Memory-Efficient Modeling and Slicing of Large-Scale Adaptive Lattice Structures

Liu, Shengjun; Liu, Tao; Zou, Qiang; Wang, Weiming; Doubrovski, Eugeni L.; Wang, Charlie C.L.

DOI
10.1115/1.4050290

Publication date
2021

Document Version
Final published version

Published in
Journal of Computing and Information Science in Engineering

Shengjun Liu
School of Mathematics and Statistics,
Central South University,
Changsha 410083, China
e-mail: shjliu.cg@csu.edu.cn

Tao Liu
School of Mathematics and Statistics,
Central South University,
Changsha 410083, China
e-mail: ltaocg@csu.edu.cn

Qiang Zou
State Key Laboratory of CAD&CG,
Zhejiang University,
Hangzhou 310058, China;
Department of Mechanical,
Aerospace and Civil Engineering,
University of Manchester,
Manchester M13 9PL, UK
e-mail: qiangzou@cad.zju.edu.cn

Weiming Wang
School of Mathematical Sciences,
Dalian University of Technology,
Dalian 116024, China;
Faculty of Industrial Design Engineering,
Delft University of Technology,
Delft 2628 DS, The Netherlands
e-mail: wwmdlut@dlut.edu.cn

Eugeni L. Doubrovski
Faculty of Industrial Design Engineering,
Delft University of Technology,
Delft 2628 DS, The Netherlands
e-mail: E.L.Doubrovski@tudelft.nl

Charlie C. L. Wang[1]
Department of Mechanical,
Aerospace and Civil Engineering,
University of Manchester,
Manchester M13 9PL, UK
e-mails: charlie.c.l.wang@manchester.ac.uk;
changling.wang@manchester.ac.uk

Memory-Efficient Modeling and Slicing of Large-Scale Adaptive Lattice Structures

Lattice structures have been widely used in various applications of additive manufacturing due to its superior physical properties. If modeled by triangular meshes, a lattice structure with huge number of struts would consume massive memory. This hinders the use of lattice structures in large-scale applications (e.g., to design the interior structure of a solid with spatially graded material properties). To solve this issue, we propose a memory-efficient method for the modeling and slicing of adaptive lattice structures. A lattice structure is represented by a weighted graph where the edge weights store the struts' radii. When slicing the structure, its solid model is locally evaluated through convolution surfaces in a streaming manner. As such, only limited memory is needed to generate the toolpaths of fabrication. Also, the use of convolution surfaces leads to natural blending at intersections of struts, which can avoid the stress concentration at these regions. We also present a computational framework for optimizing supporting structures and adapting lattice structures with prescribed density distributions. The presented methods have been validated by a series of case studies with large number (up to 100 M) of struts to demonstrate its applicability to large-scale lattice structures. [DOI: 10.1115/1.4050290]

Keywords: computational foundations for additive manufacturing, computer aided design, computer aided manufacturing

1 Introduction

Additive manufacturing has enabled the fabrication of objects with highly complicated shapes and structures. Recently, increasing attention has been drawn towards modeling interior structures of 3D models rather than the exterior appearance [1–3]. One important type of interior structures is the lattice structure, which consists of interconnected struts. Such structures are lightweight, yet have superior mechanical properties [4]. When such lightweight structures are used in vehicles, less energy will be consumed. Moreover, with carefully designed density distributions, spatially graded material properties can be realized at different regions of a 3D printed lattice structure, which therefore introduces varied mechanical properties in the part.

When using lattice structures to realize spatially graded material properties, the number of struts can be huge. In such a scenario, if triangular meshes (i.e., the de facto standard representation format in 3D printing) are to be used to represent lattice structures, the memory consumption can be extremely high. This poses a significant computational challenge in applications of large-scale, adaptive lattice structures. The previously developed out-of-core modeling algorithms [5,6] for lattice structures could handle uniform or periodical lattice structures but becomes difficult for processing large-scale adaptive lattice structures. There is also prior research [7] seeking to reduce the number of facets in triangulating a lattice structure. However, when a tremendous number of struts are involved to model lattice structures, the method will generate too many triangles to run on a general computer.

To address the aforementioned challenge of large-scale lattice structures in additive manufacturing (AM), we propose a memory-efficient method for modeling and slicing adaptive lattice structures in large-scales. A lattice structure is represented by a weighted graph with edges representing struts, nodes representing intersections of the struts, and edge weights specifying radii of the corresponding struts. The corresponding solid of the lattice structure is locally defined using convolution surfaces with compactly supported kernel functions. In this way, the solid's boundary surface will only be locally generated when needed. This yields a

[1]Corresponding author.

Manuscript received November 21, 2020; final manuscript received February 15, 2021; published online May 13, 2021. Assoc. Editor: Yong Chen.

highly scalable representation for adaptive lattice structures. In addition, solid models based on convolution surfaces have naturally blended shape at the intersections of struts, which can avoid the stress concentration at these regions. Based on the above representation scheme, a streaming-based slicing algorithm for 3D printing is then developed to demonstrate the scalability of our method—models with a massive number (up to 100 M) of struts can be successfully handled.

Based on the scalable modeling method, an algorithm for constructing a lattice structure model in accordance with a prescribed density distribution is also presented in this paper. The lattice structure is initialized with edges and nodes of a tetrahedral mesh generated by the method presented in Ref. [8]. Then, the density in the initial lattice structure is adjusted to match with the prescribed density in two steps as tetrahedra subdivision and strut radius adjustment. To facilitate the printing process, the lattice structure is further optimized to be self-supported for additive manufacturing.

The technical contributions of our work are:

(1) A new memory-efficient representation of lattice structures that can be employed to solve the large-scale modeling problem of spatially graded material properties.
(2) A slicing algorithm in streaming mode to realize the fabrication of lattice structures with a large number of struts.
(3) A computational framework to reversely design a lattice structure that matches a prescribed density distribution and achieves the optimal self-supporting. This framework, when combined with the above two contributions, provides a comprehensive and practical pipeline for modeling and slicing lattice structures.

The rest of this paper is organized as follows. After reviewing the related works in Sec. 2, we present the modeling method of large-scale lattice structures in Sec. 3, which is followed by introducing a streaming-based slicing algorithm in Sec. 4. The computational framework for constructing a lattice structure according to the required density distribution is then presented in Sec. 5. The effectiveness of our approach is evaluated in Sec. 6 and our paper ends with the conclusion.

2 Related Works

In the literature, there are numerous techniques of modeling complicated geometry for additive manufacturing applications. Here we only study the most relevant works. More comprehensive surveys can be found in Refs. [9–11].

Different from conventional methods [12–14] that generate a full infill inside a part, more and more research studies are being conducted to design distributed infill structures for functionally tailored 3D printing. Examples include fractal-like space filling curves/surfaces [3,15], adaptive rhombic grid [16,17], Voronoi diagram-based structure [1,18–20], beam-like structure [21,22], periodic tiles [23,24], procedural periodic tiles [2,25], and voxel-based structure [26]. Among these approaches, only the approaches presented in Refs. [1,3] provide a method to construct an infill-structure matching the given density field. However, the demand of self-supporting is not considered in Ref. [1], and thin-shell structures are employed in Ref. [3] which is not able to achieve high sparsity as the lattice structures studied in this work.

In general, the inner surfaces of a model are represented by two-manifold triangular meshes, which can be obtained from different representations, such as voxels in Ref. [26], tetrahedral meshes in Ref. [27], rhombic cells in Refs. [16,17], elliptic cylinders in Ref. [19], and extended distance fields in Refs. [1,2]. In this paper, we introduce a skeletonal representation of lattice structures by only storing the set of nodes and the edge-connectivity of a graph as skeletons. The solid can be efficiently and effectively evaluated from these skeletons by convolution surface with compactly supported kernel functions. Different from distance fields, the formulation of convolution surface provides highly smooth surfaces at the intersected regions of struts. This can avoid the stress concentration at those regions with sharp creases. In addition, due to local kernel functions used in the convolution surface, the computation in slicing algorithm can be evaluated in a streaming manner—i.e., with high scalability.

When additive manufacturing is applied to fabricate models with complicated geometry, supporting structures need to be added below the part with large overhang [28,29]. In general, the additional support will lead to the problems of hard-to-remove, surface damage, and additional cost of material and fabrication time. For the 3D printed models of spatially graded density, the additional support may change the designed density distribution. In literature, many approaches have been developed to reduce the usage of supporting structures. For example, some methods tried to compute an optimal printing direction to reduce the influence of the supporting structure [30,31]. Different supporting structures are designed to reduce the volume of material usage, such as the tree-structures proposed in Ref. [30] and the bridge structures proposed in Ref. [32]. Moreover, approaches have also been developed to generate completely self-supported infill structures [3,16,17,19,26]. As a design tool, we propose to conduct the strategy employed in Ref. [33] to deform a model to reduce the demand of support.

3 Modeling of the Lattice Structure

In this section, we will first present the representation of lattice structures and then formulate its implicit solid by the convolution surface with compactly supported kernel functions. After that, we study shape error at the joint regions which may have over-blending problems.

3.1 Implicit Solid Representation. A lattice structure is represented as a graph of interconnected skeletons Ω, which is stored as a complex-based data structure $\Omega = (\mathcal{V}, \mathcal{E})$ with a set of nodes and a set of edges being a simplified version of the data structure for general non-manifold objects [34]. For each $\mathbf{v}_i \in \mathcal{V}$, it defines the position of a node as $\mathbf{v}_i \in \mathcal{R}^3$. For an edge $e_j \in \mathcal{E}$, it is represented as a pair of vertices associated with the radius of the edge's corresponding strut as $e_j = (\mathbf{v}_s, \mathbf{v}_e, r_j)$. The solid of a lattice structure is formulated as an implicit surface defined around the skeletons.

Given the skeleton representation Ω of a lattice structure, its corresponding solid is formulated as

$$\mathcal{S}(\Omega) = \{\mathbf{p} \mid F(\mathbf{p}) \leq 0 \ (\forall \mathbf{p} \in \mathcal{R}^3)\} \quad (1)$$

where $F(\cdot)$ is an implicit function returning the value proportional to the distance between $\mathbf{p}$ and Ω. A straightforward solution is to use an offset *distance function* as

$$F(\mathbf{p}) = -D + \min_{\forall \mathbf{q} \in \Omega} \|\mathbf{q} - \mathbf{p}\| \quad (2)$$

with D being assigned as $D = r_j$ when the closest point of $\mathbf{p}$ is located on the edge e_j. This definition based on the distance function has two problems:

(1) The closest point of a query $\mathbf{p}$ is not unique—there may be multiple closest points. As a consequence, the value of D (i.e., the function value of $F(\mathbf{p})$) is not well-defined when different radii r_js are assigned to different edges in Ω.
(2) Sharp creases are formed at the boundary surface of solid due to the discontinuity of the distance function (see Figs. 1(*b*) and 1(*d*)). Concentrated stresses can be easily generated in these regions.

A formulation of $F(\cdot)$ based on the convolution surface is employed in our framework, which can essentially solve these two problems.

$$F(\mathbf{p}) = -C + \int_V h(\mathbf{x}) f(\mathbf{p} - \mathbf{x})\, \mathrm{d}V = -C + (f \otimes h)(\mathbf{p}) \quad (3)$$

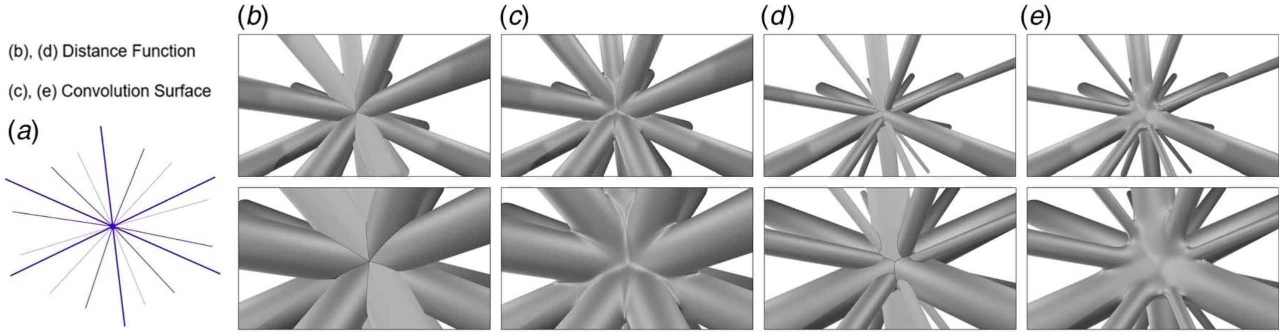

Fig. 1 For a given skeleton model as shown in (*a*), the solid models generated by distance field have sharp creases—see (*b*) using uniform radius and (*d*) using different radii for different edges of the skeleton. Differently, solids with smooth surfaces can be constructed by our approach in (c) and (e).

where $h:\mathcal{R}^3 \mapsto \mathcal{R}$ is a geometric function representing the skeleton Ω

$$h(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \in \Omega \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

and C is a constant isovalue defined according to the radii defined on the skeleton edges.

3.2 Representation With Compactly Supported Kernels.

To enable the modeling of lattice structures in large-scale, compactly supported kernels are employed in our method. The local support leads to local modification property, which is important for the fast computation of slicing in Sec. 4 and also the later density control in Sec. 5. Specifically, a quartic polynomial kernel function

$$f(\mathbf{p}-\mathbf{x}) = \begin{cases} \left(1 - \|\mathbf{p}-\mathbf{x}\|^2/R^2\right)^2, & \|\mathbf{p}-\mathbf{x}\| < R \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

with R being the support size is adopted and its evaluation is very simple and efficient, as already noted in Ref. [35]. Moreover, a local weight is assigned to each edge to specify the radius of its corresponding strut, as discussed in Ref. [36]. Therefore, the convolution surface can be defined as

$$F(\mathbf{p}) = -C + \sum_{e_i \in \Omega} r_i \int_{\mathbf{x} \in e_i} f(\mathbf{p}-\mathbf{x}) de_i \tag{6}$$

which can be efficiently evaluated. Specifically, for a query point $\mathbf{p}$, nonzero terms in Eq. (6) only contain the edges with distance to $\mathbf{p}$ less than R. Moreover, the convolution integral along an edge can be computed by an analytical close-from. Details are given below.

Without loss of generality, it is assumed that the line of e_i intersects with the sphere (centered at $\mathbf{p}$ and radius R) at two points $\mathbf{p}_1$ and $\mathbf{p}_2$ (see Fig. 2(*a*)).The valid segment of e_i inside the sphere can be defined in a parametric form as

$$\mathbf{x}(s) = (1-s)\mathbf{p}_1 + s\mathbf{p}_2 \quad (\forall\, s \in [s_1, s_2])$$

with

$$s_1 = \max\{0, (\mathbf{v}_s - \mathbf{p}_1)\cdot(\mathbf{p}_2-\mathbf{p}_1)/\|\mathbf{p}_2-\mathbf{p}_1\|^2\}$$
$$s_2 = \min\{1, (\mathbf{v}_e - \mathbf{p}_1)\cdot(\mathbf{p}_2-\mathbf{p}_1)/\|\mathbf{p}_2-\mathbf{p}_1\|^2\}$$

As a result, the field value at point $\mathbf{p}$ contributed by e_i can be computed as

$$\begin{aligned} F_{e_i}(\mathbf{p}) &= r_i \int_{e_i} f(\mathbf{p}-\mathbf{x}) de_i \\ &= r_i \int_{s_1}^{s_2} \left(1 - \|\mathbf{p}-\mathbf{x}(s)\|^2/R^2\right)^2 \mathrm{d}s \\ &= \frac{r_i}{15R^4}\left(3l^4s^5 - 15al^2s^4 + 20a^2s^3\right)\Big|_{s_1}^{s_2} \end{aligned} \tag{7}$$

with $l = \|\mathbf{p}_2 - \mathbf{p}_1\|$ and $a = (\mathbf{p}-\mathbf{p}_1)\cdot(\mathbf{p}_2-\mathbf{p}_1)$.

When the isovalue C in Eq. (6) is fixed, we can adjust the value of r_i to change the radius of strut generated from each skeleton edge e_i. While decreasing the value of r_i, a point with the same isovalue will be closer to e_i thus reducing the strut radius. As illustrated in

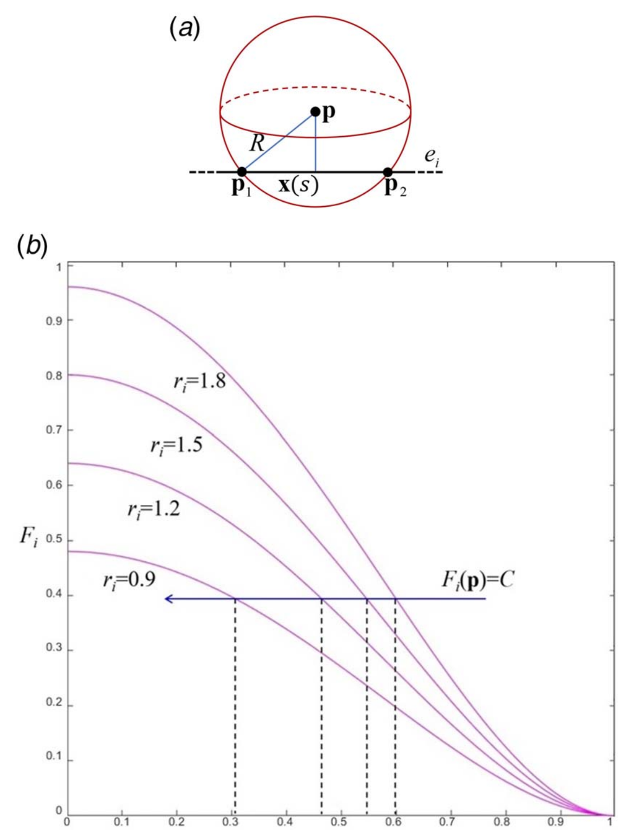

Fig. 2 Illustrations for computing the field value of F_i(p): (*a*) along an edge and (*b*) adaptive strut radius by changing r_i

Fig. 2(*b*), we can reduce the radius of strut along the direction of the blue arrow using smaller value of r_i for each kernel function $F_{e_i}(\mathbf{p})$. An example is given in Fig. 1(*e*) that represents a solid having the same skeleton as the one shown in Fig. 1(*c*) but using different radii for struts.

In our formulation, the support size of each kernel function is not changed even when using different values of r_is for different struts. The support size R for all kernel functions serves as the upper bound for the radii that can be realized by convolution. As a consequence, users can select the value of R by the range of strut radii that they wish to obtain (e.g., 1.5× the maximal radius used in our implementation).

Comparing to the convolution solid generated by globally defined kernels such as Gaussian [37,38] or by *fast Fourier transform* [39], the benefit of convolution by compactly supported kernel functions is twofold.

(1) The evaluation of implicit solid can be conducted in an out-of-core manner. For evaluating the function value at a point $\mathbf{p} \in \mathcal{R}^3$, only those kernel functions with the distance between $\mathbf{p}$ and their centers less than R will involve. Therefore, large-scale lattice structures can be efficiently modeled and sliced in our approach (see the streaming mode slicing algorithm presented in Sec. 4).

(2) A convolution surface constructed by compactly supported kernels will generate less over-blending artifacts in the region with many overlapped kernels, as demonstrated by the comparison in Figs. 3 and 4. The reason for this advantage is that overblending is often caused by including unwanted contribution from nearby kernels when computing convolutions, and the limited range of compactly supported kernels can prevent such unwanted contribution to some degree.

Both advantages are very important for efficiently and effectively modeling large-scale adaptive lattice structures.

4 Slicing

In this section, we present the method for slicing the solid of a lattice structure efficiently in both memory and computational time. Benefiting from the locally supported representation of solids as formulated in Eq. (6), only the skeleton edges with its swept sphere bounding volume [40] intersecting the slicing plane $\mathcal{P}(z) = \{\mathbf{p} \mid \mathbf{p}^z = z (\forall \mathbf{p} \in \mathcal{R}^3)\}$ will contribute to the field value of $F(\mathbf{p})$. In practice, this can be detected by a simple condition on the z-value of an edge's two endpoints. A set of intersected edges, denoted by $\mathcal{E}_{act}$, can be obtained as

$$\mathcal{E}_{act}(\mathcal{P}(z)) = \{e_j = (\mathbf{v}_s, \mathbf{v}_e, r_j) \mid (\mathbf{v}_s^z - R \leq \mathbf{p}^z) \cap (\mathbf{v}_e^z + R \geq \mathbf{p}^z)\} \quad (8)$$

where we have $\mathbf{v}_s^z \leq \mathbf{v}_e^z$ without loss of generality. An algorithm in the streaming mode can be used to generate slices for 3D printing a lattice structure represented by our method (as shown Fig. 5), which is a variant of the scanning line algorithm [41] and the sweeping plane algorithm [42]. Differently, we are searching for kernels which contribute to the function value $F(\mathbf{p})$ with $\mathbf{p}^z = z$. The spherical swept volumes of convolution kernels using the support size R as sphere radius are considered in our algorithm.

Two lists of edges are constructed in our algorithm: (1) a list of all edges $\mathcal{E}$ in Ω sorted by the z-coordinates (in an ascending order) of their "lower" vertices and (2) a list of active edges $\mathcal{E}_{act}$ according to a slicing plane $\mathcal{P}(z)$. With the help of $\mathcal{E}_{act}$, the field value for a point $\mathbf{q} \in \mathcal{P}(z)$ can be computed by only using the edges in $\mathcal{E}_{act}$ and using Eq. (6). Therefore, a binary image with a user specified resolution can be generated by efficiently evaluating if $F(\mathbf{q}) \leq 0$ (inside the solid) or $F(\mathbf{q}) > 0$ (outside the solid). The resultant binary image can be directly applied to the *digital light processing* (DLP)-based 3D printer [43]. Resultant binary images of example models can be found in Fig. 14 in Sec. 6.

When changing the slicing plane from $\mathcal{P}(z)$ to $\mathcal{P}(z+t)$ with t being the layer thickness, we need to update the list of active edges. As all the edges in $\mathcal{E}$ have been sorted in an ascending order by the z-coordinate of their first vertex, the new list of active edges can be efficiently obtained if we still record the index of the first remaining edge in $\mathcal{E}$. We search the edges in $\mathcal{E}$ starting from this one until reaching an edge the swept solid of which is completely above $\mathcal{P}(z+t)$. Steps of our slicing algorithm are given as follows:

Step 1: Initializing $k = 1$, $z = t/2$, and $\mathcal{E}_{act} = \emptyset$.

Step 2: Repeatedly checking edges $e_j \in \mathcal{E}$ with $j = k, k+1, \ldots, m$ until reaching an edge $e_m = (\mathbf{v}_p, \mathbf{v}_q, r_m)$ with $\mathbf{v}_p^z - R > z$ (i.e., the swept solid of e_m is completely above $\mathcal{P}(z)$) and adding all edges $e_{j, j=k,\ldots,m-1}$ into $\mathcal{E}_{act}$.

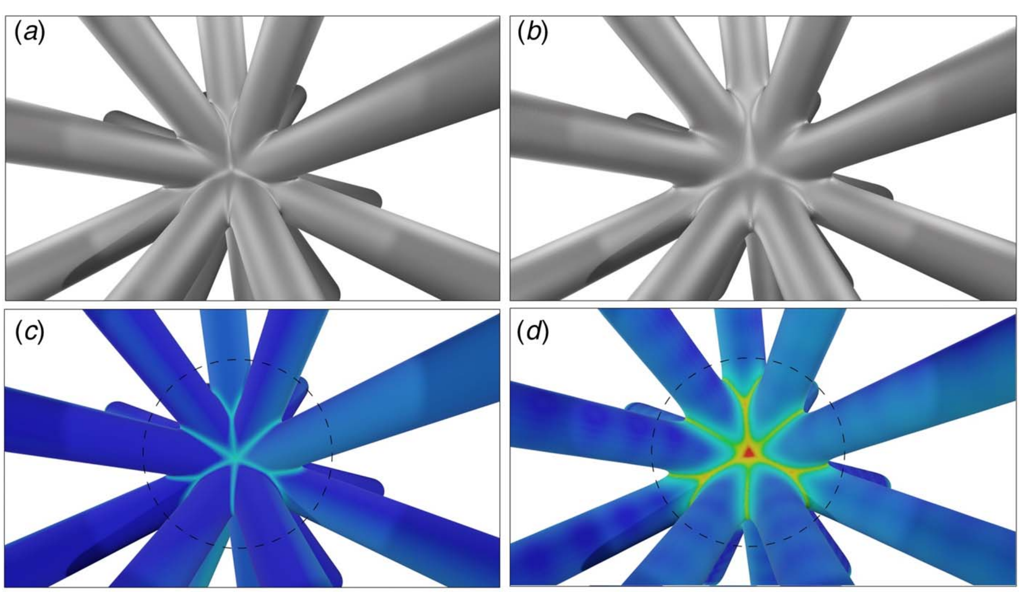

Fig. 3 Comparison of solids generated by (*a*) our approach and (*b*) the Gaussian kernel—both for the skeletons shown in Fig. 1(*a*). It can clearly be observed that more over-blending artifacts are generated on the result of Gaussian kernel. The gray level in (*c*) and (*d*) are used to visualize the distance between implicit surfaces and the "ideal" solid as the union of cylinders.

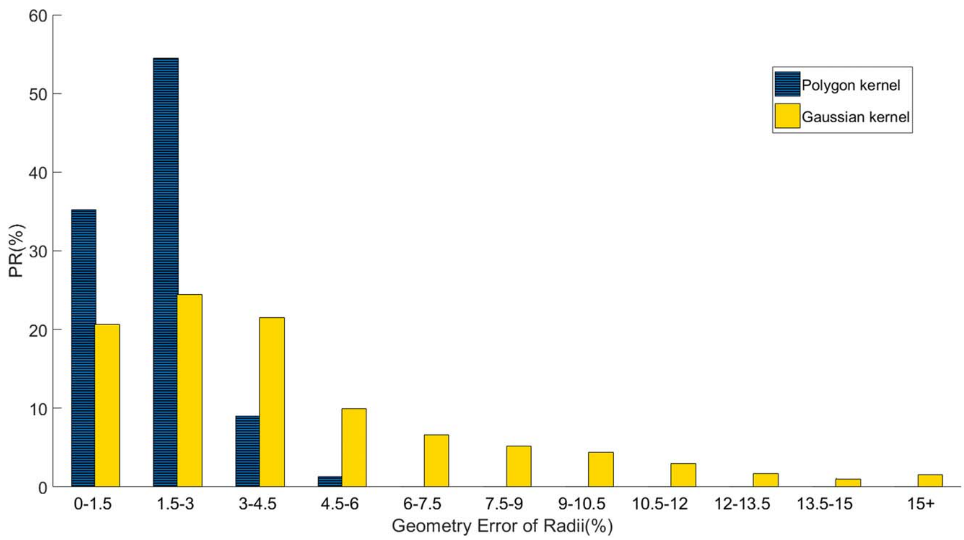

Fig. 4 To visualize the surface distance errors, we generate sample points in a spherical region with radius as 2.83*R*—the circled region shown in Figs. 3(*c*) and 3(*d*). Histogram of distances between sample points to the "ideal" solid is given. For the result with Gaussian kernel, around 40% sample points have the distance more than 6%*r*. Differently, our result has no sample point with distance more than 6%*r*.

Step 3: Checking all edges in $\mathcal{E}_{act}$ and removing an edge $e_i = (\mathbf{v}_s, \mathbf{v}_e)$ from $\mathcal{E}_{act}$ if $\mathbf{v}_e^z + R < z$ (i.e., its swept solid is completely below $\mathcal{P}(z)$).

Step 4: Evaluating the field value of points on the slicing plane $\mathcal{P}(z)$ using the kernels defined on all edges in $\mathcal{E}_{act}$.

Step 5: Let $z = z + t$ and $k = m$.

Step 6: Go back to *Step 2* until all planes have been sliced.

The memory usage of our slicing algorithm is very limited. In summary, only the edges in $\mathcal{E}_{act}$ and the binary image for a slicing plane need to be stored in the main memory. The rest skeleton edges (i.e., $\mathcal{E}$) for a lattice structure can be processed in an out-of-core manner. Edges are sorted by external sorting, which also has a lightweight memory consumption. Detailed statistics of memory consumption (without counting the external sorting), for example, lattice structures, will be provided in Sec. 6.

Our slicing algorithm can also be applied to generate G-code of tool-path for filament-based deposition or laser sintering. First of all, a marching square algorithm [44] is applied to a binary image to generate the rough boundary curves of a model. Then, the intersection-free smoothing and simplification algorithm [45] can be applied to generate topology-preserved boundary curves of the model. The combination of the binary image and the marching square algorithm can guarantee that no degenerated case will be given in slicing, as long as the resolution of the image is large enough [45]. The bone model shown in Fig. 16 is fabricated by

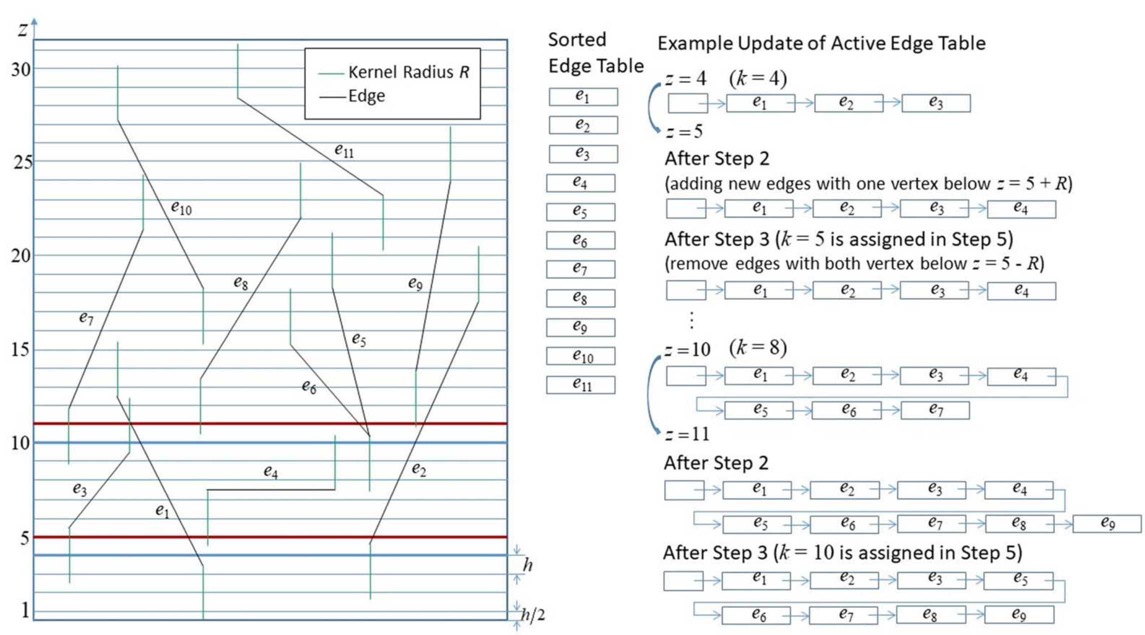

Fig. 5 The scanning plane algorithm can efficiently generate slides for fabricating the lattice structure represented by our method in a streaming mode—only the skeleton edges with their swept solids intersecting with the slicing plane need to be processed

tool-paths generated in this method on a selective laser melting (SLM)-based 3D metal printer.

5 Computation for Spatially Graded Density

This section presents a framework to automatically generate a lattice structure according to an input distribution of density. Given the voxel representation $\mathcal{V} = V_{i,j,k}$ of a given mesh model $\mathcal{M}$, the required density can be specified for each voxel $V_{i,j,k}$ as $\rho_{i,j,k}$. The problem to be solved is to construct a lattice structure Ω inside $\mathcal{M}$ so that the density of its corresponding solid $\mathcal{S}(\Omega)$ inside $V_{i,j,k}$ satisfies

$$\rho(\mathcal{S}(\Omega) \cap V_{i,j,k}) \approx \rho_{i,j,k} \quad (9)$$

where the density $\rho(\mathcal{S}(\Omega) \cap V_{i,j,k})$ is evaluated by the *Monte-Carlo integration*, whether a sample point inside the solid $\mathcal{S}(\Omega)$ can be evaluated by the implicit function defined in Eq. (6) efficiently.

5.1 Overview.

Our framework for generating spatially graded density consists of three major parts, which are explained below with the help of illustration given in Fig. 6.

(1) *Adaptive surface/tetrahedral mesh generation:* For a given mesh model $\mathcal{M}$, we first estimate the target edge-length in different regions on its surface. An adaptive remeshing approach akin to Ref. [46] is conducted to generate a surface adaptive mesh. Specifically, for using a material with density τ to fabricate a lattice structure with initial strut radius r for realizing a target density ρ, the target length $\bar{L}$ of a tetrahedron can be roughly estimated as

$$\bar{L} = \max\{4r, L_{ini}\} \quad (10)$$

where L_{ini} is a solution of the density estimate formula (Eq. (19)) that is closest to the average edge-length of the initial given mesh. Detailed calculation can be found in Appendix 1. The target lengths at different surface regions are computed by the above equation to control the result of surface remeshing. After that, a tetrahedral mesh generation method [8] is applied to construct the volumetric mesh adaptive to the surface mesh. Note that the step of surface remeshing is very important because it is difficult to generate a locally coarse tetrahedral mesh if the surface mesh is dense. After this step, edges of the tetrahedral mesh are employed to generate the initial lattice structure Ω.

(2) *Optimization for self-supporting:* When being fabricated by additive manufacturing, supporting structures need to be added below the large overhangs in Ω. This will change density on the finally fabricated models. Therefore, we develop an algorithm to improve the self-supporting of edges in Ω, which consists of two steps—the scaling/re-scaling and the vertex re-positioning. Details are given in Sec. 5.2.

(3) *Density matching:* After optimizing the self-supporting of a lattice structure, its density is further adjusted in the final phase of our framework to match the required density distribution. This is implemented by applying local subdivision on tetrahedra and local adjustment for the radii of struts. Details are given in Sec. 5.3.

With the help of this framework, we enable the inverse design of spatially graded density using lattice structures.

5.2 Optimization for Self-Supporting.

For additive manufacturing, supporting structures need to be added below the region with large overhang, which prolongs the printing process and wastes more material. More seriously, the supporting structures are hard-to-remove, and keeping these supporting structures will change the density distribution of a designed lattice structure. Therefore, it is important to reduce the demand of support by optimizing a design.

For all edges in a lattice structure $\Omega = (\mathcal{V}, \mathcal{E})$, we define a metric of self-supporting based on the projected area of risky regions that are facing down. For an edge $e_j = (\mathbf{v}_s, \mathbf{v}_e, r_j) \in \mathcal{E}$, whether it is fully self-supported depends on the angle $\theta(e_j)$ between it and the printing direction $\mathbf{t}_P$ as

$$\theta(e_j) = \arccos\left|\frac{\mathbf{v}_e - \mathbf{v}_s}{\|\mathbf{v}_e - \mathbf{v}_s\|} \cdot \mathbf{t}_P\right| \quad (11)$$

with $\mathbf{t}_P$ being a unit vector. For an edge satisfying $\theta(e_j) \leq \alpha$, the strut generated by this edge is fully self-supported. Here the angle α is the *self-supporting angle* that depends on the type of AM process and the materials used. In the rest of our paper, we conduct a widely used parameter $\alpha = \pi/4$. The set of fully self-supported edges is denoted as $\mathcal{E}_S$.

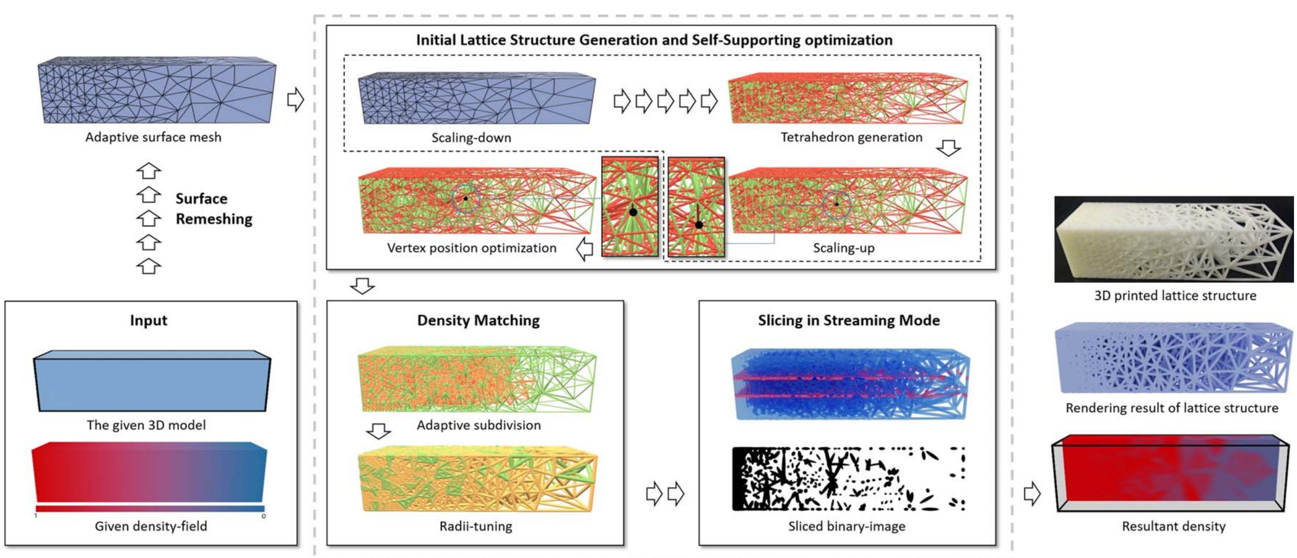

Fig. 6 The framework of our method to generate a lattice structure for spatial graded density consists of three major parts: (1) adaptive surface/tetrahedral mesh generation, (2) optimization for self-supporting, and (3) final density matching. By applying the slicing algorithm in the streaming mode, the binary images for every slides can be generated for additive manufacturing.

When $\theta(e_j) > \alpha$, a portion of the facing-down surface on the strut needs to be fabricated by adding supporting structures. For a cylindrical strut with skeleton as e_j, the projected facing down area is $A(e_j) = 2\ r_j\ L(e_j)$ with $L(e_j)$ being the length of e_j. According to our analysis on a cylinder, the percentage of projected area that needs to add support is a function $g(\theta)$ in terms of the angle θ between the cylinder's axis and the printing direction. Detailed analysis can be found in Appendix II. For the sake of computational simplicity, we conduct a polynomial to approximate $g(\theta)$. When $\alpha = \pi/4$, it is

$$g(\theta) \approx \begin{cases} \Sigma_{i=0}^{5} a_i \theta^i, & (\theta > \pi/4) \\ 0, & (\theta \leq \pi/4) \end{cases} \tag{12}$$

where $a_i = [-0.02, -0.31, 1.44, -1.11, 0.58, -0.16]$.

Based on this analysis, we define a metric of self-supporting for the lattice model as

$$\Gamma(\Omega) = \frac{\sum_{e_j \in (\mathcal{E} \setminus \mathcal{E}_S)} r_j L(e_j) g(\theta_j)}{\sum_{e_j \in \mathcal{E}} r_j L(e_j)} \tag{13}$$

which is the percentage of projected areas that needs additional support; the smaller the better. Besides of $\Gamma(\Omega)$, we also define a length percentage of completely self-supported struts as

$$\Psi(\Omega) = \frac{\sum_{e_j \in \mathcal{E}_S} L(e_j)}{\sum_{e_j \in \mathcal{E}} L(e_j)} \times 100\% \tag{14}$$

which is higher the better.

We develop two schemes to improve the value of $\Gamma(\Omega)$ on a lattice structure Ω: *scaling operation* (SO) and vertex *position optimization* (PO). As illustrated by the kitten model shown in Fig. 7, these schemes can effectively reduce the percentage of projected areas that need to add support in a lattice structure.

5.2.1 Scaling. For a given 3D printing direction $\mathbf{t}_P$ and an edge $e_j = (\mathbf{v}_s, \mathbf{v}_e)$ bounded in a box M as shown in Fig. 8, the angle θ between $\mathbf{t}_P$ and $\mathbf{v}_s\mathbf{v}_e$ will become large when they are not perpendicular. Without loss of generality, we can assume $\mathbf{t}_P = (0, 0, 1)$, $\mathbf{v}_s = (x_1, y_1, z_1)$, $\mathbf{v}_e = (x_2, y_2, z_2)$, and the scaling factor as k. After scaling, the new positions of the edge become $\mathbf{v}'_s = (x_1, y_1, z'_1)$ and $\mathbf{v}'_e = (x_2, y_2, z'_2)$ with $z'_1 = z_1/k$ and $z'_2 = z_2/k$. Then, we get

$$\begin{aligned} \theta(\mathbf{v}_s\mathbf{v}_e) &= \frac{\pi}{2} - \arctan\left(\frac{|z_2 - z_1|}{l}\right) \\ \theta(\mathbf{v}'_s\mathbf{v}'_e) &= \frac{\pi}{2} - \arctan\left(\frac{|z_2 - z_1|}{kl}\right) \end{aligned} \tag{15}$$

where $l = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. It is easy to find $\theta(\mathbf{v}_s\mathbf{v}_e) < \theta(\mathbf{v}'_s\mathbf{v}'_e)$ when $k > 1$.

Based on this analysis, we can improve the self-supporting of a lattice structure during its construction by scaling. Specifically, we first compress the space for constructing the lattice structure by a factor $1/k$ along the printing direction $\mathbf{t}_P$. After constructing the tetrahedral mesh as an initial lattice structure in the compressed space, we re-scale the mesh back to the original size by a factor k along the direction $\mathbf{t}_P$. Lattice structures constructed with the help of this scaling step have more self-supported edges (see Figs. 7(*a*) and 7(*b*) for an example). Using k with too large values will lead to very sparse vertices generated inside a model. Moreover, the quality of tetrahedra can be very poor when using a large k, which can be quantitatively evaluated by the aspect ratio on all tetrahedra. In short, the aspect ratio of a tetrahedron T_i is calculated by $R(T_i) = h_{\max}/h_{\min}$, where $h_{\max}$ and $h_{\min}$ are the maximum and the minimum distances from a vertex to its opposite face inside T_i. The ideal value of $R(T_i)$ is 1.0 and the quality of a mesh is considered as poor when many tetrahedra have the aspect ratio greater than 5.0. The histograms of aspect ratios for tetrahedral meshes generated using different k are studied to find a good balance between the quality of mesh and the level of self-supporting (see Fig. 9). According to this study, we usually employ $k \in [1.5, 1.8]$ in practice.

5.2.2 Vertex Re-Positioning. When moving a vertex $\mathbf{v}$, all edges linked to it (denoted the set as $\mathcal{E}_{\mathbf{v}}$) will be changed. Therefore, we can potentially move the vertices to generate more fully self-supported edges. We define an objective function to evaluate the self-supporting property around a vertex $\mathbf{v}$ as

$$J(\mathbf{v}) = \frac{\sum_{e_j \in \mathcal{E}_{\mathbf{v}}} r_i L(e_j) g(\theta_i)}{\sum_{e_i \in \mathcal{E}_{\mathbf{v}}} r_i L(e_i)} \tag{16}$$

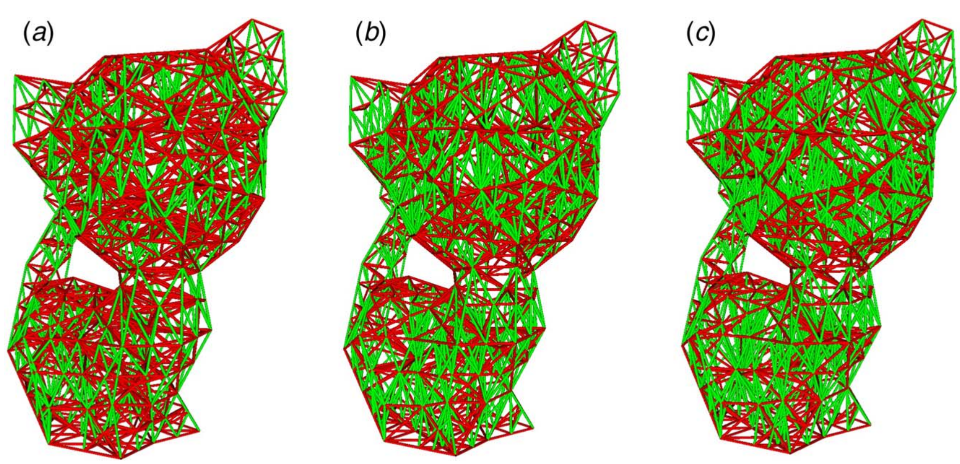

Fig. 7 An illustration for demonstrating the effectiveness of our algorithm for enlarging the percentage of self-supported edges, where edges need to add support are displayed. Two schemes, the SO and the vertex PO, are applied to the lattice structure Ω for a kitten model. The length percentages of completely self-supported edges are (*a*) $\Psi = 27.97\%$, (*b*) $\Psi = 41.67\%$, and (*c*) $\Psi = 49.38\%$, respectively. When the same radius is employed for all struts, the metric of self-supporting can be significantly reduced from (*a*) $\Gamma = 0.6275$ to (*b*) $\Gamma = 0.4830$, and then to (*c*) $\Gamma = 0.3726$.

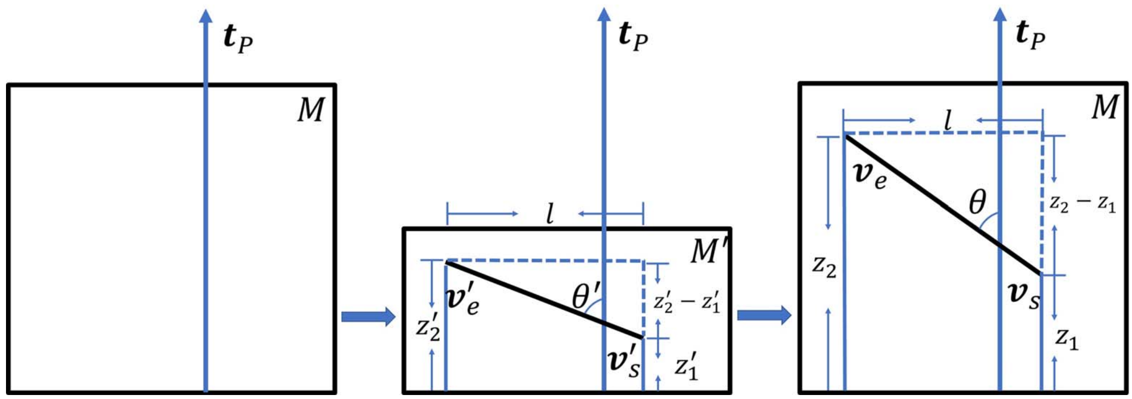

Fig. 8 The edges with large overhang generated in a scaled model (with height collapsed along the printing direction $\mathbf{t}_P$—as shown in the middle) have good chance to become self-supported after being scaled back to the model's original height

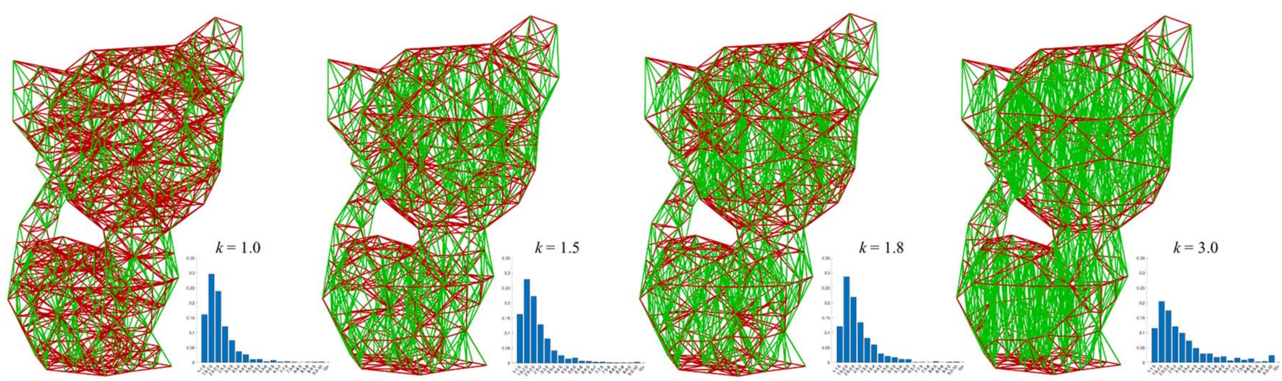

Fig. 9 A study to find the balance between the level of self-supporting (can be enhanced using large *k*) and the quality of constructed tetrahedral mesh (will be reduced using large *k*). From left to right, the length percentage of completely self-supported edges Ψ are 27.97%, 41.67%, 52.38%, and 79.28%, respectively (from left to right). When the same radius is used for all struts, the metric of self-supporting Γ gives the values of 0.6275, 0.4230, 0.3316, and 0.1271 (from left to right).

which need to be minimized. Moreover, in order to prevent intersections between tetrahedra, the new position of $\mathbf{v}$ and $\bar{\mathbf{v}}$ will be confined in a limited space. The optimization is formulated as

$$\begin{aligned} \bar{\mathbf{v}} &= \arg\min_{\mathbf{v}} J(\mathbf{v}) \\ \text{s.t. } & \|\mathbf{v} - \mathbf{o}\| \leq \tau \end{aligned} \tag{17}$$

where $\mathbf{o}$ and τ are the center and the radius of an inscribed sphere of the polyhedron formed by the vertices incident to the vertex $\mathbf{v}$. When solving the above optimization problem, the movement of vertex is conducted along the gradient direction. Together with the step length in every iteration, it determines how the vertices should be moved to improve the self-supporting property of the lattice.

The optimization is randomly applied to all the interior vertices one by one. The iteration stops when (1) no more vertex can move, (2) no more vertex's movement can reduce the value of $\Gamma(\mathbf{v})$, or (3) the maximum number of iterations (i.e., 100 used in our implementation) have been reached. Figure 10(*a*) gives a histogram chart to show the distribution of angles between edges and the printing direction $\mathbf{t}_P$, where the vertical axis gives the percentage of edges in terms of length. Moreover, it is interesting to study the aspect ratios of tetrahedra before and after PO. As shown in Fig. 10(*b*), the distributions of aspect ratios do not change too much, which is benefited by constraining the magnitude of movement in Eq. (17).

5.3 Density Matching. To explore the porosity of lattice structure, we initially define the radius $r_i = 2r_{\min}$ on all edges $\{e_i\}$ with $r_{\min}$ being the smallest feature size that can be reliably fabricated on a 3D printer. The radii will be adjusted for density matching below.

To satisfy the density distribution that is already defined as input of voxel-based function values (i.e., Eq. (9)), two operators are developed to adjust the density given on a lattice structure.

(1) *Subdivision*: When the density of material inside a tetrahedron (formed by the struts on its six edges) is not large enough, we would subdivide a tetrahedron T_i into eight tetrahedra by splitting in the middle of each edge (as illustrated in Fig. 11). From the geometry of tetrahedra obtained after subdivision, it can be observed that the newly created edges (except the unparallel one in Fig. 11) are always parallel to one of the six edges on T_i. As a result, the responding edges will be self-supported if the original edges of T_i are. The total length of edges is increased from $\sum_{i=1}^{6} l_i$ (with l_is being the length of a tetrahedron's six edges) to $l_r + 2\sum_{i=1}^{6} l_i$ with l_r being length of the unparallel edge as shown in Fig. 11. Therefore, the density in the volume of the original tetrahedron is more than doubled when using the same radii for the struts of newly generated edges.

(2) *Radii-tuning*: When the density of material inside a tetrahedron T_i is larger than a target value, we match the designed density by reducing the value of r_k on every edge $e_k \in T_i$. Note that, when changing r_k, the radius of strut on e_k is

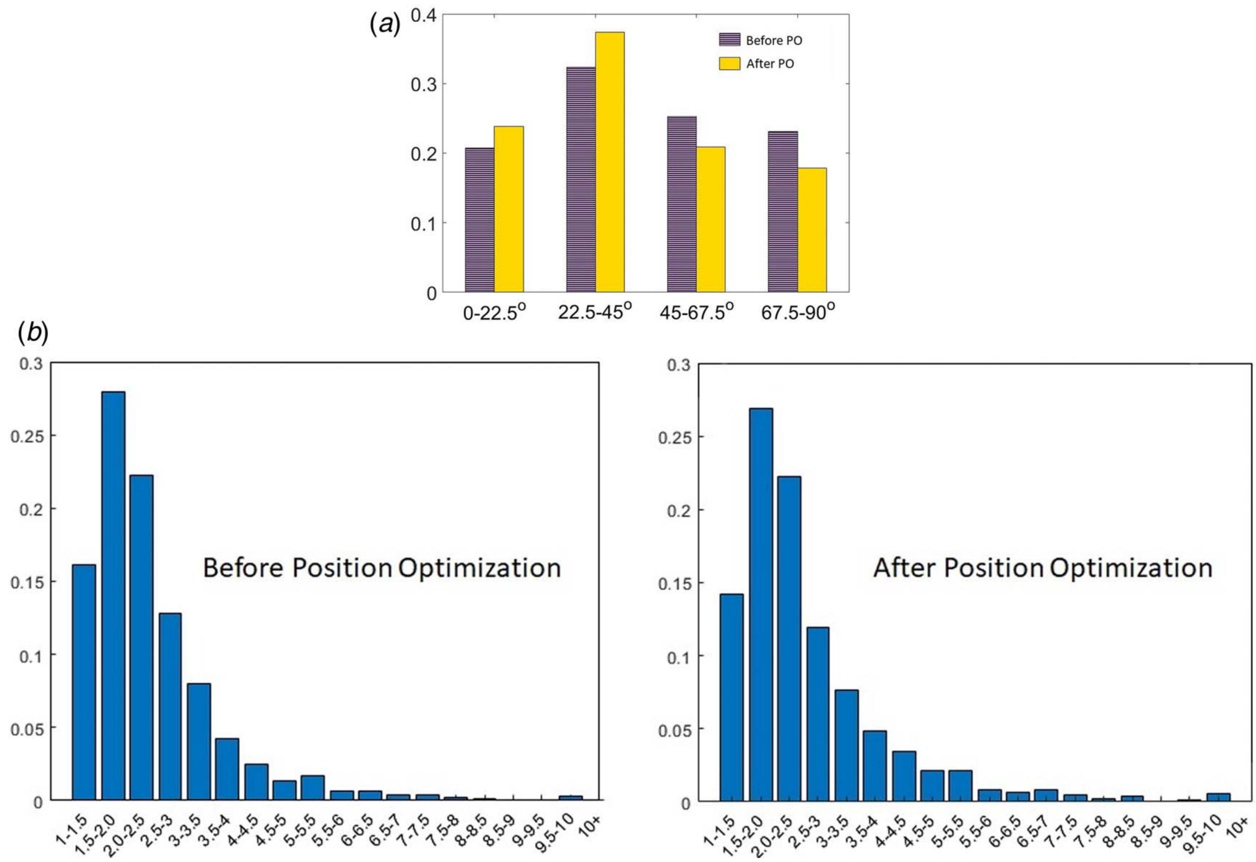

Fig. 10 Statistic visualization for (*a*) the histogram of angles between edges and the printing direction and (*b*) the aspect ratio of tetrahedra before versus after PO

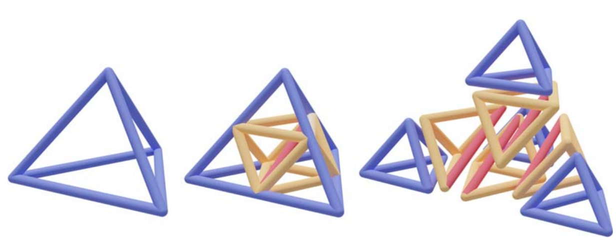

Fig. 11 The subdivision operator for density matching—the density can be increased by more than double after subdivision. Most of the newly generated edges are support-free, except for the unparallel one in center (which is kept in the structure).

changed monotonically but not linearly (see Fig. 2(*b*) for an illustration). Note that the minimal radii of edges should be bounded by the smallest feature size that can be fabricated by 3D printing.

These two operators are applied in our algorithm to generate a lattice structure matching the desired density distribution.

Our algorithm generates the density distribution of a lattice structure according to the input in four steps:

(1) First, we assign the desired density $\bar{\rho}_T$ to every tetrahedron T by the target density given to voxels in T—the maximal one is employed when T_i contains more than one voxels. Here, $V_{i,j,k}$ is considered as being contained by T when its center is inside T.
(2) For each tetrahedron T, we denote its current density as ρ_T and its density after making all edges' radii doubled as ρ_T^*. When $\rho_T^* < \bar{\rho}_T$, we subdivide this tetrahedron into eight tetrahedra recursively until this condition is satisfied in every new tetrahedron. For a small tetrahedron does not contain the center of any given voxel, the desired density is evaluated at the center of this tetrahedron by linearly interpolating the densities given at the centers of voxels.
(3) For each tetrahedron T, if it satisfies $\rho_T < \bar{\rho}_T < \rho_T^*$, we amplify the radii of edges on this tetrahedron using the binary searching strategy to match the target density $\bar{\rho}_T$.
(4) Lastly, we check the scaled radii of all edges in every voxel to generate a more accurate density matching. Specifically, for every voxel $V_{i,j,k}$, we try to determine a common radii-scaling ratio for all edges intersected with this voxel so that the density $\rho_{i,j,k}$ is matched more accurately. If the determined scaling ratio is less than the current ratio stored on an edge, the ratio is updated by the newly determined one. After checking all voxels, the struts of all edges are scaled by the most updated ratios.

After applying these four steps of our algorithm, the material density given on the lattice structure can well match the desired density distribution.

6 Results and Discussion

The approach described in the previous sections has been implemented using C++ based on a 4.00 GHz Intel Core i7-4790K and 16GB memory. Using this implementation, a variety of case studies and comparisons will be presented in this section to validate the approach.

6.1 Memory-Efficient Representation. Traditionally, lattice structures are represented as triangular meshes to allow easy

Table 1 Statistic of triangular meshes generated by MC

Models	Bunny	Bone	Finger	Kitter-HR
No. of struts	123,610	75,484	359,212	14,063,027
Approx. Err.	5% of strut radius $r = 0.5$ mm			
Box size	0.0847 mm × 0.0847 mm × 0.0847 mm			
Res. of MC	1368 × 1368 × 1447	1384 × 1384 × 1826	904 × 904 × 1758	2864 × 2864 × 2788
No. of triangles	355.7M	304.9M	226.2M	2359M

Table 2 Comparisons of float numbers (or file sizes) for different algorithms

Models	Bunny	Bone	Finger	Kitty-HR
No. of struts	123,610	75,484	359,212	14,063,027
No. of triangles[a] (unit: 10^6)				
MC	355.7	304.9	226.2	2358.6
LSLT	26.91	19.69	27.83	258.7
File size (unit: MB)				
MC	12,551.2	10,759.1	789.17	83,225.4
LSLT	949.2	695.1	980.9	9128.5
Ours	3.36	0.76	10.7	475.63

[a]Note that the number of triangles generated by LSLT are estimated according to their ratios to the result of MC, which are provided by Chougrani et al. [7].

integration into the 3D printing pipeline [9]. To convert an implicitly represented lattice structure (e.g., the scheme used in this work) to a triangular mesh, the *marching cubes* (MCs) method [47] is often used. However, for large-scale lattice structures, the MC method produces a huge number of triangles. As can be found from Table 1, 226–2359M triangles were generated for the models tested when setting the approximation error as 5% of the strut radii during triangulation.

Recently, a new method named *lattice structure lightweight triangulation* (LSLT) was proposed to triangulate lattice structures [7]. This method can reduce the number of the generated triangles, as shown by the statistics in Table 2. Nevertheless, it still generated too many triangles (26–258M) for the tested models when the number of struts increases significantly. As a result, the memory consumption of this method is still too large for slicing and toolpath planning algorithms to run properly.

By contrast, the proposed method can significantly reduce the memory consumption (see the file size block given in Table 2). This is essentially achieved by completely dropping the explicit triangular mesh representation scheme and, instead, by representing lattice structures implicitly using convolution surfaces with line segments as skeletons. The corresponding solid models of the lattice

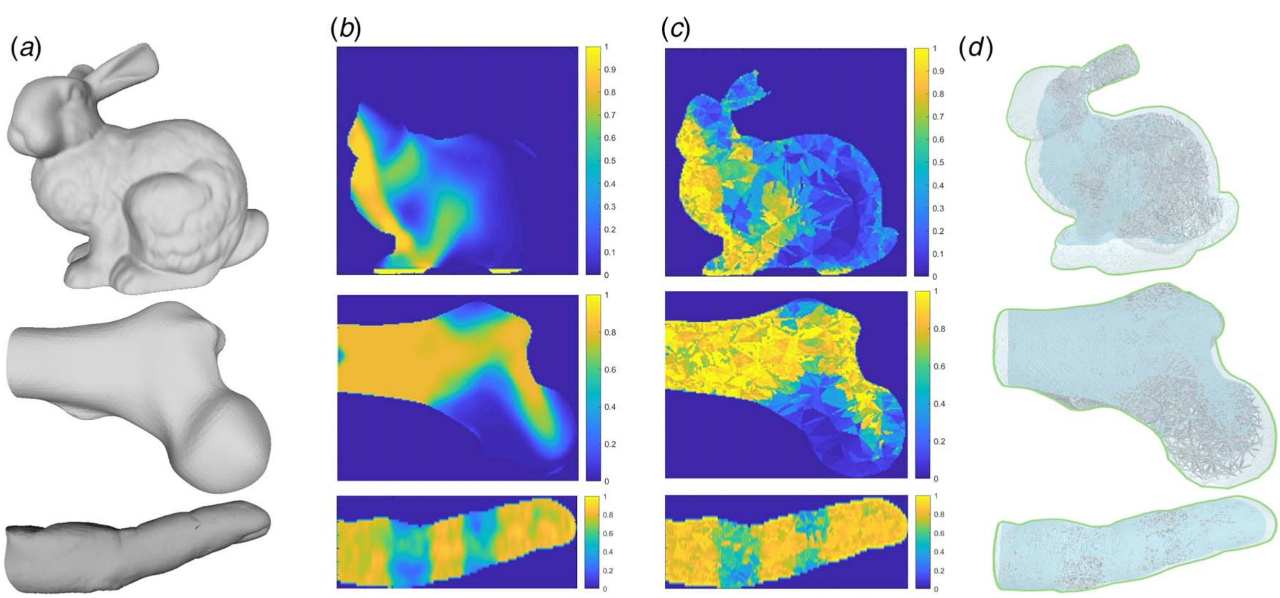

Fig. 12 Three examples to demonstrate the function of our approach in generating lattice structures for density matching: (*a*) the input 3D models, (*b*) the required density, (*c*) the resultant density, and (*d*) the resultant lattice structures. Note that the resultant lattice structures are rendered by directly applying ray-tracing in Persistence of Vision Raytracer (POV-ray) [48] (i.e., no mesh surface is generated).

Fig. 13 Lattice structures as infills for four example models—(from left to right) Teddy, Kitten, Camel, and Horse, where the edges need additional supporting structures in 3D printing are displayed. It is easy to find our results having much less number of additional supporting edges. The length percentage of completely self-supported struts Ψ (Eq. (14)) are reported as well. The values of self-supporting metric Γ (Eq. (13)) are reported in Table 3 for each model using the constant radius for all struts. (*a*) Lattice structures directly generated by Tet—Gen [8]—the values of Y are 30:4% (Teddy), 28:0% (Kitten), 27:7% (Camel) and 29:9% (Horse) and (*b*) lattice structures generated by applying the SO and PO steps for self—supporting optimization in our approach—the values of Y are 47:3% (Teddy), 48:6% (Kitten), 45:5% (Camel) and 49:7% (Horse) respectively.

structures are generated on-site in a streaming manner. Table 2 shows the comparison of the file size for storing the same lattice structures with the MC method, the LSLT method, and our method. Clearly, the proposed method resulted in files with much lighter size for representing lattice structures.

6.2 Density Matching. Having shown the memory-efficiency feature of the proposed approach, we now move to the effectiveness of our approach: generating lattice structures that match the prescribed density distribution.

Three models were tested, and the results are given in Fig. 12.The given density distribution took the form of a voxel set, and each voxel's density value was evaluated by applying the Monte-Carlo integration to the implicit solid models generated by our approach. The corresponding implicit solid models are depicted in Fig. 12(*d*). From Fig. 12(*c*), our method is found to generate lattice structures that match the prescribed density (Fig. 12(*b*)) very well.

Table 3 The statistic for self-supporting optimization by measuring the values of $\Gamma(\Omega)$ in Eq. (13)

			Our self-supporting optimization		
Model	Fig.	Tet-Gen [8]	SO	SO + PO	Time (s)
Teddy	13	59.7	48.8	30.1	171.3
Kitten	13	63.5	47.9	37.0	120.2
Camel	13	63.1	50.0	37.3	89.3
Horse	13	60.7	54.8	33.8	92.5
Cube	6	60.4	47.3	40.3	22.6

Note: All models are evaluated using the same radius for all struts.

6.3 Self-Supporting Optimization. This section shows the effectiveness of the self-supporting optimization module of our approach. We compared the raw results from the *tetrahedral mesh generation* (Tet-Gen) method [8] with those optimized by ours (i.e., with two additional optimization steps: SO and PO).

The comparison results are shown in Fig. 13,where the first row gives the results of Tet-Gen, and the second row is our results. We also report the length percentage of self-supporting edges Ψ (in Fig. 13) and the values of self-supporting metric Γ (in Table 3). Our method achieved a 50.8–73.5% improvement to the Tet-Gen method, as expected. In addition, from the computation time listed in the last column of Table 3, our method is very fast, with all optimizations done within 3 min.

6.4 Slicing and Fabrication. In this section, we demonstrate the efficiency feature of our slicing method (Sec. 4). All the models previously tested have been sliced using this algorithm. Figure 14 shows the resultant binary images of three of them: the bunny model, the bone model, and the finger model. Based on the binary images, we printed out the bunny model to further validate our slicing algorithm, as shown in Fig. 15. A DLP 3D printer was used, and the method presented in Ref. [43] was chosen to generate supports wherever necessary.

A result of the bone model is fabricated by a SLM-based metal 3D printer (as shown in Fig. 16)—the dimensions are 19.2 mm × 12.6 mm × 26.7 mm. After generating the binary image for each slice, the contour of boundary is generated by the method of Huang et al. [45] and the zigzag toolpath is employed to fill the interior region. The model was fabricated in 9.5 h. The printer has a 500 W IPG fiber laser and a 25 μm beam size.

Table 4 gives the memory consumption and the time usage statistics in slicing those models. As our method works in a streaming

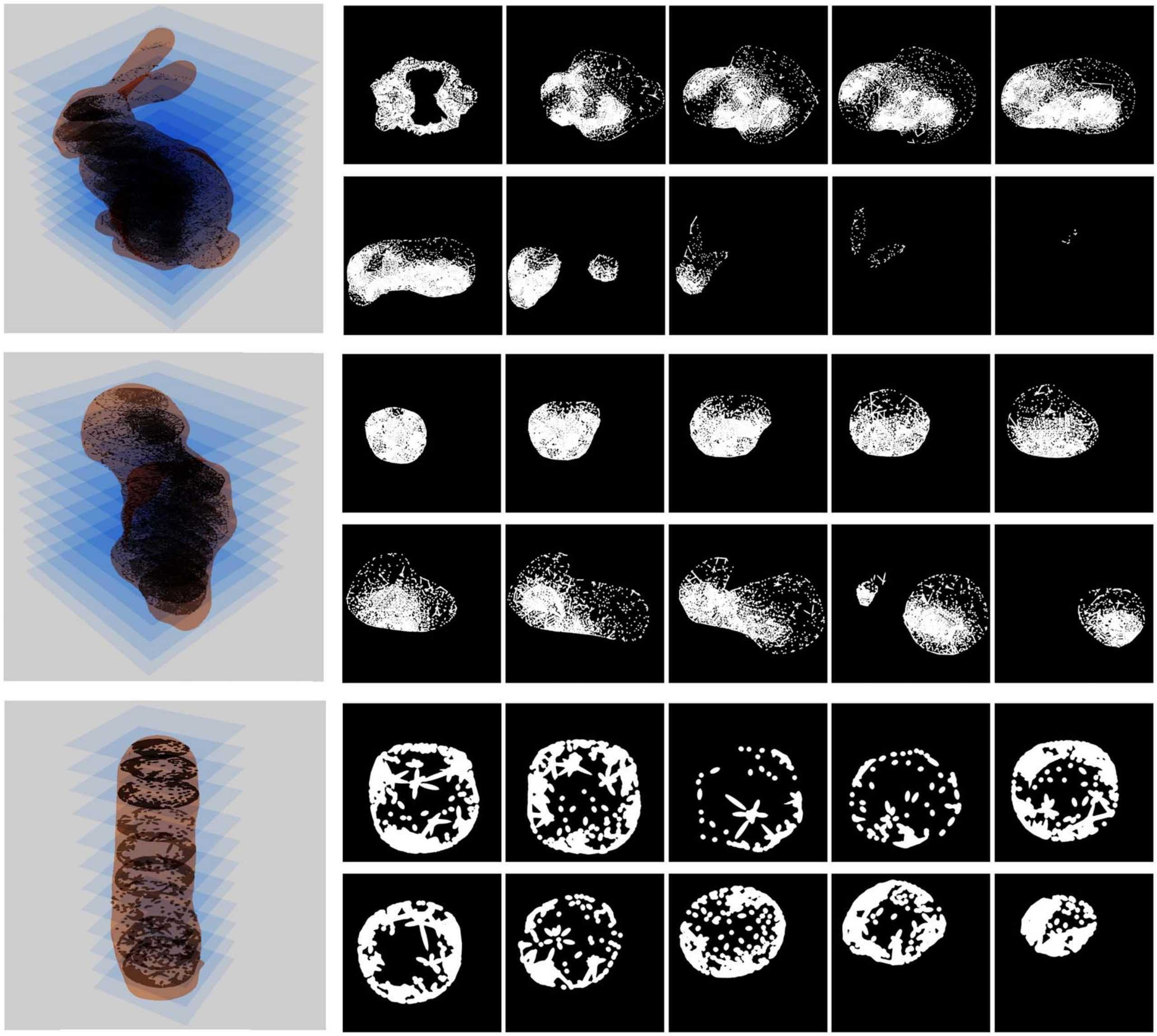

Fig. 14 The binary images obtained by slicing lattice structures (the ones shown in Fig. 12) represented by our method

manner, the consumed memory and time is layer dependent, rather than model dependent. Specifically, the memory and time have a positive correlation with the maximal number of struts intersecting with a specific slicing plane, as confirmed by the statistics in Table 4. For all the tested models, the slicing algorithm is observed to generate correct binary images fast and memory-efficiently.

Our approach is very scalable for models even with a huge number of struts. Although the size of models that can be 3D printed is currently limited by the hardware made available, we demonstrate the method's scalability using a lattice structure with more than 101M struts (i.e., the Kitten-HR model as shown in Fig. 17).When generating binary images for 3D printing, the maximal number of intersected struts is about 1M with the maximal memory usage at 447MB. This fits quite well in a commercially available computer system.

Fig. 15 A bunny model with lattice structure generated by our approach as shown in Fig. 12—the model is fabricated using a Connex Object350 3D printer

Fig. 16 A bone model with lattice structure generated by our approach as shown in Fig. 12—the metal model is fabricated by a SLM 3D printer

Table 4 Statistic for our slicing algorithm

Model (no. of struts)	Maximum number of intersected struts	Used memory (MB)	Resolution of images	Time (s)/slice
Teddy (17,462)	1720	22	1536 × 768	0.10
Kitten (122,925)	38,439	88	1489 × 1368	18.82
Camel (12,241)	2642	43	1456 × 728	1.10
Horse (16,301)	1152	35	1592 × 796	0.09
Finger (359,212)	21,490	97	1808 × 904	15.63
Bone (75,484)	17,303	87	2768 × 1384	14.50
Bunny (123,610)	26,069	146	3648 × 1824	26.47
Kitten-HR (101,514,060)	1,061,866	447	5728 × 2864	347.23

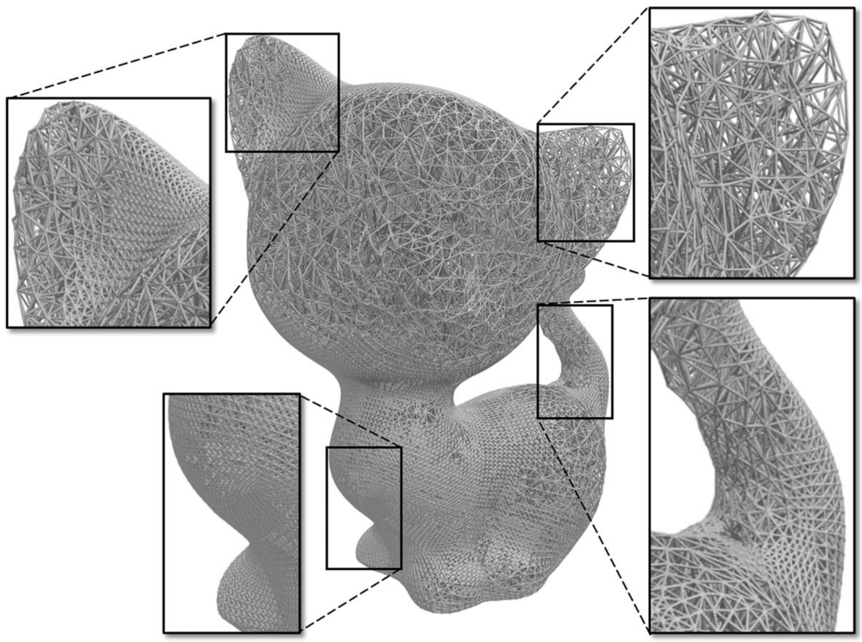

Fig. 17 We are able to model and slice a lattice structure with millions of struts (14,063,027) by the streaming mode. Here, the solid implicit model generated by our method is rendered by direct ray-tracing [48].

7 Conclusions

An implicit modeling technique is presented in this paper for large-scale adaptive lattice structures, which have a lot of applications in additive manufacturing. Starting from the edges of a graph, the solid of an initial lattice structure is defined using convolution surfaces with edges of the graph as skeletons. Different from the methods based on distance field, solids defined by convolution surfaces are highly smooth at the knots with complex topology. This gives better mechanical strength than the solids with creases. Benefit from the local support formulation of convolution surface in our approach, the representation is very memory-efficient as only skeletons need to be stored and the slicing of solids can be efficiently computed as only limited number of skeletons are involved in computing the intersection. This results in a highly scalable approach—lattice structures with more than tens of million struts can be effectively modeled by our method.

The functionality of our approach has been demonstrated in the application to generate an adaptive lattice structure matching the given density distribution. The matched density is achieved by two operations: structural subdivision and strut radius adaptation. The results are quite encouraging, where the desired densities are realized at all places inside a given 3D model. For those regions who need to add supporting structures, the finally realized density on a physically fabricated model could be larger than the desired one when using single-material 3D printing. Although this will not reduce the mechanical strength in the model, we plan to model supporting structures by convolutional surface in an unique representation. As a consequence, the lattice structure fabricated by single-martial 3D printing can also match the desired density precisely.

Moreover, in the future work, we plan to use this method for designing lattice structures with different spatially graded physical properties such as a heat exchanger with large surface area within a small volume, an energy absorber tolerating great deformation at a low stress level, and an acoustic insulator with its large number of internal pores. In all these applications, the convolution surface-based modeling method proposed in our work can show great advantages in its effectiveness and scalability.

Acknowledgment

This research work is partially supported by the Natural Science Foundation of China (NSFC) (Project Ref. Nos. 61628211 and 61572527) and the Hunan Science Fund for Distinguished Young Scholars (Ref. No. 2019JJ20027). The authors would like to acknowledge the private communication with Jun Wu during the development of this project.

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The authors attest that all data for this study are included in the paper.

Appendix 1: Estimate Target Edge-Length of Tetrahedron

Given the edge-length of a regular tetrahedron as L, the tetrahedron's volume is

$$V_{tet} = \frac{\sqrt{2}}{12} L^3 \tag{18}$$

We then calculate the volume of beams inside the tetrahedron, which consists of two parts—the cylindrical regions and the spherical regions (see Fig. 18 for an illustration).

(1) The cylinder volume of an edge with length L and radius r is $L\ r^2\pi$, which has the volume $\sqrt{3}r^3\pi$ overlapped with two spheres with radius r centered at the edge's two endpoints. The dihedral angle of every two faces of tetrahedron is $\text{Arccos}(1/3)$. A tetrahedron has six edges, thus we have the volume of the cylindrical part inside the tetrahedron

$$V_{\text{cylinders}} = 6\,\text{Arccos}\left(\frac{1}{3}\right)(L - 2\sqrt{3}r)r^2$$

(2) The sphere at a vertex has the approximate volume $V_{\text{sphere}} = 4\sqrt{3}\pi r^3$. A tetrahedron has four vertices, and the steradian Ω of each vertex could be calculated as

$$\Omega = \alpha + \beta + \gamma - \pi = 3\,\text{Arccos}\left(\frac{1}{3}\right) - \pi$$

where α, β, and γ are dihedral angles of a vertex. We can have the total volume at all the corners as

$$V_{\text{corners}} = \frac{32}{3}\left(3\,\text{Arccos}\left(\frac{1}{3}\right) - \pi\right)r^3$$

When using material with density τ to realize the target density ρ inside the tetrahedron, we should have

$$\rho V_{tet} = \tau(V_{\text{cylinders}} + V_{\text{corners}})$$

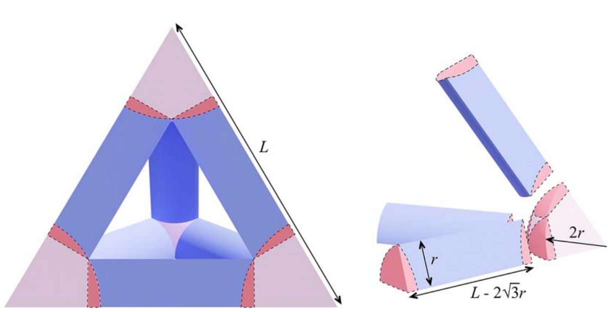

Fig. 18 The volume of the lattice structure inside a regular tetrahedron can be approximately evaluated by the decomposition of six cylinders (with length *L* and radius *r*) and four spheres (with radius 2*r*), where the dotted regions are overlapped so that leads to approximation errors

This leads to a density estimating formula as

$$\rho = 2\sqrt{2}\left(\frac{9\,\text{Arccos}(1/3)r^2}{L^2} - \frac{((96 - 18\sqrt{3})\text{Arccos}(1/3) - 32\pi)r^3}{L^3}\right)\tau$$

Note that the volume of merged struts estimated in the above way has some errors as the volume in overlapped regions of sphere and cylinders are double counted (see the dotted portion shown in Fig. 18). However, as the purpose of this estimation is only to generate a target length for remeshing, this approximation will not influence the final result of density matching. The volume of $\mathcal{S}(\Omega)$ in our density matching framework is computed by Monte-Carlo integral with reference to the implicit solid.

When the target density ρ is given, the formula can be rewritten into a cubic polynomial equation to estimate the target edge-length of the surface mesh

$$L^3 + pL + q = 0 \tag{19}$$

where

$$p = -18\sqrt{2}\,\text{Arccos}\left(\frac{1}{3}\right)\tau r^2/\rho$$

and

$$q = -\left(\left(192\sqrt{2} - 36\sqrt{6}\right)\text{Arccos}\left(\frac{1}{3}\right) - 64\sqrt{2}\pi\right)\tau r^3/\rho$$

According to Cardano formula, it is easy to approve the discriminant for the roots $\Delta = (q/2)^2 + (p/3)^3 < 0$, so this equation has three different real solutions:

$$L_1 = (h_1)^{1/3} + (h_2)^{1/3}$$
$$L_2 = \omega(h_1)^{1/3} + \omega^2(h_2)^{1/3}$$
$$L_3 = \omega^2(h_1)^{1/3} + \omega(h_2)^{1/3}$$

where $h_1 = -(q/2) + \sqrt{(q/2)^2 + (p/3)^3}$, $h_2 = -(q/2) - \sqrt{(q/2)^2 + (p/3)^3}$, and $\omega = (-1 + \sqrt{(3)}i)/2$. Assuming the average edge-length L_{cur} of the current mesh, L_{ini}, the one in $\{L_1, L_2, L_3\}$ being closest to L_{cur} will be selected as a possible estimation of the target edge-length $\bar{L}$. And in order to avoid the vanish of an edge, the edge-length should be not less than $4r$. In short, we can have the following solution for $\bar{L}$

$$\bar{L} = \max\{4r, L_{ini}\}$$

Appendix 2: Ratio of Risky Projected Area

In this appendix, we derive the formula to calculate the ratio of risky projected area on a cylinder that needs additional supporting structure for 3D printing. All the analysis is conducted on a cylinder with unit radius, unit length, and bottom-circle's center located at the origin $\mathbf{o}$. In its initial configuration, the cylinder's axis is aligned with the z-axis. Then, the parametric representation for a point on the bottom-circle is $\mathbf{q}(\phi) = (\cos\phi, \sin\phi, 0)$.

Without loss of generality, any strut with the angle θ between its axis and the printing direction $\mathbf{t}_P$ (see the illustration in Fig. 8) can be considered as rotating around x- and z-axes and scaling the unit cylinder. Only rotating around x-axis will change the ratio of projected area that needs to add supporting structures. Considering the rotation matrix around x-axis as $\mathbf{R}_x(\theta)$, a point on the bottom-circle becomes

$$\mathbf{p}(\phi) = \mathbf{R}_x(\theta)\mathbf{q}(\phi) = (\cos\phi,\ \sin\phi\cos\theta,\ \sin\phi\sin\theta)$$

Therefore, the surface normal of any point on this circle is $\mathbf{n} = \mathbf{p}(\phi) - \mathbf{o} = \mathbf{p}(\phi)$.

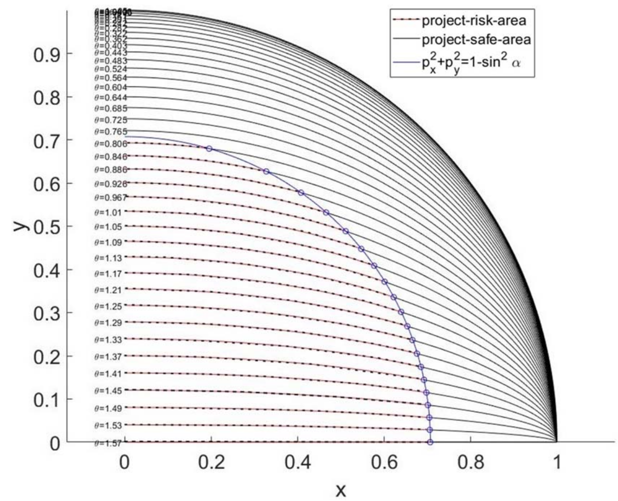

Fig. 19 When $\alpha = \pi/4$, this figure shows the projected ellipse of the bottom-circle of a cylinder with different θ, where the red arcs indicate the risky regions. The ratio of the risky region is evaluated as the arc length ratio of the red region versus the total elliptic arc.

Considering the condition to add support as

$$\mathbf{n} \cdot \mathbf{t}_P < -\sin\alpha$$

we can determine the portion on the circle to add support by determining the range of ϕ that makes

$$\mathbf{n} \cdot \mathbf{t}_P = \mathbf{p}(\phi) \cdot \mathbf{t}_P = \sin\phi \sin\theta < -\sin\alpha$$

Here, we apply $\mathbf{t}_P = (0, 0, 1)$.

Now we project the circle back onto the xy-plane. For any $\theta \in (0, \pi/2)$, projection of the bottom-circle gives an ellipse with width $a = 1$ and height $b = \cos\theta$. The critical point that changes from self-supporting to support-needed can then be determined by solving the elliptic equation and the equation embedding $\mathbf{n} \cdot \mathbf{t}_P = -\sin\alpha$, which can be

$$\mathbf{p}_x^2 + \mathbf{p}_y^2 + \sin^2\alpha = \cos^2\phi + \cos^2\theta \sin^2\phi + \sin^2\theta \sin^2\phi = 1$$

Therefore, we have

$$\begin{cases} \mathbf{p}_x^2 + \dfrac{\mathbf{p}_y^2}{\cos^2\theta} = 1 \\ \mathbf{p}_x^2 + \mathbf{p}_y^2 = 1 - \sin^2\alpha \end{cases}$$

By elimination, we obtain

$$\left(\cos^2\theta - 1\right)\mathbf{p}_x^2 = \cos^2\theta - 1 + \sin^2\alpha$$

$$\left(1 - \frac{1}{\cos^2\theta}\right)\mathbf{p}_y^2 = -\frac{\sin^2\theta}{\cos^2\theta}\mathbf{p}_y^2 = -\sin^2\alpha$$

As a result, the solution of ϕ as ϕ_0 that satisfies the above two equations can be obtained when the values of θ and α are given. In short, we have $\mathbf{p}_y = \sin\alpha\cos\theta/\sin\theta = \sin\phi_0\cos\theta$, which results in the value of ϕ_0 as

$$\phi_0 = \arcsin\left(\frac{\sin\alpha}{\sin\theta}\right)$$

Since $\mathbf{p}_x^2 \geq 0$ and $(\cos^2\theta - 1) < 0$, we should let $\cos^2\theta - 1 + \sin^2\alpha < 0$ to ensure there is a solution for $\mathbf{p}_x$. This leads to $\sin^2\alpha \leq 1 - \cos^2\theta = \sin^2\theta$, which actually requires $\theta > \alpha$ for risky area (i.e., area needs additional support).

The ratio of risky region for the whole projected area can be evaluated as the ratio of arc length in the region $\phi > \phi_0$. As illustrated in Fig. 19, the ratio of the dotted portion curve's length on the whole ellipse is the ratio of risky region. As a consequence, we derive the following general formula for $g(\theta, \alpha)$ as

$$g(\theta, \alpha) = \begin{cases} 0, & (\sin\theta \leq \sin\alpha) \\ \dfrac{\int_0^{\phi_0}(\sin^2\phi + \sin^2\theta\cos^2\phi)^{1/2}d\phi}{\int_0^{\pi/2}(\sin^2\phi + \sin^2\theta\cos^2\phi)^{1/2}d\phi}, & (\sin\theta > \sin\alpha) \end{cases} \quad (20)$$

The corresponding shape of $g(\theta, \alpha)$ is given in Fig. 20.

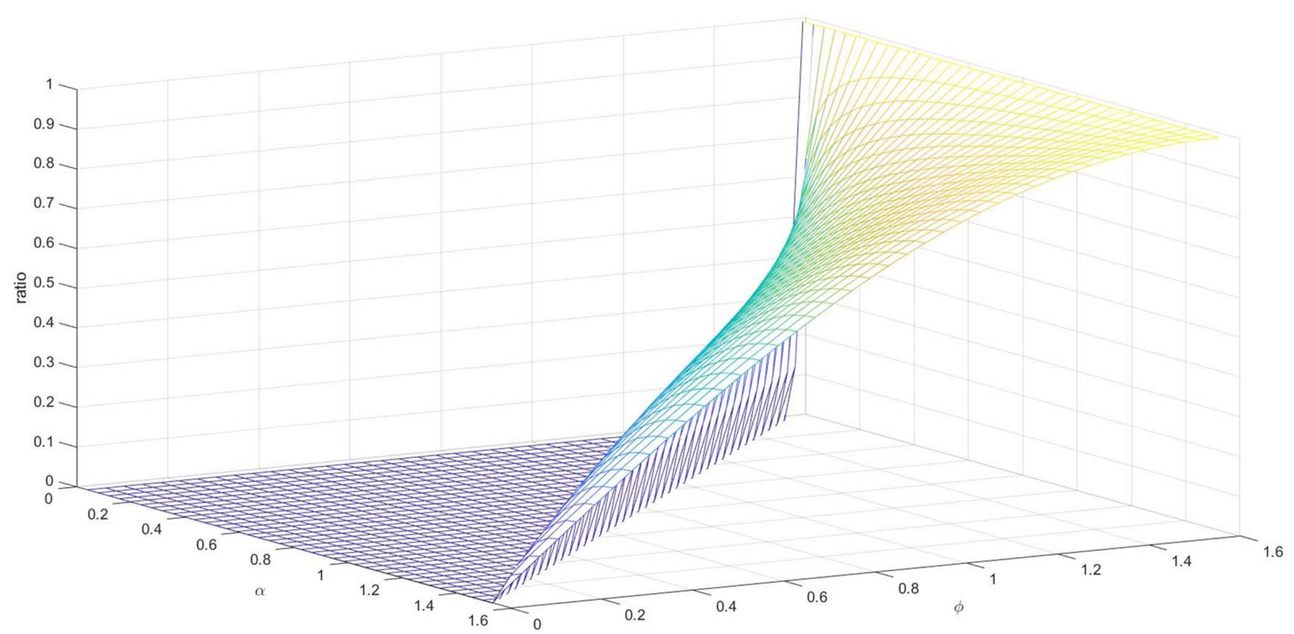

Fig. 20 The 3D shape of $g(\theta, \alpha)$ as a height field of (θ, α)

To ease the computation of $g(\cdots)$ in optimization, we approximate it by polynomials as follows:

$$g(\theta, \alpha) \approx \begin{cases} 0, & (\sin\theta \le \sin\alpha) \\ \Sigma_{i=0}^{5}\Sigma_{j=0}^{5} a_{i,j}\theta^i\alpha^j, & (\sin\theta > \sin\alpha) \end{cases} \quad (21)$$

with

$$a_{i,j} = 10^{-1} \times \begin{bmatrix} -5.15 & 31.71 & -56.03 & 34.12 & -5.45 & 0.43 \\ 26.36 & -127.20 & 164.41 & -67.87 & 4.99 & 0 \\ -38.00 & 134.20 & -106.20 & 25.86 & 0 & 0 \\ 17.79 & -49.65 & 1.64 & 0 & 0 & 0 \\ 0.60 & 6.60 & 0 & 0 & 0 & 0 \\ -1.66 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

References

[1] Martínez, J., Dumas, J., and Lefebvre, S., 2016, "Procedural Voronoi Foams for Additive Manufacturing," ACM Trans. Graph., **35**(4), p. 44.
[2] Martínez, J., Song, H., Dumas, J., and Lefebvre, S., 2017, "Orthotropic K-Nearest Foams for Additive Manufacturing," ACM Trans. Graph., **36**(4), p. 121.
[3] Kuipers, T., Wu, J., and Wang, C. C., 2019, "CrossFill: Foam Structure With Graded Density for Continuous Material Extrusion," Comput. Aided Des., **114**, pp. 37–50.
[4] Qin, Z., Jung, G. S., Kang, M. J., and Buehler, M. J., 2017, "The Mechanics and Design of a Lightweight Three-Dimensional Graphene Assembly," Sci. Adv., **3**(1), p. e1601536.
[5] Rosen, D., Johnston, S., Reed, M., and Wang, H., 2006, "Design of General Lattice Structures for Lightweight and Compliance Applications," Rapid Manufacturing Conference, London, UK, July 5–6.
[6] Chen, Y., and Wang, C. C., 2008, "Layered Depth-Normal Images for Complex Geometries—Part One: Accurate Sampling and Adaptive Modeling," ASME IDETC/CIE 2008 Conference, 28th Computers and Information in Engineering Conference, Brooklyn, NY, Aug. 3–6, pp. 729–739.
[7] Chougrani, L., Pernot, J.-P., Véron, P., and Abed, S., 2017, "Lattice Structure Lightweight Triangulation for Additive Manufacturing," Comput. Aided Des., **90**, pp. 95–104.
[8] Si, H., 2015, "Tetgen, a Delaunay-Based Quality Tetrahedral Mesh Generator," ACM Trans. Math. Soft., **41**(2), p. 11.
[9] Gao, W., Zhang, Y., Ramanujan, D., Ramani, K., Chen, Y., Williams, C. B., Wang, C. C., Shin, Y. C., Zhang, S., and Zavattieri, P. D., 2015, "The Status, Challenges, and Future of Additive Manufacturing in Engineering," Comput. Aided Des., **69**, pp. 65–89.
[10] Livesu, M., Ellero, S., Martínez, J., Lefebvre, S., and Attene, M., 2017, "From 3D Models to 3D Prints: An Overview of the Processing Pipeline," Comput. Graph. Forum, **36**(2), pp. 537–564.
[11] Leung, Y.-S., Kwok, T.-H., Li, X., Yang, Y., Wang, C. C. L., and Chen, Y., 2019, "Challenges and Status on Design and Computation for Emerging Additive Manufacturing Technologies," ASME J. Comput. Inf. Sci. Eng., **19**(2), p. 021013.
[12] Ding, D., Pan, Z. S., Cuiuri, D., and Li, H., 2014, "A Tool-Path Generation Strategy for Wire and Arc Additive Manufacturing," Int. J. Adv. Manuf. Technol., **73**(1–4), pp. 173–183.
[13] Zhao, H., Gu, F., Huang, Q.-X., Garcia, J., Chen, Y., Tu, C., Benes, B., Zhang, H., Cohen-Or, D., and Chen, B., 2016, "Connected Fermat Spirals for Layered Fabrication," ACM Trans. Graph., **35**(4), p. 100.
[14] Steuben, J. C., Iliopoulos, A. P., and Michopoulos, J. G., 2016, "Implicit Slicing for Functionally Tailored Additive Manufacturing," Comput. Aided Des., **77**, pp. 107–119.
[15] Kumar, G. S., Pandithevan, P., and Ambatti, A. R., 2009, "Fractal Raster Tool Paths for Layered Manufacturing of Porous Objects," Virtual Phys. Prototyp., **4**(2), pp. 91–104.
[16] Wu, J., Wang, C. C., Zhang, X., and Westermann, R., 2016, "Self-Supporting Rhombic Infill Structures for Additive Manufacturing," Comput. Aided Des., **80**, pp. 32–42.
[17] Lee, J., and Lee, K., 2017, "Block-Based Inner Support Structure Generation Algorithm for 3d Printing Using Fused Deposition Modeling," Int. J. Adv. Manuf. Technol., **89**(5–8), pp. 2151–2163.
[18] Lu, L., Sharf, A., Zhao, H., Wei, Y., Fan, Q., Chen, X., Savoye, Y., Tu, C., Cohen-Or, D., and Chen, B., 2014, "Build-to-Last: Strength to Weight 3d Printed Objects," ACM Trans. Graph., **33**(4), p. 97.
[19] Lee, M., Fang, Q., Cho, Y., Ryu, J., Liu, L., and Kim, D.-S., 2018, "Support-Free Hollowing for 3d Printing Via Voronoi Diagram of Ellipses," Comput. Aided Des., **101**, pp. 23–36.
[20] Stanković, T., and Shea, K., 2020, "Investigation of a Voronoi Diagram Representation for the Computational Design of Additively Manufactured Discrete Lattice Structures," ASME J. Mech. Des., **142**(11), p. 111704.
[21] Wang, W., Wang, T. Y., Yang, Z., Liu, L., Tong, X., Tong, W., Deng, J., Chen, F., and Liu, X., 2013, "Cost-Effective Printing of 3d Objects With Skin-Frame Structures," ACM Trans. Graph., **32**(6), p. 177.
[22] Zhang, X., Xia, Y., Wang, J., Yang, Z., Tu, C., and Wang, W., 2015, "Medial Axis Tree—An Internal Supporting Structure for 3d Printing," Comput. Aided Geom. Des., **35**, pp. 149–162.
[23] Schumacher, C., Bickel, B., Rys, J., Marschner, S., Daraio, C., and Gross, M., 2015, "Microstructures to Control Elasticity in 3d Printing," ACM Trans. Graph., **34**(4), p. 136.
[24] Panetta, J., Zhou, Q., Malomo, L., Pietroni, N., Cignoni, P., and Zorin, D., 2015, "Elastic Textures for Additive Fabrication," ACM Trans. Graph., **34**(4), p. 135.
[25] Fryazinov, O., Vilbrandt, T., and Pasko, A., 2013, "Multi-Scale Space-Variant Frep Cellular Structures," Comput. Aided Des., **45**(1), pp. 26–34.
[26] Yang, Y., Chai, S., and Fu, X.-M., 2018, "Computing Interior Support-Free Structure Via Hollow-to-Fill Construction," Comput. Graph., **70**, pp. 148–156.
[27] Christiansen, A. N., Schmidt, R., and Bærentzen, J. A., 2015, "Automatic Balancing of 3D Models," Comput. Aided Des., **58**, pp. 236–241.
[28] Stava, O., Vanek, J., Benes, B., Carr, N., Mès, R., Jérémie, and Lefebvre, S., 2012, "Stress Relief: Improving Structural Strength of 3d Printable Objects," ACM Trans. Graph., **31**(4), pp. 48:1–48:11.
[29] Ou, J., Dublon, G., Cheng, C.-Y., Heibeck, F., Willis, K., and Ishii, H., 2016, "Cilllia: 3D Printed Micro-Pillar Structures for Surface Texture, Actuation and Sensing," Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, CA, May.
[30] Vanek, J., Galicia, J. A. G., and Benes, B., 2014, "Clever Support: Efficient Support Structure Generation for Digital Fabrication," Comput. Graph. Forum, **33**(5), pp. 117–125.
[31] Zhang, X., Le, X., Panotopoulou, A., Whiting, E., and Wang, C. C., 2015, "Perceptual Models of Preference in 3D Printing Direction," ACM Trans. Graph., **34**(6), p. 215.
[32] Dumas, J., Hergel, J., and Lefebvre, S., 2014, "Bridging the Gap: Automated Steady Scaffoldings for 3d Printing," ACM Trans. Graph., **33**(4), p. 98.
[33] Hu, K., Jin, S., and Wang, C. C., 2015, "Support Slimming for Single Material Based Additive Manufacturing," Comput. Aided Des., **65**, pp. 1–10.
[34] Wang, C. C., Wang, Y., and Yuen, M. M., 2003, "Feature-Based 3d Non-Manifold Freeform Object Construction," Eng. Comput., **19**, pp. 174–190.
[35] Sherstyuk, A., 1999, "Kernel Functions in Convolution Surfaces: A Comparative Analysis," Vis. Comput., **15**, pp. 171–182.
[36] Hubert, E., and Cani, M.-P., 2012, "Convolution Surfaces Based on Polygonal Curve Skeletons," J. Symb. Comput., **47**(6), pp. 680–699.
[37] Tang, Y., Xiong, Y., Boddeti, G. N., and Rosen, D. W., 2019, "Generation of Lattice Structures With Convolution Surface," Proceedings of CAD'19, Singapore, June 26–29.
[38] Jin, X., and Tai, C.-L., 2002, "Analytical Methods for Polynomial Weighted Convolution Surfaces With Various Kernels," Comput. Graph., **26**(3), pp. 437–447.
[39] Nelaturi, S., and Shapiro, V., 2015, "Representation and Analysis of Additively Manufactured Parts," Comput. Aided Des., **67**, pp. 13–23.
[40] Larsen, E., Gottschalk, S., Lin, M. C., and Manocha, D., 1999, "Fast Proximity Queries With Swept Sphere Volumes," Technical Report 99-018, Department of Computer Science, University of North Carolina at Chapel Hill.
[41] Hearn, D., Baker, M. P., and Manocha, D., 2004, *Computer Graphics with OpenGL*, Pearson Prentice Hall, Upper Saddle River, NJ.
[42] McMains, S. A., and Carlo, H. S., 2000, "Geometric Algorithms and Data Representation for Solid Freeform Fabrication," Ph.D. Dissertation, University of California, Berkeley, CA.
[43] Huang, P., Wang, C. C., and Chen, Y., 2014, "Algorithms for Layered Manufacturing in Image Space," ASME Advances in Computers and Information in Engineering Research, Hong Kong, Sept. 1.
[44] Maple, C., 2003, "Geometric Design and Space Planning Using the Marching Squares and Marching Cube Algorithms," Proceedings of International Conference on Geometric Modeling and Graphics 2003, London, UK, July 16–18.
[45] Huang, P., Wang, C. C. L., and Chen, Y., 2013, "Intersection-Free and Topologically Faithful Slicing of Implicit Solid," ASME J. Comput. Inf. Sci. Eng., **13**(2), p. 021009.
[46] Surazhsky, V., and Gotsman, C., 2003, "Explicit Surface Remeshing," Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, Aachen Germany, June, pp. 20–30.
[47] Lorensen, W. E., and Cline, H. E., 1987, "Marching Cubes: A High Resolution 3d Surface Construction Algorithm," ACM Trans. Graph., **21**(4), pp. 163–169.
[48] Plachetka, T., 1998, "Pov Ray: Persistence of Vision Parallel Raytracer," Proceedings of Spring Conference on Computer Graphics, Budmerice, Slovakia, Apr. 23–25, pp. 123–129.