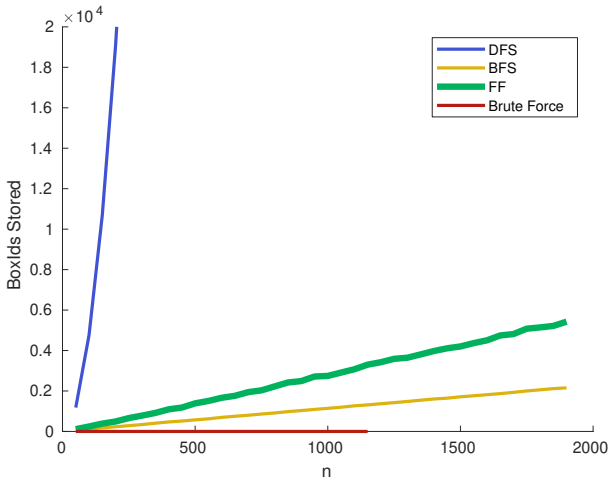

Figure 16: Storage comparison: The DFS algorithm uses storage quadratic in n whereas FF and BFS are linear.

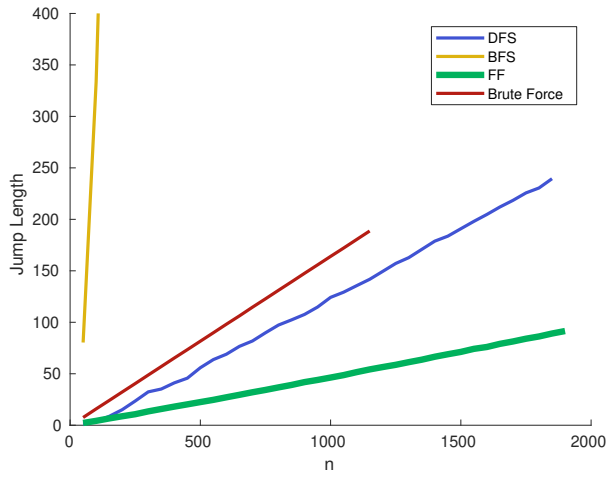

Figure 17: Jump length comparison: The jump length for the BFS algorithm increases dramatically with n due to an increase in the number of jumps.

of its neighboring boxes have been visited, the box is removed from the list. This reduces storage to $O(n)$. But BFS-consecutive boxes are often far from one another causing high jump length.

We measure the performance of the four different algorithms: FF, DFS, BFS, and brute force on a typical map-plane intersection and we apply four metrics: run time, storage cost, total distance between consecutive intersections, and time per box. Figures 3, 5, and 4 illustrate the FF algorithm on meshes with many maps, and Figure 8 shows the FF algorithm on a total-degree 4 map.

For our tests, we incremented n by 50 until either the algorithm took longer than 300 seconds or until $n = 1900$. For $n = 1900$ the complete partition has more than 10^9 boxes which should cover the range of interest. Due to the time cutoff, several of the plots (Figures 16, 17) have lines that do not extend all the way to $n = 1900$. All experiments were run an Intel i7-6700K processor running at 4.0 GHz with 16GB of RAM.

Run time is compared in Figure 15. The complexity of the brute force algorithm is $O(n^3)$. While initially competitive thanks to the highly efficient Intersection Test, brute force becomes impractical for large n. The complexity of DFS and BFS is $O(n^2)$ and the complexity of the FF algorithm is $O(n^2 \log n)$. In practice, the DFS is the slowest of the remaining algorithms over this

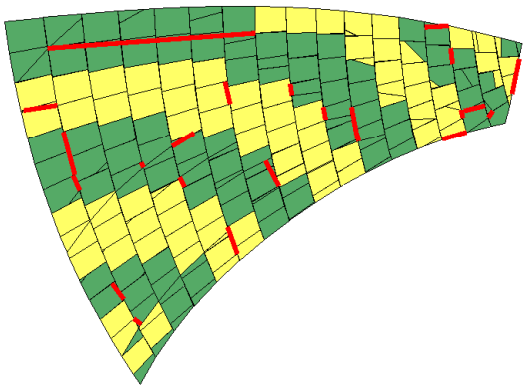

Figure 18: The minimum distance between consecutive non-adjacent intersections is shown in red. The *jump length* is the sum of all red segments. .

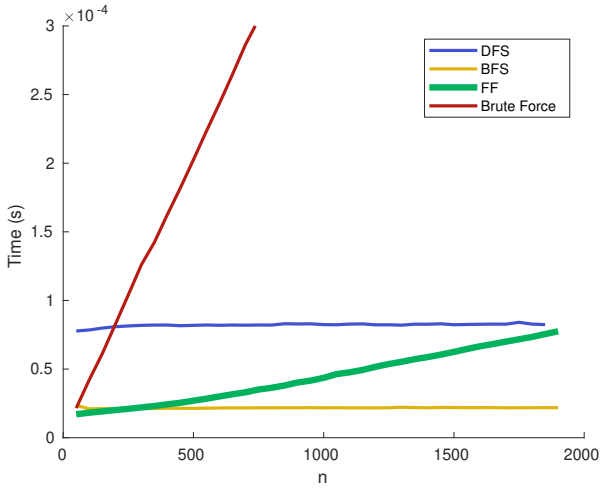

Figure 19: Average time per box as a function of n, computed as the total time to run divided by the total number of boxes in the intersection. The times for DFS, BFS and FF are below 10^{-4} seconds.

range, but due to the extra $\log n$ term, the FF algorithm overtakes it for larger n. BFS is fastest.

Storage costs are compared in Figure 16. As expected, due to storage for backtracking, the DFS uses excessive storage. Brute Force requires no storage and is therefore space optimal (but not practical due to run time). BFS and FF require $O(n)$ storage for the front with an additional constant factor for FF due to maintaining the fat front.

The jump lengths are compared in Figure 17. The jump length of a slice is illustrated by the red line segments in Figure 18. As expected, the BFS algorithm performs significantly worse than all others. FF performs best of the four.

Time per box is compared in Figure 19. Time per box is the total run time divided by the total number of boxes in the slice. DFS, FF, and BFS all take less than 10^{-4} seconds per box, with DFS and BFS nearly constant regardless of n. Time per box for the FF algorithm grows as $O(\log n)$, but since $\log_2(1900) < 11$, this is not prohibitive for the range of interest. For DFS, BFS, and FF, the time per box is well under the expected time for the printer to process the underlying micro-structure.

The best choice of algorithm depends on whether one metric is critical for a specific use case. In cases where memory management is of utmost importance, the only viable option is the brute-force method since it is the only algorithm with storage costs that do not grow with the size of the input. However, this algorithm has prohibitive run time for large inputs and longer jump lengths than either DFS and FF. The DFS algorithm does not perform best under any metric; and DFS has prohibitively high storage cost for large inputs. The BFS enumeration is fastest, but its jump length leads to excessive printer head motion. FF has much lower jump length.

For the majority of use-cases the FF algorithm is preferred: FF has the smallest jump length, runs nearly as fast as the BFS algorithm, and does not have excessive storage costs.

8. Conclusions

The two contributions of this paper are to efficiently select domain boxes of mapped micro-structure near a plane of interest and to organize the traversal of all boxes in a monotone fashion without large jumps in box location. (Reducing the jump length to zero requires computing a Hamiltonian path, which is only possible for special planar graphs.) By choice of the brick layer's partition, the implementation applies to both simplex and cube domains Δ.

The fat front (FF) algorithm excels for large slicing problems with $O(n^3)$ boxes and index storage limited to $O(n)$. Only local neighbors need ever be tested to generate all boxes whose micro-structure images can intersect the plane.

Acknowledgements. This work was supported in part by DARPA HR00111720031, NSF grant DMS1564480 (Sitharam) and NIH R01 EB018625 (Peters)

References

[1] O. Fogel, S. Winter, E. Benjamin, S. Krylov, Z. Kotler, Z. Zalevsky, 3d printing of functional metallic microstructures and its implementation in electrothermal actuators, Additive Manufacturing 21 (2018) 307 – 311. doi:https://doi.org/10.1016/j.addma.2018.03.018.
URL http://www.sciencedirect.com/science/article/pii/S2214860418300861

[2] A. Vyatskikh, S. Delalande, A. Kudo, X. Zhang, C. Prtela, J. Greer, Additive manufacturing of 3d nano-architected metals, Nature Communicationsdoi:10.1038/s41467-018-03071-9.
URL https://doi.org/10.1038/s41467-018-03071-9

[3] Y. Wang, L. Zhang, S. Daynes, H. Zhang, S. Feih, M. Y. Wang, Design of graded lattice structure with optimized mesostructures for additive manufacturing, Materials & Design 142 (2018) 114 – 123. doi:https://doi.org/10.1016/j.matdes.2018.01.011.
URL http://www.sciencedirect.com/science/article/pii/S026412751830011X

[4] B. Bickel, M. Bächer, M. A. Otaduy, H. R. Lee, H. Pfister, M. Gross, W. Matusik, Design and fabrication of materials with desired deformation behavior, ACM Trans. Graph. 29 (4). doi:10.1145/1778765.1778800.
URL https://doi.org/10.1145/1778765.1778800

[5] C. Schumacher, B. Bickel, J. Rys, S. Marschner, C. Daraio, M. Gross, Microstructures to control elasticity in 3d printing, ACM Trans. Graph. 34 (4). doi:10.1145/2766926.
URL https://doi.org/10.1145/2766926

[6] J. Panetta, Q. Zhou, L. Malomo, N. Pietroni, P. Cignoni, D. Zorin, Elastic textures for additive fabrication, ACM Trans. Graph. 34 (4). doi:10.1145/2766937.
URL https://doi.org/10.1145/2766937

[7] S. K. Saha, D. Wang, V. H. Nguyen, Y. Chang, J. S. Oakdale, S.-C. Chen, Scalable submicrometer additive manufacturing, Science 366 (6461) (2019) 105–109. arXiv:https://science.sciencemag.org/content/366/6461/105.full.pdf, doi:10.1126/science.aax8760.
URL https://science.sciencemag.org/content/366/6461/105

[8] L. Jonušauskas, D. Gailevičius, S. Rekštytė, T. Baldacchini, S. Juodkazis, M. Malinauskas, Mesoscale laser 3d printing, Opt. Express 27 (11) (2019) 15205–15221. doi:10.1364/OE.27.015205.
URL http://www.opticsexpress.org/abstract.cfm?URI=oe-27-11-15205

[9] T. W. Sederberg, S. R. Parry, Free-form deformation of solid geometric models, in: SIGGRAPH, 1986, pp. 151–160.

[10] F. Massarwi, J. Machchhar, P. Antolin, G. Elber, Hierarchical, random and bifurcation tiling with heterogeneity in micro-structures construction via functional composition, Comput. Aided Des 102 (2018) 148–159.

[11] M. Sitharam, J. Youngquist, M. Nolan, J. Peters, Corner-sharing tetrahedra for modeling micro-structure, in: P. A. Xin (Shane) Li, Yong-Jin Liu (Ed.), Symposium on Solid and Physical Modeling 2019, Solid Modelling Association, 2019, pp. 1–14.

[12] L. Feng, P. Alliez, L. Busé, H. Delingette, M. Desbrun, Curved optimal delaunay triangulation, ACM Trans. Graph. 37 (4). doi:10.1145/3197517.3201358.
URL https://doi.org/10.1145/3197517.3201358

[13] J. Lee, K. Lee, Block-based inner support structure generation algorithm for 3d printing using fused deposition modeling, The International Journal of Advanced Manufacturing Technology 89. doi:10.1007/s00170-016-9239-3.

[14] S. Hornus, T. Kuipers, O. Devillers, M. Teillaud, J. Martínez, M. Glisse, S. Lazard, S. L. 0001, Variable-width contouring for additive manufacturing, ACM Trans. Graph 39 (4) (2020) 131.

[15] J. Kopf, C.-W. Fu, D. Cohen-Or, O. Deussen, D. Lischinski, T.-T. Wong, Solid texture synthesis from 2D exemplars, ACM Transactions on Graphics 26 (3) (2007) 2:1–2:?? doi:https://doi.org/10.1145/1275808.1276380.

[16] X. Liu, V. Shapiro, Random heterogeneous materials via texture synthesis, Computational Materials Science 99 (2015) 177–189.

[17] K. Vidimče, S.-P. Wang, J. Ragan-Kelley, W. Matusik, Openfab: A programmable pipeline for multi-material fabrication, ACM Transactions on Graphics 32.

[18] K. Vidimce, A. Kaspar, Y. Wang, W. Matusik, Foundry: Hierarchical material design for multi-material fabrication, in: J. Rekimoto, T. Igarashi, J. O. Wobbrock, D. Avrahami (Eds.), Proceedings of the 29th Annual Symposium on User Interface Software and Technology, UIST 2016, Tokyo, Japan, October 16-19, 2016, ACM, 2016, pp. 563–574.

[19] A. Pasko, V. Adzhiev, A. Sourin, V. Savchenko, Function representation in geometric modeling: concepts, implementation and applications, The Visual Computer 11 (1995) 429–446. doi:10.1007/BF02464333.

[20] A. A. Pasko, V. Adzhiev, A. Sourin, V. V. Savchenko, Function representation in geometric modeling: concepts, implementation and applications, Vis. Comput 11 (8) (1995) 429–446.

[21] M. Ostoja-starzewski, Lattice models in micromechanics, App. Mech. Review 55 2002.

[22] C. Huet, Application of variational concepts to size effects in elastic heterogeneous bodies, Journal of the Mechanics and Physics of Solids 38 (6) (1990) 813 – 841. doi:https://doi.org/10.1016/0022-5096(90)90041-2.
URL http://www.sciencedirect.com/science/article/pii/0022509690900412

[23] A. Pasko, O. Fryazinov, T. Vilbrandt, P.-A. Fayolle, V. Adzhiev, Procedural function-based modelling of volumetric microstructures, Graphical Models 73 (5) (2011) 165 – 181. doi:https://doi.org/10.1016/j.gmod.2011.03.001.
URL http://www.sciencedirect.com/science/article/pii/S1524070311000087

[24] Y. Song, Z. Yang, Y. Liu, J. Deng, Function representation based slicer for 3d printing, Comput. Aided Geom. Des 62 (2018) 276–293.

[25] S. F. Frisken, R. N. Perry, A. P. Rockwood, T. R. Jones, Adaptively sampled distance fields: A general representation of shape for computer graphics, in: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 2000, pp. 249–254.

[26] P. Antolin, A. Buffa, E. Cohen, J. F. Dannenhoffer, G. Elber, S. Elgeti, R. Haimes, R. Riesenfeld, Optimizing micro-tiles in micro-structures as a design paradigm, Computer-Aided Design 115 (2019) 23 – 33. doi:https://doi.org/10.1016/j.cad.2019.05.020.
URL http://www.sciencedirect.com/science/article/pii/S0010448519301939

[27] A. Gupta, K. Kurzeja, J. Rossignac, G. Allen, P. S. Kumar, S. Musuvathy, Programmed-lattice editor and accelerated processing of parametric program-representations of steady lattices, Computer-Aided Design 113 (2019) 35 – 47. doi:https://doi.org/10.1016/j.cad.2019.04.001.
URL http://www.sciencedirect.com/science/article/pii/S0010448518303798

[28] Y. Tang, G. Dong, Y. Zhao, A hybrid geometric modeling method for lattice structures fabricated by additive manufacturing, International Journal of Advanced Manufacturing Technology (102). doi:10.1007/s00170-019-03308-x.

[29] J. Martínez, J. Dumas, S. Lefebvre, Procedural voronoi foams for additive manufacturing, ACM Trans. Graph. 35 (4) (2016) 44:1–44:12. doi:10.1145/2897824.2925922.
URL http://doi.acm.org/10.1145/2897824.2925922

[30] J. Martínez, S. Hornus, H.-C. Song, S. L. 0001, Polyhedral Voronoi diagrams for additive manufacturing, ACM Trans. Graph 37 (4) (2018) 129:1–129:15.

[31] T. J. R. Hughes, J. A. Cottrell, Y. Bazilevs, Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement, CMAME 194 (39-41) (2005) 4135–4195.

[32] L. Liu, Y. Zhang, T. J. R. Hughes, M. A. Scott, T. W. Sederberg, Volumetric T-spline construction using boolean operations, Eng. Comput. (Lond.) 30 (4) (2014) 425–439.
URL http://dx.doi.org/10.1007/s00366-013-0346-6

[33] M. OstojaStarzewski, Microstructural randomness and scaling in mechanics of materials (2007) 1–497doi:10.1201/9781420010275.

[34] M. C. Messner, A fast, efficient direct slicing method for slender member structures, Additive Manufacturing 18 (C). doi:10.1016/j.addma.2017.10.014.

[35] K. Kurzeja, J. Rossignac, Rangefinder: Accelerating ball-interference queries against steady lattices, Comput. Aided Des 112 (2019) 14–22.

[36] A. Wang, C. Zhou, Z. Jin, W. Xu, Towards scalable and efficient gpu-enabled slicing acceleration in continuous 3d printing, presented in 22nd Asia and South Pacific Design Automation Conference (ASP-DAC) in Chiba, Japan. (Jan 2017).
URL http://www.aspdac.com/aspdac2017/archive/pdf/7C-1_add_file.pdf

[37] X. Zhang, G. Xiong, Z. Shen, Y. Zhao, C. Guo, X. Dong, A gpu-based parallel slicer for 3d printing, in: 2017 13th IEEE Conference on Automation Science and Engineering (CASE), 2017, pp. 55–60. doi:10.1109/COASE.2017.8256075.
URL https://ieeexplore.ieee.org/document/8256075

[38] C. C. Dant, Design and implementation of asymptotically optimal mesh slicing algorithms using parallel processing, 2016.
URL https://www.semanticscholar.org/paper/Design-and-Implementation-of-Asymptotically-Optimal-Dant/915a1c4296bfa215c55acd4463e9b66672bda2a9

[39] S. Choi, K. Kwok, A tolerant slicing algorithm for layered manufacturing, Rapid Prototyping Journaldoi:10.1108/13552540210430997.
URL http://hdl.handle.net/10722/74432

[40] M. Vatani, A. Rahimi, F. Brazandeh, A. Sanati Nezhad, An enhanced slicing algorithm using nearest distance analysis for layer manufacturing, Vol. 37, 2009, pp. 721–726.

[41] R. Minetto, N. Volpato, J. Stolfi, R. M. Gregori, M. V. da Silva, An optimal algorithm for 3d triangle mesh slicing, Computer-Aided Design 92 (2017) 1 – 10. doi:https://doi.org/10.1016/j.cad.2017.07.001.
URL http://www.sciencedirect.com/science/article/pii/S0010448517301215

[42] Y. Sasaki, M. Takezawa, S. Kim, H. Kawaharada, T. Maekawa, Adaptive direct slicing of volumetric attribute data represented by trivariate b-spline functions, The International Journal of Advanced Manufacturing Technology 91. doi:10.1007/s00170-016-9800-0.

[43] T. W. Sederberg, S. R. Parry, Free-form deformation of solid geometric models, in: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '86, Association for Computing Machinery, New York, NY, USA, 1986, p. 151–160. doi:10.1145/15922.15903.
URL https://doi.org/10.1145/15922.15903

[44] G. Farin, Curves and Surfaces for CAGD: A Practical Guide, 5th Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[45] J. Peters, Evaluation and approximate evaluation of the multivariate Bernstein-Bézier form on a regularly partitioned simplex, ACM Transactions on Mathematical Software 20 (4) (1994) 460–480.

[46] S. Schaefer, J. Hakenberg, J. Warren, Smooth subdivision of tetrahedral meshes, in: Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, SGP '04, Association for Computing

Machinery, New York, NY, USA, 2004, p. 147–154. doi:10.1145/1057432.1057452.
URL https://doi.org/10.1145/1057432.1057452
[47] D. Filip, R. Magedson, R. Markot, Surface algorithms using bounds on derivatives, Computer Aided Geometric Design 3 (4) (1986) 295–311.
[48] G. Farin, Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide, Academic Press, 1988.
[49] J. Peters, X. Wu, Sleves for planar spline curves, Computer Aided Geometric Design 21 (6) (2004) 615–635.
[50] P. Sinha, E. Klassen, K. K. Wang, Exploiting topological and geometric properties for selective subdivision, in: Proceedings of the First Annual Symposium on Computational Geometry, SCG '85, Association for Computing Machinery, New York, NY, USA, 1985, p. 39–45. doi:10.1145/323233.323239.
URL https://doi.org/10.1145/323233.323239