A Combinatorial Approach for Constructing Lattice Structures

Chaman Singh Verma
Systems Science Laboratory,
Palo Alto Research Center,
Palo Alto, CA 94304
e-mail: cverma@parc.com

Behzad Rankouhi
Department of Mechanical Engineering,
University of Wisconsin-Madison,
Madison, WI 53706
e-mail: rankouhi@wisc.edu

Krishnan Suresh
Department of Mechanical Engineering,
University of Wisconsin-Madison,
Madison, WI 53706
e-mail: ksuresh@wisc.edu

Lattice structures exhibit unique properties including a large surface area and a highly distributed load-path. This makes them very effective in engineering applications where weight reduction, thermal dissipation, and energy absorption are critical. Furthermore, with the advent of additive manufacturing (AM), lattice structures are now easier to fabricate. However, due to inherent surface complexity, their geometric construction can pose significant challenges. A classic strategy for constructing lattice structures exploits analytic surface–surface intersection*; this, however, lacks robustness and scalability. An alternate strategy is voxel mesh-based* isosurface extraction*. While this is robust and scalable, the surface quality is mesh-dependent, and the triangulation will require significant postdecimation. A third strategy relies on explicit geometric stitching where tessellated open cylinders are stitched together through a series of geometric operations. This was demonstrated to be efficient and scalable, requiring no postprocessing. However, it was limited to lattice structures with uniform beam radii. Furthermore, existing algorithms rely on explicit convex-hull construction which is known to be numerically unstable. In this paper, a* combinatorial stitching *strategy is proposed where tessellated open cylinders of arbitrary radii are stitched together using topological operations. The convex hull construction is handled through a simple and robust projection method, avoiding expensive exact-arithmetic calculations and improving the computational efficiency. This is demonstrated through several examples involving millions of triangles. On a typical eight-core desktop, the proposed algorithm can construct approximately up to a million cylinders per second.*
[DOI: 10.1115/1.4044521]

Keywords: computational geometry, computer-aided design, design optimization

1 Introduction

A lattice structure is defined as a collection of unit cells, where a unit cell can have any topology. Typical unit cell topologies include face-centered-cubic (FCC) structure, body-centered-cubic (BCC) structure, boxed-body-centered-cubic (Box-BCC) structure, and so on [1]. For example, Fig. 1(*a*) illustrates a FCCz-type lattice structure. Lattice structures exhibit a large surface area and a highly distributed load-path. This makes them very effective in engineering applications where weight reduction, thermal dissipation, and energy absorption are critical [1]. Furthermore, with the advent of additive manufacturing (AM), lattice structures can now be easily fabricated [2,3]. For example, Fig. 1(*b*) illustrates a lattice structure fabricated via selective laser melting.

Numerous studies have been conducted to understand the mechanical behavior of lattice structures, through both theoretical and experimental studies [2,4,5]. These studies summarize the influence of lattice topology, lattice parameters, and material on the thermal and mechanical behavior of lattice structures. However, researchers have repeatedly raised concerns on efficient constructions of lattice structures [3,6]. Traditional computer-aided design (CAD) methods simply do no scale-up to the needs. Specialized methods are needed to fill the gap. The objective of this paper is to develop a simple, robust, and highly efficient method for constructing lattice structures for downstream applications.

Specifically, it will be assumed that a lattice graph with nodes and edges has been defined; for example, Fig. 2(*a*) illustrates a simple box-type lattice graph with 45 nodes and 96 edges. Furthermore, it will be assumed that each edge has been assigned a radius that defines a cylinder associated with each edge, where the profile of the cylinder will be approximated by an N-sided polygon. Under some mild assumptions about the graph and radii, the objective is to construct a water-tight triangulated model of the lattice structure (see Fig. 2(*b*)).

Furthermore, while lattice structures typically imply a repetitive pattern of a unit cell, the algorithm developed here apply equally to any graph structure, i.e., a collection of nodes and edges. For example, Fig. 3(*a*) illustrates a generic (nonrepeated) graph, and Fig. 3(*b*) illustrates the corresponding structure with finite radius, that we wish to construct efficiently. The ability to construct such generic structures will be particularly advantageous in constructing optimal load-carrying frames, as demonstrated later.

The remainder of the paper is organized as follows. In Sec. 2, popular methods for constructing 3D lattice structures are reviewed, and their deficiencies are identified. In Sec. 3, assumptions underlying the proposed strategy are summarized, followed by a detailed discussion on the proposed strategy and algorithm. In Sec. 4, the algorithm and its implementation are demonstrated through several numerical experiments. Finally, in Sec. 5, potential applications and some of the limitations of the current work are summarized.

2 Literature Review

In this section, the relevant literature is covered by addressing three different strategies that have been proposed for constructing lattice structures.

(1) *Surface–surface intersections:* Commercial CAD systems such as SolidEdge, CATIA, NX, and SOLIDWORKS rely on analytic surfaces to model geometry. Thus, each cylindrical surface can be modeled analytically, and the structure can be constructed through Boolean operations that rely on repeated surface–surface intersections. The resulting model can then be tessellated, depending on the degree of approximation desired. Simple structures such as the one in Fig. 2(*b*) can be easily constructed using such CAD systems. However, this strategy does not scale beyond a few

Contributed by the Design Automation Committee of ASME for publication in the Journal of Mechanical Design. Manuscript received October 16, 2018; final manuscript received July 31, 2019; published online September 4, 2019. Assoc. Editor: James K. Guest.

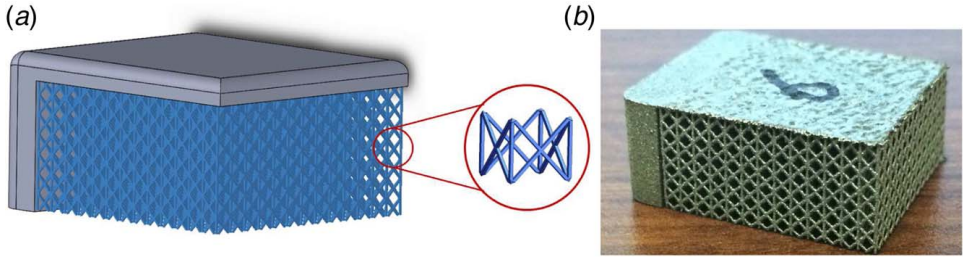

Fig. 1 (*a*) A lattice structure model and (*b*) an AM fabricated lattice structure

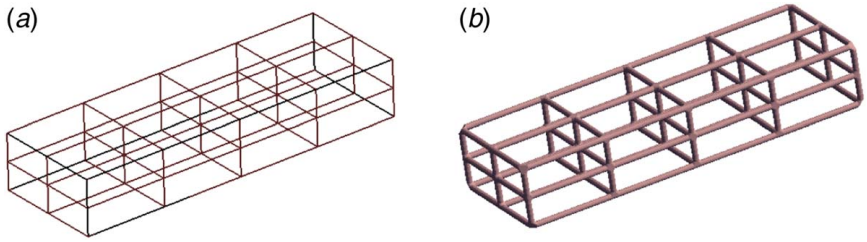

Fig. 2 (*a*) A repeated lattice graph and (*b*) a corresponding lattice structure

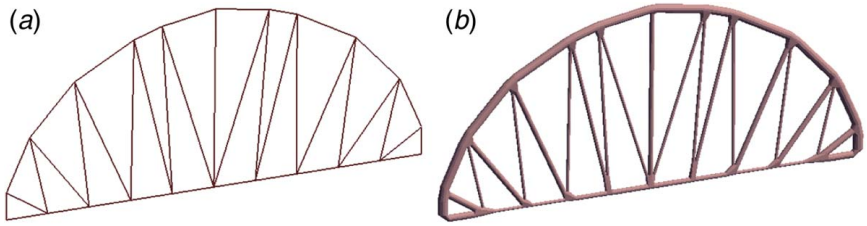

Fig. 3 (*a*) A generic graph and (*b*) a corresponding structure

hundred lattices due to large memory requirements and time complexity [6,7]. An alternate hybrid strategy was proposed in Ref. [6] where, instead of analytic surfaces, tesselated cylinders are directly used for Boolean operations. It was demonstrated that lattice structures with about 5000 beams can be handled with ease. Unfortunately, Boolean operations over tesselations are fragile, require exact arithmetic [8], and are computationally demanding [9].

(2) *Isosurface extraction:* In the isosurface extraction method, a distance function that captures the lattice structure implicitly is defined over an underlying voxel mesh. Then, the marching cubes algorithm is used to extract the zero-value isosurface [10,11]. While this method is both robust and scalable, the surface quality depends on the voxel size. Furthermore, a large number of triangles is usually constructed, requiring significant postdecimation. Alternately, the *morphological dilations* [12] strategy has been proposed, where a virtual sphere is rolled over each edge of the lattice graph, embedded within a voxel mesh. Although this approach is simple and reliable, its accuracy once again depends on the resolution of the underlying voxel mesh, and a large number of triangles is typically generated. Furthermore, this method does not exploit the cylindricity of the beams.

(3) *Stitching methods:* The third strategy uses open tessellated cylinders as a starting point. Here, two approaches have been proposed [13,14]; the two differ in the way they handle cylinder intersections. In the former, intersection points are explicitly computed and then stitched together using geometric operations, assuming that the radii of intersecting cylinders are identical. In the latter, convex hulls at the intersections are constructed, and these convex hulls are then stitched with the open cylinders. However, as is well known, convex hull construction is numerically unstable and typically require exact arithmetic [15]. In addition, the latter does not consider reducing the volume of the convex hull; this is often desirable in AM. In an effort similar to Ref. [14], exoskeleton[1] is a Rhino plug-in for lattice structure construction.

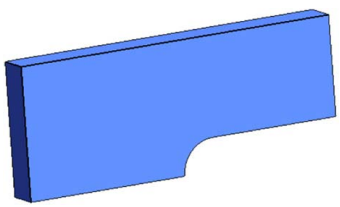

Fig. 4 A J-shaped model

Finally, several commercial implementations are now available for constructing lattice structures; these include Element,[2] Meshify,[3] Frustrum,[4] and NetFabb.[5] Element[6] uses isosurface extraction and stitching methods to construct lattice structures from CAD or mesh geometries. It also allows for creating

[1] https://www.grasshopper3d.com/group/exoskeleton
[2] https://www.ntopology.com/
[3] http://www.adimant.com/meshify.html
[4] https://www.frustum.com/
[5] https://www.autodesk.com/products/netfabb/overview
[6] See Note 2.

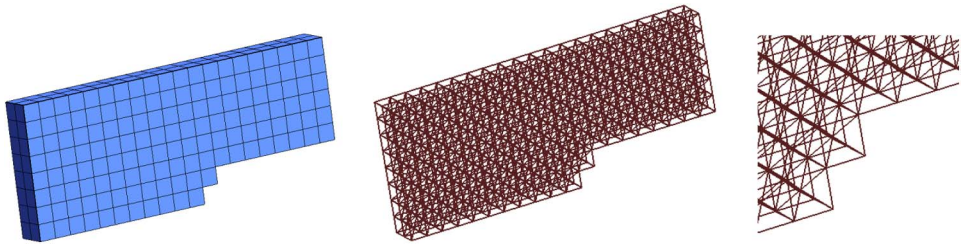

Fig. 5 Creating a lattice graph from a voxel mesh

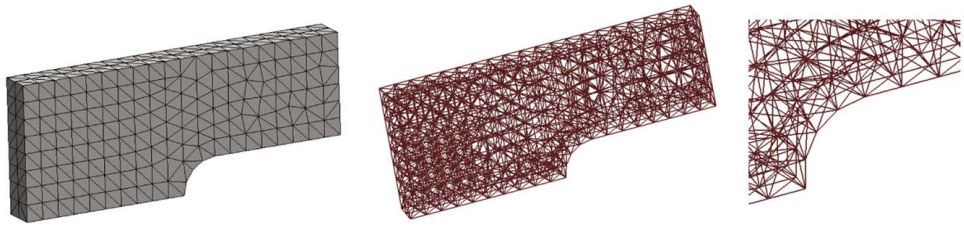

Fig. 6 Creating a lattice graph from a tetrahedral mesh

nonuniform lattice structures based on user defined functions. Meshify[7] has a focus on manufacturability of final designs; it is a cloud-based program that allows users to perform slicing operations on the constructed lattice structures, a necessary step for additive manufacturing. Frustrum[8] uses its geometry kernel to create and blend lattice structures with traditional CAD geometries with a focus on manufacturability and topology optimization. Finally, NetFabb[9] uses lattice generation and optimization as a lightweighting tool for additive manufacturing.

3 Proposed Strategy

This section covers the proposed algorithm, starting with a discussion on assumptions and terminology.

3.1 Assumptions and Terminology. As stated earlier, a fundamental assumption is that a lattice graph has been defined, where the graph consists of nodes and edges, and a radius value has been assigned to each edge. Formally, a lattice graph is defined as $G = G(V, E, R, P)$ where:

- V is the set of nodes and their 3D coordinates; it is assumed that no two nodes are coincident.
- E is the edge connectivity, where each edge is connected to two unique nodes in the graph. There are no restrictions on the number of edges attached to a node. It is assumed that no two edges intersect, except at the nodes.
- R is the user-specified radius for each edge; the radius must be strictly less than half the edge length. David Stasiuk[10] have constructed lattices by specifying nodal radii since this often leads to smoother nodal geometry. However, the choice in this paper is driven by engineering applications where edge radii (rather than nodal radii) are specified through optimization.
- P is the number of segments (≥ 3) by which the circumference of the cylinder associated with each edge is divided. For example, with $P = 4$, the circumference will be approximated by a square, and so on. It will be assumed that P is common for all edges (this is a limitation of the current implementation, not the underlying algorithm).

Under these mild assumptions, a deterministic and simple algorithm is proposed to construct a triangulated model of the lattice structure with the following characteristics:

(1) The model is watertight, i.e., there will be no gaps or cracks in the mesh.
(2) The model is topologically valid, i.e., none of the triangles is inverted or geometrically overlap with other triangles.
(3) The model is consistently oriented and two-manifold.
(4) The number of lattice triangles can be estimated prior to constructing the lattice structure.

3.2 Constructing Lattice Graphs. As stated earlier, a starting point for the proposed algorithm is a lattice graph. For completeness, two methods are proposed for constructing lattice graphs from a 3D geometry, using the example in Fig. 4.

The first involves creating a voxel mesh for a given geometry and creating a lattice graph inside each voxel (see Fig. 5). A drawback with a uniform voxel mesh is that the mesh will not necessarily conform to the parent geometry. Deforming the voxel mesh to conform to the geometry can alleviate this problem [16] but can also create highly distorted lattices; also see Ref. [10] for an alternate strategy for creating conforming lattices from a voxel mesh.

Instead of using a voxel mesh, the edges and vertices of a *tetrahedral* mesh can also be used for creating graphs (see Fig. 6). While it is easier to create a conforming graph via a tetrahedral mesh, the graph will exhibit a much more complex topology, i.e., on the average, more edges will be connected to a node. Note that

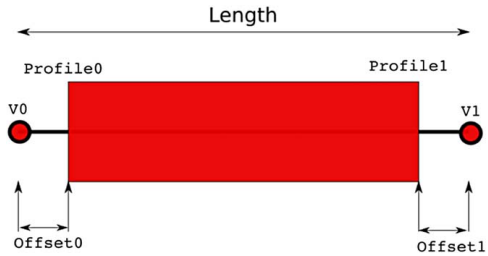

Fig. 7 Computing the profile offset

[7]See Note 3.
[8]See Note 4.
[9]See Note 5.
[10]See Note 1.

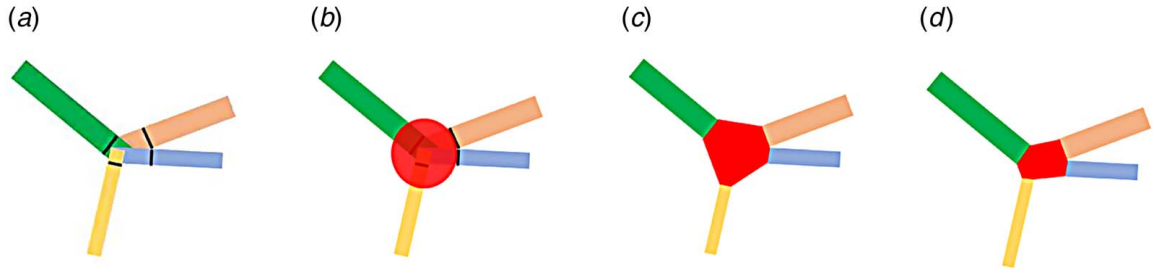

Fig. 8 Four simple steps for constructing lattice structures: (*a*) compute profile offsets, (*b*) compute node radius, (*c*) build convex hull, and (*d*) reduce node volume

the resulting structure no longer fits the standard definition of a lattice structure that typically consists of repeated unit cells.

3.3 Algorithm. Once a lattice graph has been defined, a key concept in constructing the lattice structure is that of *profiles* associated with each lattice edge (Fig. 7). A profile is the cross-section of the cylinder and is offset from the two end points by a specific distance computed below. At any given node, the profiles of all cylinders associated with a given node are stitched together, and the resulting nodal-volume is shrunk to a minimum. These steps are illustrated in Fig. 8 and described below.

(1) *Pair-Wise Minimum Profile Offset:* Let E_i and E_j be two edges coincident at a node, with a subtended angle of θ_{ij} (see Fig. 9). Let the radii associated with the two edges be r_i and r_j, respectively. The minimum profile offset d_{ij} for the cylinder on edge E_i with respect to the cylinder on E_j to avoid intersection is given by (see proof in Fig. 9) $d_{ij} = (r_j + r_i \cos(\theta_{ij}))/\sin(\theta_{ij})$.

(2) *Computing Node Radius:* At each node, the node radius is defined as the maximum profile offsets for all pairs of edges E_i and E_j (associated with that node) computed in step 1. At this value, none of the cylinders will locally intersect. Using this radius, profiles are constructed for each cylinder. The profile is governed by the number of user-defined segments P. To construct the profile, P vertices are first generated on a unit circle lying on an XY plane, and then mapped using standard quaternions. Furthermore, to ensure that there is no twisting of vertices located at profile_0 and profile_1 (see Fig. 7), the same quaternion is used for both profiles. An open cylinder is then constructed by connecting the two profiles as in Fig. 10(*a*). If there are P segments on a profile, then the number of triangles on the open cylinder is $2P$. If the edge has no adjacent neighbor, then it is closed at that end as in Fig. 10(*b*); $P-2$ triangles are needed to close the end.

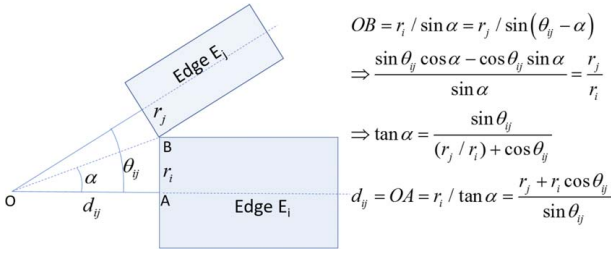

Fig. 9 Minimum profile offset to avoid interference with neighboring cylinders

(3) *Constructing Convex Hulls:* Observe that since all profile vertices, associated with a given node, are at the same distance, they form a convex hull. The total number of vertices at a given node is $N_e * P$, where N_e is the number of intersecting edges and P is the number of circle divisions. Since this is typically small (100–200), any off-the-shelf 3D convex hull algorithm [15,17] can be used. However, some of these algorithms require exact arithmetic calculations. To avoid this, here, a modified convex hull algorithm is proposed that relies on standard (and significantly cheaper) Institute of Electrical and Electronics Engineers floating point calculations. First, the profile vertices are projected onto a sphere of radius one; this normalization helps in avoiding overflow and underflow, and in avoiding edge recovery methods described in Ref. [18]. Furthermore, special care is taken to avoid collinear points by (1) rotating the quaternion associated with each edge by a random angle about its axis and (2) perturbing any remaining collinear points. A typical convex hull and connecting cylinders are illustrated in Figs. 11(*a*) and 11(*b*).

(4) *Volume Reduction:* Once the convex hull is computed, the next task is to reduce the volume of the convex hull. When the angle between edges is uniform, the convex hull at the node is tight (see Fig. 12(*a*)), i.e., further reduction in nodal volume is not possible. However, for varying angles, the size of the convex hull can be quite large (see Fig. 12(*b*)).

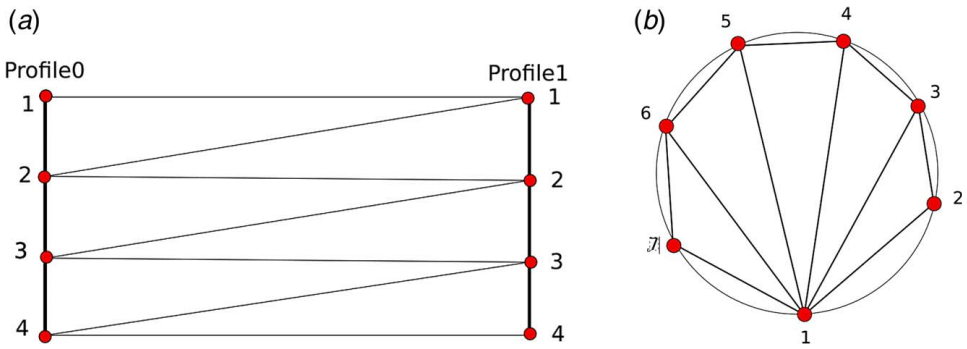

Fig. 10 (*a*) Triangulating the cylindrical surface and (*b*) optionally the closed end of a cylinder

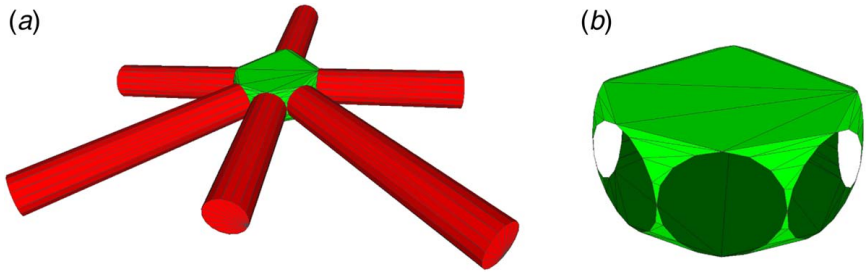

Fig. 11 (*a*) Convex hull at a node and (*b*) holes to attach cylinders

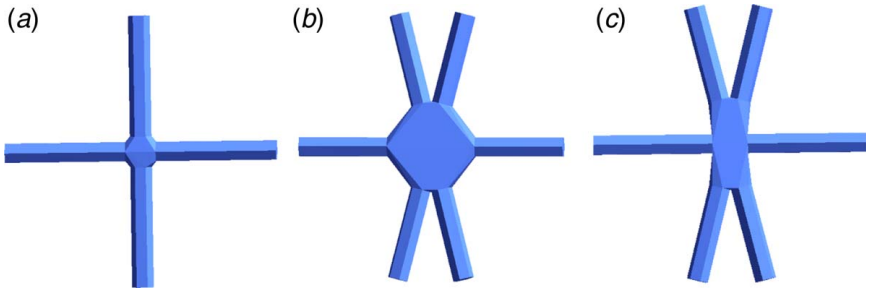

Fig. 12 (*a*) Equal angles lead to small convex hulls, while (*b*) small angles can create artificially large convex hulls, but (c) the nodal volume is reduced iteratively

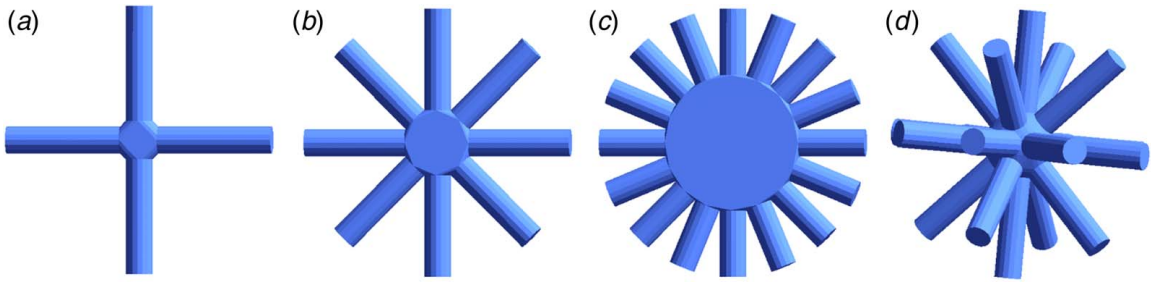

Fig. 13 Nodal connectivity with (*a*) 4 beams, (*b*) 8 beams, (*c*) 16 beams, and (*d*) 18 beams (3D)

This will increase the total volume of the structure and can be detrimental to optimized light-weight lattice structures. Therefore, the last step in this algorithm is to reduce the volume at each node. Specifically, the profile offset distances are reduced to the ideal distance for each edge as derived earlier in Fig. 9; the process is illustrated in Fig. 12(*c*).

The pseudocode of all algorithms is included in the Appendix.

3.4 Estimates on Model Size. An implicit advantage of the proposed algorithm is that the size of the model (number of triangles and vertices) can be estimated a priori. Specifically, the number of vertices (N_V) is determined as follows. Since there are two profiles for each edge (on either end) and each profile has P vertices, the number of vertices in the lattice structure is given by

$$N_V = 2N_E P \tag{1}$$

where N_E is the number of edges in the graph and P is the number of circle divisions.

An estimate of the number of triangles (N_T) can be computed as follows. Associated with each cylinder are triangles, where each triangle has one of the profile segments as an edge; this leads $2P$ triangles per cylinder, i.e., a total of $2N_E P$ cylinder-triangles. Furthermore, each of the profile segments also serves as a base for a triangle on the convex hull. This leads to an additional $2N_E P$ convex-hull-triangles. If the lattice graph has any end nodes (nodes connected to only one beam), two less triangles are needed to close the end, i.e., we subtract $2N_V$ triangles, where N_V is the number of end-nodes, leading to an estimate of

$$N_T = \underbrace{2N_E P}_{\text{Cylinders}} + \underbrace{2N_E P}_{\text{Convex hulls}} - \underbrace{2N_V}_{\text{End caps}} \tag{2}$$

This is only an estimate since additional triangles, shared by multiple beams, may be included in the convex hull.

Table 1 Lattice structure data for increasing degree of connectivity

NumBeams	Expected NumTriangles	Actual NumTriangles	Volume (mm^3)
4	248	252	0.119
8	496	508	0.224
16	992	1020	0.404
18	1116	1148	0.486

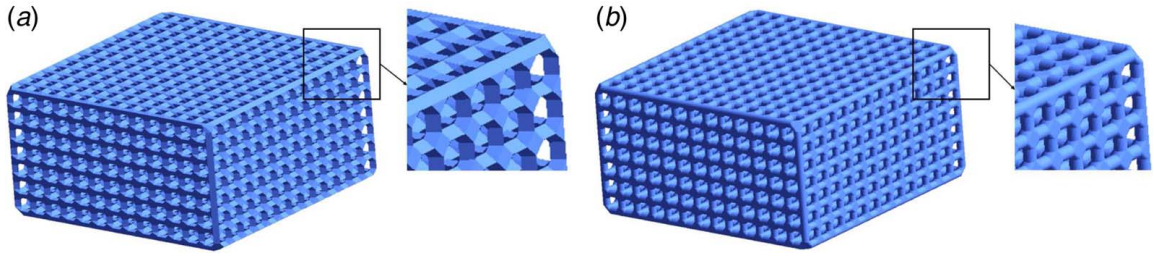

Fig. 14 Lattice structure with varying profile approximation: (*a*) $P = 3$ and (*b*) $P = 24$

4 Numerical Experiments

In this section, the robustness and performance of the algorithm is investigated through several numerical experiments. Unless otherwise stated, all experiments were conducted on a Windows i7-5790X@3GHz (eight physical cores), equipped with 32GB memory. Furthermore, since the construction of the open cylinders and convex hulls can be easily parallelized, OpenMP™ was exploited in the implementation.

4.1 Algorithm Characteristics. The first set of experiments illustrate the basic characteristics and robustness of the algorithm.

4.1.1 Nodal Connectivity. The robustness of the algorithm in handling nodes with increasing degree of connectivity is illustrated in Fig. 13; each beam is of length 1 mm, the radius is 0.1 mm, and $P = 16$. As one expect, with increasing degree of connectivity, the node radius also increases.

Table 1 summarizes the expected number of triangles, the actual number of triangles, and the lattice volume for increasing degree of connectivity.

4.1.2 Profile Approximation. The number of circular divisions for each cylinder is controlled via the parameter P. To illustrate, box lattice structures were constructed inside a cuboid of dimensions 20 mm × 20 mm × 10 mm, with 1.5 mm voxel resolution and 0.25 mm radius, but with varying values of P; two such structures are illustrated in Fig. 14.

Table 2 summarizes the number of triangles, the lattice volume, and computational time for increasing values of P. Furthermore, note in Table 2 that the volume converges only gradually due to the poor approximation of a circle by an N-sided polygon; this can be easily corrected by appropriately increasing the size of the polygon for various values of P (but not implemented intentionally). The table suggests that the computational time grows (approximately) linearly with P. This was further investigated and confirmed by generating lattices with larger values of P as illustrated in Fig. 15.

4.1.3 Nonuniform Radius. Next, we compare lattices with both uniform and nonuniform radii cylinders. A honeycomb lattice graph with 5120 nodes and 7680 edges was constructed over a sphere of radius 1 m. Figure 16(*a*) illustrates the corresponding lattice structure with uniform cylinder radius of 6 mm and $P = 6$; the structure consists of 194,560 triangles. On the other hand, Fig. 16(*b*) illustrates the structure where the radii was randomly distributed between 0.6 mm and 11.4 mm, with $P = 6$ constructed; the structure consists of 194,565 triangles (difference can be attributed to convex hull triangulation). Both models were determined to be water-tight.

Table 2 Lattice structure data for increasing values of *P*

P	Triangles	Volume (mm^3)	Time (s)
3	62,272	598.1	0.010
4	79,408	772.8	0.014
6	113,686	948.3	0.016
10	182,260	1037.3	0.026
18	319,397	1071.3	0.055
24	422,297	1082.3	0.080

4.2 Computational Performance. Consider the Stanford bunny model[11] in Fig. 17(*a*) that was also used in Ref. [13] for lattice structure construction. Here, a BCC lattice structure is used as a template with $P = 6$ (circle divisions) and $R = 0.25$ mm (cylinder radius). With these parameters, lattice structures were constructed for various resolutions of the voxel mesh. Figures 17(*b*) and 17(*c*) illustrate the lattice structures for 7 mm and 3.5 mm resolution, respectively.

The performance of the algorithm is summarized in Table 3; the computational times reported for the proposed method use 8-core OpenMP™ shared memory implementation (single-core performance is discussed later); the computational time is computed as the time taken to convert the graph into lattice structure and does not include voxel mesh construction, graph construction, etc. Table 3 also summarizes the results published in Ref. [13] and for the commercial code Element.[12] In Table 3, LSLT is the geometric stitching method, CDT is the constrained Delaunay triangulation (relies on Boolean operations), and MCM is the marching cube method. The computational times for these algorithms were not reported in Ref. [13]. Element offers several options for creating lattice structures; two of them were selected for comparison: Hex (corresponds to $P = 6$) and *Round* (corresponds to the isosurface method). Element was installed and executed by the authors on the aforementioned Windows machine (same as the proposed algorithm) for direct comparison. As one can observe, for the 2 mm and 1.75 mm resolution, Element–Hex did not terminate after 5 min of

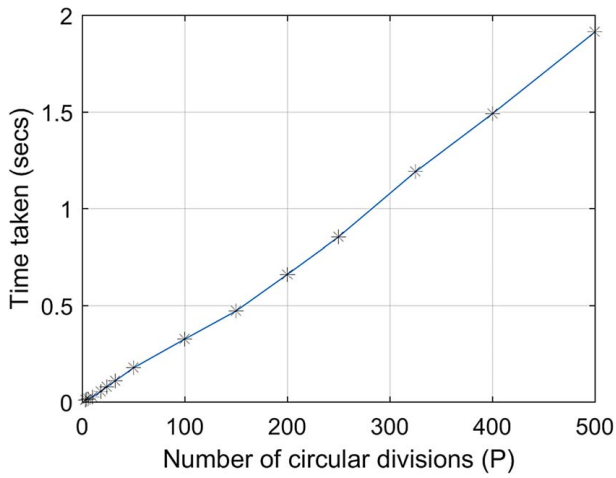

Fig. 15 Computational time as a function of circular divisions

[11] http://graphics.stanford.edu/data/3Dscanrep/
[12] See Note 2.

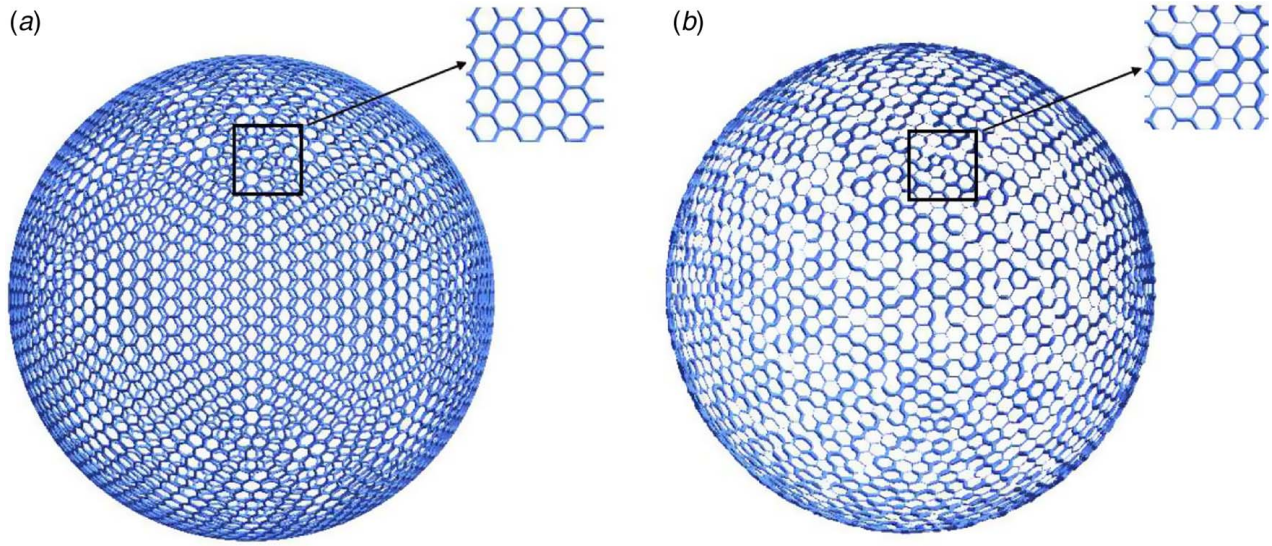

Fig. 16 Lattice structures over a sphere: (*a*) uniform radii and (*b*) nonuniform radii

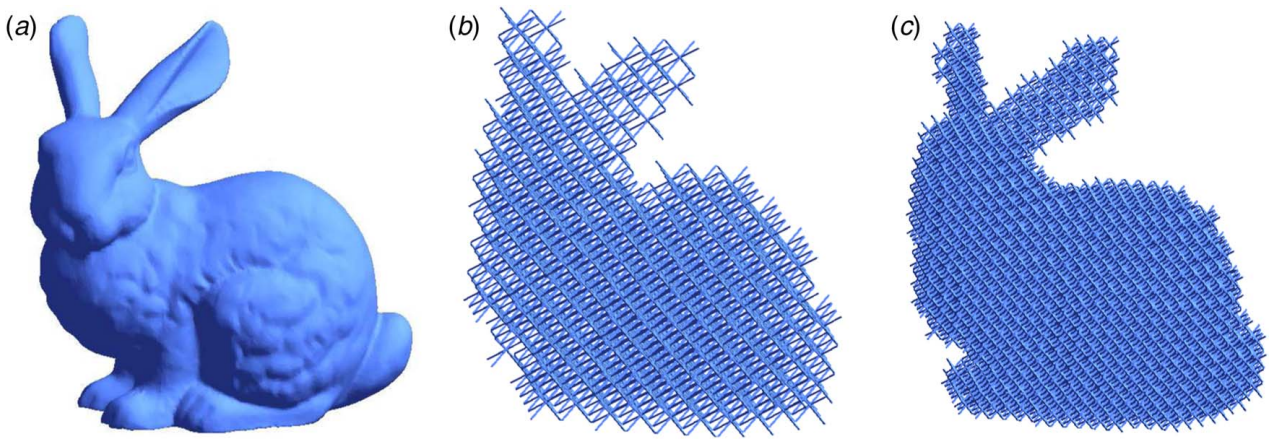

Fig. 17 (*a*) The Stanford bunny, (*b*) lattice structure with 7 mm resolution, and (*c*) lattice structure with 3.5 mm resolution

Table 3 Model complexity and computational time for the Stanford bunny model

Voxel resolution	Lattice graph edges	Lattice structure triangles	Time (s)
7 mm (proposed)	3496	93 K	0.019
7 mm (MCM) [13]	3648	94 K	No data
7 mm (Element–Hex)[a]	3040	81.4 K	4.0
7 mm (CDT) [13]	3648	230 K	No data
7 mm (MCM) [13]	3648	924 K	No data
7 mm (element-round)[a]	3040	6.5 M	20.0
3.5 mm (proposed)	29.2 K	785 K	0.127
3.5 mm (Element–Hex)[a]	23.9 K	643 K	50.0
2 mm (proposed)	141 K	2.66 M	0.66
2 mm (Element–Hex)[a]	127.8 K	Did not terminate	No data
1.75 mm (proposed)	208 K	3.94 M	0.866
1.75 mm (Element–Hex)[a]	190.7 K	Did not terminate	No data
1.00 mm (proposed)	1.08 M	29.2 M	4.45

[a]https://www.ntopology.com/

Table 4 Impact of parallelization on the computational time for the Stanford bunny model

Voxel resolution (mm)	One-core (s)	Two-core (s)	Four-core (s)	Eight-core (s)
2	3.38	1.86	1.13	0.66
1.00	25.45	13.37	8.2	4.45

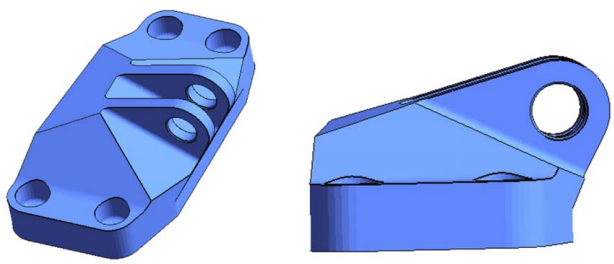

Fig. 18 The GE-GrabCAD model (https://grabcad.com/challenges/ge-jet-engine-bracket-challenge)

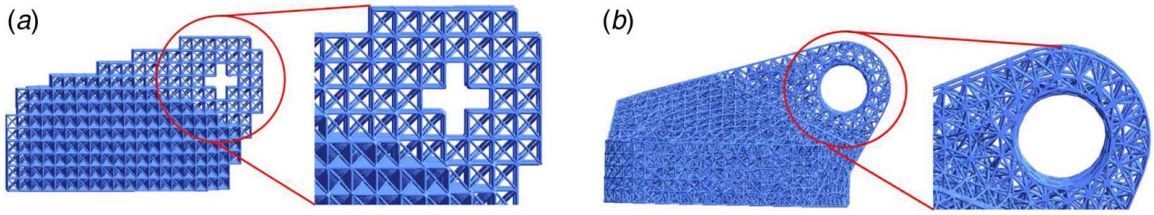

Fig. 19 Lattice structure constructed via (*a*) a voxel mesh and (*b*) a tetrahedral mesh

Table 5 Model complexity and computational time as a function of voxel resolution for the GE-GrabCAD model

	Graph nodes	Graph edges	Lattice triangles	Time (s)
5.5 mm voxel (proposed)	7145	35.4 K	964 K	0.172
5 mm tetmesh (proposed)	5853	35.2 K	979 K	0.185

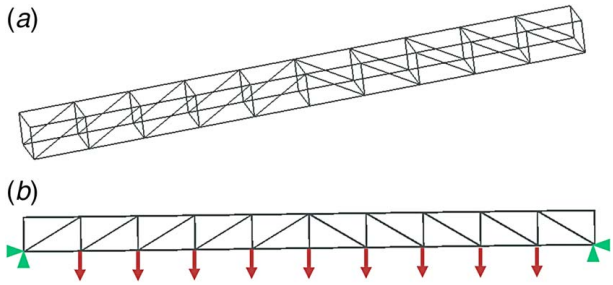

Fig. 20 (*a*) An initial truss structure (graph) and (*b*) loads and restraints

computation; the underlying reasons could not be determined. The main observations are

- The proposed algorithm constructs approximately the same number of triangles per edge as LSLT [13] and Element–Hex.[13] The differences in the number of graph edges for identical voxel resolution are likely due to the different voxelization algorithms used.
- However, it is significantly faster; the current implementation, on an eight-core machine, can construct approximately *four million triangles per second*.
- The MCM and element-round algorithms generate a significantly large number of triangles as expected.

Next, we study the impact of parallelization using the above Stanford bunny model; Table 4 summarizes the performance of the proposed algorithm for two voxel resolutions and four OpenMP™ settings. As one can observe, the algorithm exhibits a high degree of parallelization that can be easily exploited.

4.3 Computer-Aided Design Model Approximation. In this example, the algorithm's capability in handling lattice graphs with uniform and nonuniform nodal connectivity is explored. For illustrative purposes, consider the General Electric-GrabCAD model[14] in Fig. 18.

Two different lattice graphs were constructed for this model. The first uses a Box-BCC unit cell within each voxel of 5.5 mm resolution. The second uses a tetrahedral mesh, constructed via SOLIDWORKS, at 5 mm resolution, where each edge of the tetrahedral mesh serves as a lattice graph edge. (Slightly different resolutions were used to ensure almost identical number of graph edges.) In both cases, $P = 6$ (circle divisions) and $R = 0.25$ mm (cylinder radius). Table 5 summarizes the lattice structure complexity and computational costs. As one can observe, the average number of edges per node is significantly higher for the tetrahedral mesh, implying a more complex topology, which in turn implies larger convex hull volumes.

The results are illustrated in Fig. 19; as expected, for almost identical complexity, the tetrahedral lattice structure better conforms to the geometry.

4.4 Constructing Optimal 3D Truss Structures. An important application of the proposed algorithm is in the construction of 3D optimal frame and truss structures [19,20]. Specifically, the algorithm can be deployed to reconstruct 3D structures once an optimal configuration (topology, geometry, and radii) of the structure has been computed.

As an example, consider the initial configuration of a truss structure illustrated in Fig. 20(*a*) with the loading and restraints illustrated in Fig. 20(*b*) (the specific value of the load is not critical for this illustrative example). We will assume that an initial (uniform) radius has been prescribed for all edges.

We consider now the optimization of the truss. Specifically, the location of the free nodes and the radii of all beams were optimized using the open source algorithm,[15] with a constraint that the total volume of the structure must remain a constant. The resulting optimal truss (graph) is illustrated in Fig. 21(*a*), while Fig. 21(*b*)

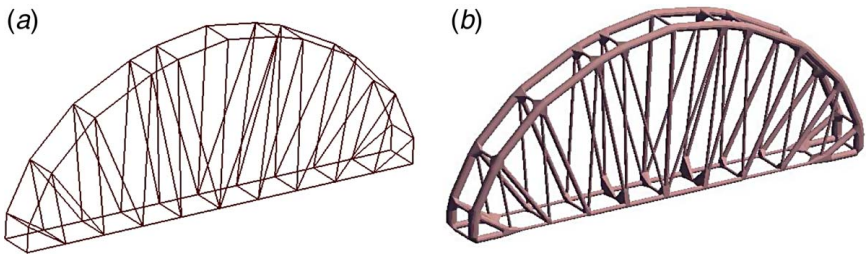

Fig. 21 (*a*) An optimal frame (graph) and (*b*) a corresponding 3D structure

[13]See Note 2.

[14]https://grabcad.com/challenges/ge-jet-engine-bracket-challenge
[15]https://github.com/amandaghassaei/trussoptimization2d

illustrates the 3D structure constructed using the algorithm proposed in this paper.

Rapid construction of such 3D truss structures can be particularly beneficial during the design phase.

5 Conclusions

The main contribution of this paper is an efficient and robust algorithm for constructing lattice structures of arbitrary topology and variable radii. Such complex lattice structures can provide new opportunities for generative design. The numerical experiments demonstrated the generality and efficacy of the algorithm.

There are several topics that are being considered for future research. For example, the current algorithm cannot handle gyroids [2] and other noncylindrical beams [21]; a possible extension to the proposed algorithm is the stitching of generalized shell-like structures. A second limitation is that there are no provisions currently to retain critical surfaces while creating lattice structures (several commercial implementations provide this option). As an extension to lattice construction, 3D mesh generation [22] of these structures is an important topic for future research. Finally, manufacturability constraints, specifically AM constraints such as overhangs and feature size, are currently being considered. These are particularly important if lattices are used as lightweight support structures.

Acknowledgment

This work was partially funded by the National Science Foundation (NSF) through grant numbers NSF-1561899 and NSF-1715970, and through Technical Data Analytics (TDA) SBIR contract N181-094. Prof. Suresh is a consulting Chief Scientific Officer of SciArt, Corp, which has licensed the Pareto technology [23,24], developed in Prof. Suresh's lab, through Wisconsin Alumni Research Foundation. This paper uses the meshing capabilities of the Pareto software.

Appendix: Pseudocode

In this appendix, we provide the pseudocode of all algorithms underlying the proposed method. Algorithm 1 is the primary algorithm that generates the 3D lattice structure. Observe that Algorithms 2, 3, 5, and 6 are easily parallelizable in that the computation at a node is independent of similar computations at any other node. For example, in Algorithm 2, the triangles associated with the convex hull at each node can be computed independently (and then later merged).

Algorithm 1 Build lattice structure

```
Input: A graph G(E, V), with radius defined for each edge, and the
       number of cylinder segments P
Output: Triangulated mesh M of lattice structure
M = empty
foreach v ∈ V do
    M += buildConvexHull(v) /* Algorithm-2 */
end
M += buildSideTriangles() /* Algorithm-7 */
M += buildCapTriangles() /* Algorithm-8 */
```

Algorithm 2 creates a convex hull at each node, with holes on the surface to attach cylinders (see Figs. 11(*a*) and 11(*b*)).

Algorithm 2 buildConvexHull(v_p)

```
Input: A node v_p of the graph
Output: Triangulated convex hull with holes where connecting cylinders
intersect hull
edges = getNumAdjacentEdges(v_p)
if edges.size() > 1 then
    computeConvexHullRadius(v_p) /* Algorithm 3 */
    for i ← 0 to edges.size() do
        generateProfiles(edges[i], v_p) /* Algorithm 4
    end
    computeConvexHull(v_p) /* Algorithm 5 */
    shrinkHull(v_p) /* Algorithm 6 */
end
```

Algorithm 3 computes the size of the convex hull (required in Algorithm 2) at a given node.

Algorithm 3 computeConvexHullRadius(v_p)

```
Input: A node v_p of the graph
Output: Radius of convext hull at the node v_p
eneighs = getAdjacentEdges(v_p)
maxdist = 0.0
foreach iedge ∈ eneighs do
    ri = E[iedge].radius
    maxdij = 0.0
    foreach jedge ∈ eneighs do
        if iedge ≠ jedge then
            t = getAngle(E[iedge], E[jedge])
            rj = E[jedge].radius
            if fabs(t) ≤ 0.5π then
                dij = (ri + rj*cos(t))/sin(t)
                maxdij = max(maxdij, dij)
            end
        end
    end
    maxdist = max(maxdist,maxdij)
end
radius(v_p) = maxdist
```

Algorithm 4 generates the profile of a cylinder (required in Algorithm 2) associated with an edge at a given node.

Algorithm 4 generateProfile(e, v_p, P)

```
Input: P, the number of segments, an edge e, and one of its node v_p
Output: A profile of the cylinder
dt = 2π/P
for i ← 0 to P do
    nodes[i] = {r*cos(i*dt), r*sin(i*dt), 0.0}
end
rotateAlong(nodes, dir(e)) /* Align points in the direction of edge*/
translate(nodes, v_p + dir(e)*radius(v_p)) /* Reposition profile */
```

Algorithm 5 generates the convex hull (required in Algorithm 2) at a given node.

Algorithm 5 computeConvexHull (v_p)

```
Input: A node v_p of the graph and associated edges
Outptut: Convex hull with holes
eneighs = getAdjacentEdges (v_p)
nodes = empty
foreach iedge ∈ eneighs do
    ep = profileNodes(E[iedge], v_p)
    group[ep] = iedge      /*All nodes on a profiles are in same group*/
    nodes += ep;
end
scale(nodes)      /*Project nodes to a unit sphere*/
triangles = convexHull(nodes)
foreach tri ∈ triangles do
    if isSameGroup(tri) then
        tri.active = 0
    end
end
```

Algorithm 5 Continued

```
15 missingNodes = nodes - nodes(triangles)
16 foreach v ∈ missingNodes do
17 |   tp = closestDistance(triangles)
18 |   triangles += refineTriangle(tp, v)
19 end
20 scale(triangles, radius(v_p))   /*Reposition to desired radius*/
```

Algorithm 6 shrinks the convex hull to minimize volume without inverting a triangle.

Algorithm 6 shrinkHull(v_p)

```
   Input: A node of the graph G
   Output: Triangle mesh moved toward the node to reduce the hull volume
1  for i ← 0 to profiles(v_p)) do
2  |   shiftProfile(profile[i].nodes)
3  |   if invertedTriangles(v_p) then
4  |   |   break()
5  |   end
6  |   updateCoords(profile[i].nodes)
7  end
```

Algorithm 7 builds the cylinders connecting to the convex hull.

Algorithm 7 BuildSideTriangles(edge)

```
   Input: A beam E of the graph and P
   Output: Triangles on the side of a beam
1  triangles = empty
2  for i ← 0 to P do
3  |   n0 = i; n1 = (i + 1)%P
4  |   n2 = n0 + nSides; n3 = n1 + P
5  |   triangles += {n0, n2, n1}
6  |   triangles += {n1, n0, n3}
7  end
```

Algorithm 8 closes the ends of cylinders for edges that terminate at a node.

Algorithm 8 BuildCapTriangles(edge)

```
   Input: An edge which has no neighbor and P
   Output: Triangles closing the profile of the edge
1  triangles = empty
2  for i ← 0 to P do
3  |   triangles += {0, (k + 1) %P, (k + 2)%P}
4  end
```

References

[1] Gibson, L. J., and Ashby, M. F., 1999, *Cellular Solids: Structure and Properties*, Cambridge University Press, Cambridge.
[2] Beyer, C., and Figueroa, D., 2016, "Design and Analysis of Lattice Structures for Additive Manufacturing," ASME J. Manuf. Sci. Eng., **138**(12), p. 121014.
[3] Tao, W., and Leu, M. C., 2016, "Design of Lattice Structure for Additive Manufacturing," International Symposium on Flexible Automation (ISFA), Columbus, OH, Aug. 1–3, IEEE, pp. 325–332.
[4] Aremu, A., Maskery, I., Tuck, C., Ashcroft, I., Wildman, R., and Hague, R., 2014, "A Comparative Finite Element Study of Cubic Unit Cells for Selective Laser Melting," The Twenty-Fifth Annual International Solid Freeform Fabrication (SFF) Symposium—An Additive Manufacturing Conference, University of Texas in Austin, Aug., pp. 4–6.
[5] Yan, C., Hao, L., Hussein, A., and Raymont, D., 2012, "Evaluations of Cellular Lattice Structures Manufactured Using Selective Laser Melting," Int. J. Mach. Tools Manuf., **62**(Nov.), pp. 32–38.
[6] Wang, H., Chen, Y., and Rosen, D. W., 2005, "A Hybrid Geometric Modeling Method for Large Scale Conformal Cellular Structures," ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Long Beach, CA, Sept. 24–28, American Society of Mechanical Engineers, pp. 421–427.
[7] Bourell, D. L., Rosen, D. W., and Leu, M. C., 2014, "The Roadmap for Additive Manufacturing and Its Impact," 3D Print. Addit. Manuf., **1**(1), pp. 6–9.
[8] Shewchuk, J. R., 1997, "Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates," Discrete Comput. Geom., **18**(3), pp. 305–363.
[9] Biermann, H., Kristjansson, D., and Zorin, D., 2001, "Approximate Boolean Operations on Free-Form Solids," Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, New York, Jan., ACM, pp. 185–194.
[10] Aremu, A., Brennan-Craddock, J., Panesar, A., Ashcroft, I., Hague, R., Wildman, R., and Tuck, C., 2017, "A Voxel-Based Method of Constructing and Skinning Conformal and Functionally Graded Lattice Structures Suitable for Additive Manufacturing," Addit. Manuf., **13**(Jan.), pp. 1–13.
[11] Panesar, A., Abdi, M., Hickman, D., and Ashcroft, I., 2018, "Strategies for Functionally Graded Lattice Structures Derived Using Topology Optimisation for Additive Manufacturing," Addit. Manuf., **19**(Jan.), pp. 81–94.
[12] Meyer, F., 1992, "Mathematical Morphology: From Two Dimensions to Three Dimensions," J. Microsc., **165**(1), pp. 5–28.
[13] Chougrani, L., Pernot, J.-P., Veron, P., and Abed, S., 2017, "Lattice Structure Lightweight Triangulation for Additive Manufacturing," Comput. Aided Des., **90**(Sept.), pp. 95–104.
[14] Srinivasan, V., Mandal, E., and Akleman, E., 2005, "Solidifying Wireframes," Proceedings of the 2004 Bridges Conference on Mathematical Connections in Art, Music, and Science, Alberta, CA.
[15] De Berg, M., Van Kreveld, M., Overmars, M., and Schwarzkopf, O., 1997, *Computational Geometry*, Springer, New York, pp. 1–17.
[16] Robbins, J., Owen, S., Clark, B., and Voth, T., 2016, "An Efficient and Scalable Approach for Generating Topologically Optimized Cellular Structures for Additive Manufacturing," Addit. Manuf., **12(B)**(July), pp. 296–304.
[17] Preparata, F. P., and Shamos, M. I., 2012, *Computational Geometry: An Introduction*, Springer Science & Business Media, New York.
[18] Si, H., 2015, "Tetgen, a Delaunay-Based Quality Tetrahedral Mesh Generator," ACM Trans. Math. Softw., **41**(2), pp. 11:1–11:36.
[19] He, L., and Gilbert, M., 2015, "Rationalization of Trusses Generated Via Layout Optimization," Struct. Multidiscipl. Optim., **52**(4), pp. 677–694.
[20] Smith, C. J., Gilbert, M., Todd, I., and Derguti, F., 2016, "Application of Layout Optimization to the Design of Additively Manufactured Metallic Components," Struct. Multidiscipl. Optim., **54**(5), pp. 1297–1313.
[21] Gupta, A., Allen, G., and Rossignac, J., 2018, "Quador: Quadric-of-Revolution Beams for Lattices," Comput. Aided Des., **102**(Sept.), pp. 160–170.
[22] Xiong, G., Musuvathy, S., and Fang, T., 2013, "Automated Structured All-Quadrilateral and Hexahedral Meshing of Tubular Surfaces," Proceedings of the 21st International Meshing Roundtable, San Jose, CA, Oct. 7–10, Springer, pp. 103–120.
[23] Suresh, K., 2013, "Efficient Generation of Large-Scale Pareto-Optimal Topologies," Struct. Multidiscipl. Optim., **47**(1), pp. 49–61.
[24] Suresh, K., 2010, "A 199-Line Matlab Code for Pareto-Optimal Tracing in Topology Optimization," Struct. Multidiscipl. Optim., **42**(5), pp. 665–679.