# Augmented Sphere Tracing for Real-time Editing Mega-scale Periodic Shell-lattice Structures

Jiajie Guo, Ming Li*

*State Key Laboratory of CAD & CG, Zhejiang University, China*

**Abstract**

We propose an augmented sphere tracing (AST) pipeline that seamlessly integrates editing, rendering, and slicing of mega-scale periodic shell-lattice structures. Traditional STL-based pipelines face challenges such as time-consuming format conversions, high storage requirements, and complex blending issues between discrete lattice and shell components, often resulting in a loss of geometric accuracy. Alternatively, implicit-based pipelines excel at smooth modeling and robust Boolean operations but require inefficient and error-prone conversions of STL shells into implicit forms, complicating the rendering process. To address these issues, AST combines hybrid implicit lattice and mesh shell representations, eliminating the need for explicit 3D model construction and unnecessary geometric format conversions. It overcomes the major challenges of hybrid forms and mega-scale rendering by using an augmented tracing distance query that avoids costly signed distance field (SDF) calculations while preserving geometric details. Additionally, it employs a local tracing distance query within a single cell, leveraging lattice periodicity for efficiency. The pipeline also supports various types of shell-lattices in industrial applications, including blending, warping, field-directed distributions, region-specific cell types, and produces arbitrary directional slicing for manufacturing. As demonstrated by various examples implemented in WebGPU, AST archives high efficiency and accuracy in real-time rendering of shell-lattices with billions of beams on an RTX 3090, outperforming traditional pipelines in storage, frame time, and detail preservation.

*Keywords:* Lattice structures, Real-time rendering, Hybrid explicit-implicit, Mega-scale, Additive manufacturing.

## 1. Introduction

Lattice structures are lightweight configurations composed of interconnected short beams that converge at common vertices, often with smooth transitions [1, 2, 3, 4]. These structures are commonly used as infill in additive manufacturing, creating specialized *shell-lattice* forms. According to a recent DARPA report, modeling such shell-lattices presents a significant challenge for conventional CAD (Computer-Aided Design) systems due to their mega-scale, which can include up to $10^{12}$ entities and demand up to 100 TB of memory [5].

The traditional approach for designing shell-lattices relies on an STL-based pipeline, where lattice structures represented in continuous implicit or parametric forms are converted into discrete triangular meshes [6, 7, 8]. Boolean operations between the STL-format lattice and the outer shell produce the final shell-lattices. However, the STL-based approach faces three key challenges: First, converting from implicit or parametric formats to STL meshes is highly time-consuming and requires substantial storage, particularly for large-scale lattices [9, 10]. Second, achieving smooth blending between the discrete STL-formatted interior lattices and the outer shell is complex [9, 10]. Third, this process can result in a loss of accuracy, which is critical for high-precision fabrication.

Alternatively, the implicit-centered pipeline shows promise for designing shell-lattice structures, offering advantages in smooth modeling, robust Boolean operations, and seamless blending [11, 12]. However, converting STL shells into implicit forms can degrade geometric accuracy by losing fine details such as sharp edges [13]. While ray-tracing techniques like sphere tracing are effective for implicit surfaces, they are less efficient than direct mesh rendering.

To address these challenges, we propose that the ideal solution for shell-lattice design should (1) support both STL and implicit inputs without tedious conversions and (2) employ a direct approach that bypasses intermediate modeling, producing outputs ready for rendering or fabrication. This is crucial for real-time rendering during interactive editing and high-precision slicing in additive manufacturing.

We introduce an augmented sphere tracing (AST) pipeline that integrates implicit lattice inputs and STL shells for real-time rendering and editing, along with direct slicing of mega-scale shell-lattices, without high-cost intermediate conversions. The pipeline bypasses the explicit generation of 3D models even during Boolean operations.

The key contributions of this study are:

- We introduce a seamless pipeline that integrates editing, rendering, and slicing of mega-scale shell-lattice structures without geometry conversions, efficiently handling both implicit and STL formats.
- We present a novel method for real-time editing of hybrid shell-lattices, unifying mesh, and implicit representations to avoid costly SDF queries while maintaining geometric detail and precision in Boolean operations.
- Leveraging lattice periodicity, our method uses only current cell data for closest distance queries, reducing computational overhead compared to traditional rendering approaches [14].
- The pipeline supports various industrial applications, including region-specific cell types, field-directed distribu-

*Corresponding author
*Email address:* liming@cad.zju.edu.cn (Ming Li*)

tions, beam-to-beam and beam-to-shell blending, and directional slicing.

- Implemented in WebGPU, AST demonstrates superior performance on an RTX 3090, outperforming traditional STL-based approaches in storage, frame time, and rendering detail preservation.

The paper is organized as follows: Section 2 covers related work. Section 3 outlines the problem and overall approach. The core technical details of AST are presented in Section 4 and 5, followed by extensions like smooth blending, variations, and slicing in Section 6. Results and evaluations are discussed in Section 7, and conclusions are drawn in Section 8.

## 2. Related work

### *2.1. Lattice modeling*

Lattice structures, widely used in CAD and solid modeling, have garnered significant research attention due to their industrial applications. Great efforts have been devoted to their modeling [11, 15, 16], simulation [17, 18, 19] or optimization [2, 3]; few has on its rendering [14].

Lattice structures can be classified based on various criteria. They can be open, closed, or mixed according to cell openness [2]. They can be periodic, semi-regular, or irregular according to cell similarity and distributions [4, 7, 20]. Shell-lattice structures have also become a prominent subject of research due to their versatile applications in industry [18, 21, 22, 23].

Lattice structures can be represented in discrete voxels, triangular meshes, continuous parametric, or implicit forms, each with its advantages. The form of discrete voxels is widely used in topology optimization due to their prominent merits in direct simulations [3]. The parametric form represents the lattices in boundary representation (B-rep) [7], which is a de facto standard in conventional industries. The discrete triangular form, particularly in STL form, is the most popular representation for modeling the lattices and their outer shells. It is, however, also heavily blamed for its huge storage and inefficiency [24]. The implicit form offers smooth, robust modeling with fewer geometric constraints, making it favorable for lattice applications [12, 25, 26].

Existing approaches to mega-scale lattice design struggle with various challenges [5, 27, 28]. Early studies explore regular lattices using intersections of infinite slabs and thick planes, and wave functions are used for more complex units [11]. While this approach accelerates queries through space mapping, it struggles with limitations in geometry, symmetry constraints, discontinuities, and artifacts. Parametric program-based representations of steady lattices [7] and a series of related works [29, 30, 31] are introduced. However, their reliance on meshing or voxelization restricts their scalability in ultra-scale rendering or slicing applications. A progressive ray-shooting method for specific lattice patterns is further proposed [32], but it requires multiple frames to achieve accurate results. Additionally, these methods do not adequately support mesh shells and variations, such as field-directed distributions.

Visualizing implicit micro-structures within a mesh-shell remains a relatively under-explored research area. Converting the shell to an implicit representation is a common practice [11]. For SDF representations, the conversion involves two approaches: presampling a level set or computing exact SDF values at query time. The former struggles to balance sampling resolution and detail preservation, while the latter, despite recent acceleration methods [33], causes significant rendering delays.

Some approaches, alternatively, consider rendering without format conversions. The interval shading via volume rendering can support implicit structures embedded in a mesh but is limited to tetrahedrons [34]. The dexel buffer and rasterization techniques in [35] provide valuable insights. Nevertheless, the microstructure encoded by the 0-1 indicator requires binary ray marching for rendering, which is inefficient [14] and takes a few seconds to produce a complete image. Similar approaches are also used in shell mapping to project volume textures onto a mesh shell [36, 37]. Such visualization can also be achieved with the multi-scale shape-material modeling [38, 14]. They consider the mesh shell as a scale and render it using sphere tracing, aligning with porous structures at finer scales. However, the rendering method is not specifically optimized for periodic or mesh structures accordingly.

Targeting the above-mentioned problems, we study the mega-scale lattice design from the aspect of editing, rendering, and slicing, ensuring high efficiency and accuracy by avoiding unreliable model conversions.

### *2.2. Implicit model rendering*

Rendering implicitly defined structures typically involves converting them into triangular meshes using algorithms like Marching Cubes [24, 39, 40, 29] or a voxel grid with a readily available containment indicator [25, 41]. While compatible with commercial software, these approaches often result in detail loss, high storage needs, and computational overhead [29, 9].

Sphere tracing, another popular approach [11], is a completely different rendering pipeline for surfaces implicitly defined by SDF [42]. Most approaches work directly using existing ray tracing tools, e.g., POV Ray [43], and seldom address the challenge of real-time frame rate for mega-scale lattice structures.

Sphere tracing is an iterative forward process where each step is accompanied by solving for the SDF value of primitives at the current query point. Approaches to its acceleration are broadly based on two ideas: (1) reducing the number of iterations [44, 45] or (2) accelerating the evaluation of SDF in one iteration. More works focus on the latter, often exploiting the localization of the ray or primitives. For instance, some studies employ widely-used spatial tree acceleration structures like bounding volume hierarchies (BVH), KD-Tree, etc., to exclude a subset of primitives [46, 47, 48]. Keeter's pruning of directed acyclic graphs composed of Boolean expressions, coupled with subdivision to exploit the image space coherence, is a representative idea, though not based on sphere tracing [49]. Furthermore, Zanni proposed to bound primitives with blending in space according to the regions they affect and to prune the blob-tree composed of primitives, with spatial subdivision taken into account [50].

The sphere tracing acceleration techniques discussed above are designed for general implicit structures without specific optimizations for periodic lattices. Leveraging both acceleration strategies, our method is tailored to the characteristics of periodic lattices and hybrid representations.

## 3. Problem statement and approach overview

### *3.1. Problem statement*

In this study, our shell-lattice is constructed from an infinitely *periodic lattice* structure $L(C_l)$ and a covering cutting *shell* $M$. The lattice is induced from a *hexahedral* background grid, each grid cell embedding a lattice element, under control parameters $C_l$ (Table 1). A lattice element is made of a set of beams $\mathbf{B} = \{\mathbf{B}[i], i = 0, \cdots, n_b - 1\}$, each of which has two end nodes $\mathbf{N}[i, 0]$

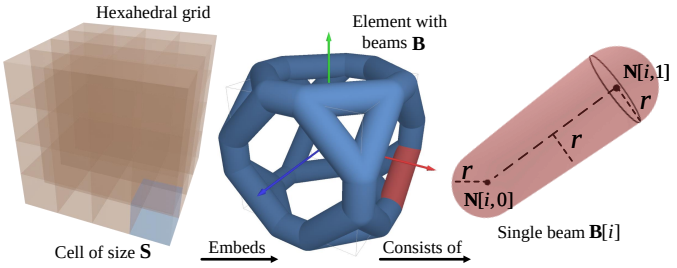



Figure 1: An example illustrating how the internal lattice is defined.

and $\mathbf{N}[i, 1]$, and a radius $r$, see Figure 1 for an illustration. The end-node can be anywhere inside the element. Without loss of generality, we require that for the input unit element, $\mathbf{N}[i, j] \in [-1, 1]$. For ease of explanation, we first assume that each cell has a fixed cell size $\mathbf{S}$, and the radius $r$ is constant across the lattice.

The extensions detailed in Section 6 further extend the scope of lattices that can be processed: $\mathbf{S}$ and $r$ can vary spatially under the control of continuous fields; the lattices can be warped [11, 29]; different cells can have distinct cell types (specified by $\mathbf{B}$); blends can be imposed between lattices and shells.

The SDF (signed distance function) value of a beam $\mathbf{B}[i]$ of radius $r$ at a given point $\mathbf{P}$ is to be used, and defined as,

$$\boldsymbol{F}_{b,i}(\mathbf{P}) = \boldsymbol{F}_{l,i}(\mathbf{P}) - r, \tag{1}$$

where $\boldsymbol{F}_{l,i}(\mathbf{P})$ is the distance from $\mathbf{P}$ to the line segment defined by $\mathbf{N}[i, 0]$ and $\mathbf{N}[i, 1]$.

The outer shell is usually induced from a watertight mesh $M$ of ignorable thickness. In certain cases, only parts of $M$ are filled with lattices, we then define $M$ as $M = M_f \cup M_s,\ M_f \cap M_s = \emptyset$, where $M_f$ is the volume to be filled with lattices while $M_s$ is the left solid part.

All together, the partly filled shell-lattice structure $\Omega(C_l, M_f, M_s)$ is defined,

$$\Omega(C_l, M_f, M_s) = L(C_l) \cap M_f \cup M_s. \tag{2}$$

We aim to construct a seamless pipeline for real-time rendering, editing, and efficient slicing for the shell-lattices defined above at a mega scale. By *real-time editing*, we mean that whenever the parameters $C_l$ in Table 1 are modified, the corresponding $\Omega(C_l, M_f, M_s)$ will be rendered on the next frame, maintaining a frame rate of over 30 frames per second (FPS) for most cases. By *mega scale*, we mean $\Omega(C_l, M_f, M_s)$ can encompass billions of, or even more, beams. Extensions for wider applications are also provided in Section 6, including blending, warping, field-directed properties, and region-specific cell types.

| Notations | Definitions |
|---|---|
| $\mathbf{B}$ | The beam list defined in the unit cell |
| $\mathbf{S}$ | The cell size represented as a triplet |
| $r$ | The radius of beams |
| $p$ | The smoothness term discussed in 6.1 |
| $A, \omega, \phi, \mathbf{A}, \mathbf{\Omega}$, and $\mathbf{\Phi}$ | Parameters of the fields discussed in 6.2.1 |
| $\alpha$ | Parameters of the warping discussed in 6.2.2 |

Table 1: Notations and meanings of $C_l$, where $\mathbf{B}$, $\mathbf{S}$, $r$ are core parameters. All of these parameters can be edited interactively in our AST framework.

### 3.2. Preliminaries

*Sphere tracing.* Sphere tracing is an iterative ray tracing technique used to render implicit surfaces, typically a signed distance function (SDF) [42]. At each iteration, the ray advances from the current point by the SDF value at this point. The iterative process terminates when the SDF value is less than a very small threshold. A key advantage of sphere tracing is its robustness, preventing ray overshoot and avoiding common issues such as surface acne in traditional methods like binary ray marching.

*Bounding volume hierarchy (BVH).* BVH is a spatial acceleration structure represented as a binary tree [51]. In a BVH, leaf nodes tightly encompass primitives, while non-leaf nodes tightly enclose two child nodes, with each node being a bounding box. A common type of bounding box is the axis-aligned bounding box (AABB) [52], as used in AST, defined by the minimum and maximum coordinates of its children. BVH is widely applied in tasks such as intersection tests, collision detection, and closest point queries due to its efficiency.

### 3.3. Approach Overview

An appropriate approach must address the critical challenges on the mega-scale and hybrid implicit-mesh form. We achieve this by the AST (augmented sphere tracing) pipeline (Figure 2), which comprises three modules: mega-scale handling for large lattices, heterogeneous rendering for hybrid implicit-mesh forms, and extensions for broader applications. During each iteration of sphere tracing, the former two modules generate the *corrected distance* for lattices and the *tracing distance* for shells, respectively, which are then used for Boolean operations.

The mega-scale module renders lattices with billions of beams and arbitrary cell types. It uses a user-friendly unit cell definition rule and two algorithms: *beam augmentation* to preserve beam visuals in a cell and *distance calibration* to ensure safe distance queries within the lattice. In this way, distance queries can be done only in a single cell, regardless of any neighbors.

The heterogeneous module renders Boolean results between implicit and mesh forms without format conversion. It is achieved by replacing the true SDF values of the meshes with aggressive but safe tracing distances derived from BVH-based ray-mesh intersections. The pruning strategy for BVH traversal is also provided.

Furthermore, AST offers extensions for design and manufacturing, including internal blending (confined to the unit cell) and internal-external blending (using a low-resolution level set for visuals). Field-directed distributions, warped lattices, and region-specific cells are achieved by modifying query points and properties during SDF solving. A slicing method is realized by adjusting certain rendering configurations, and it corrects internal-external blending errors.

The former two modules are explained in Sections 4 and 5. The extensions are in Section 6.

## 4. Mega-scale rendering for lattices

Mega-scale rendering is mainly achieved via restricting distance query for the lattice within a single cell. We start by detailing the distance query for a cell. The correction for accurate rendering of the entire lattice is discussed next.

### 4.1. Distance query within a single cell

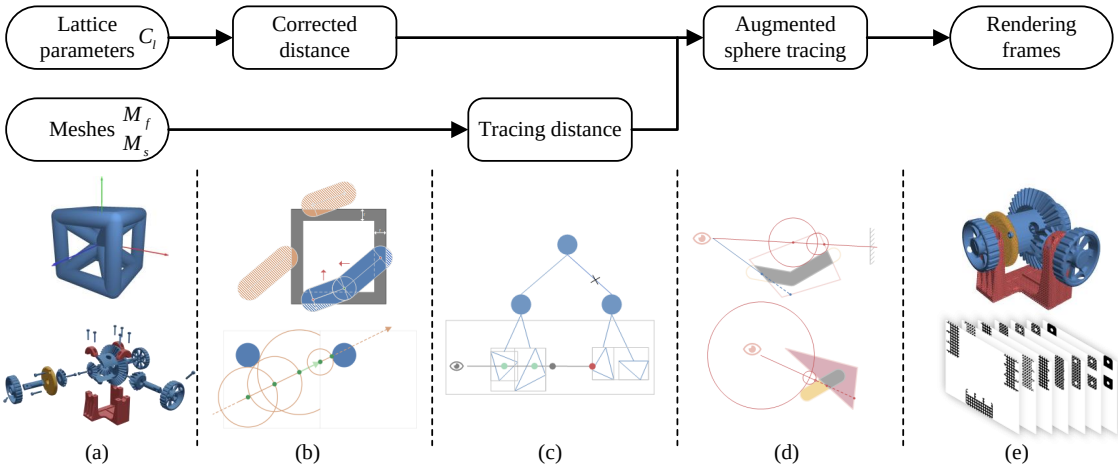
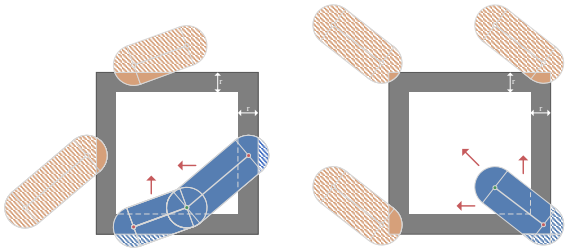



Figure 2: Overview of the augmented sphere tracing (AST) algorithm. The algorithm takes lattice descriptions and outer shell meshes as inputs (a) and produces shell-lattice structures as outputs (e). For lattices (b), corrected distances are computed through distance queries within a single cell. For meshes (c), tracing distances are obtained via BVH-based ray-mesh intersections. Although the distances are not exact SDF values, they are calculated efficiently and serve as inputs to the augmented sphere tracing process, robustly incorporating Boolean operations to generate the final rendering results (d).

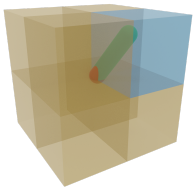


(a) Beam augmentation triggered by edges in two dimensions.

(b) Beam augmentation triggered by vertices in two dimensions .

Figure 3: 2D schematic on beam augmentation. The blue beams are manually designed, and the yellow ones come from their beam augmentation. The shadowed segments are not rendered in the current cell containing the query point, and the arrows indicate the translation direction.

Simply confining the lattice cell within a cube does not render the entire element, as some beams extend beyond it. For example, in the inset, only the green volume within the blue cell is rendered, while the red part is missing. Specifically, it happens when



$$\exists \star \in \{x, y, z\}, |\mathbf{N}[i, j]_\star| > 1 - \frac{2r}{\mathbf{S}_\star}. \tag{3}$$

Based on the above observations, we are to render with a cube element all the regions of the beams within the cell. The approach is illustrated in Figure 3 and explained in Algorithm 1.

In Algorithm 1, we sequentially consider three scenarios regarding the cube's vertices, edges, and faces while traversing all nodes $\mathbf{N}[i, j]$ in the unit cell. First, we check if any of the 8 vertices are inside the node surface of $\mathbf{N}[i, j]$. If so, all 8 surrounding cells render the beam, and the beam is copied and moved to the other 7 vertices (lines 4-16). If not, we check if any of the 12 edges pass through the surface. If so, the 4 surrounding cells render the beam, and it is copied and translated to the other 3 parallel edges (lines 18-35). If not, we finally check if any of the 6 faces intersect the surface. If so, the beam is copied once and translated to the opposite face (lines 39-46). Lattice symmetry can also accelerate the traversal (line 4).

While $\mathbf{B}$ is the user-defined beam list within the unit cell, $\hat{\mathbf{B}}$ includes the beams that intersect the cell. With $\hat{\mathbf{B}} = \{\hat{\mathbf{B}}[i], i = 1, \cdots, \hat{n_b}\}$ obtained, the SDF value of this cell (denoted as $\boldsymbol{F}_b$) is the minimum of all the SDF values of the beams in $\hat{\mathbf{B}}$, i.e.,

$$\boldsymbol{F}_b(\mathbf{P}) = \min_{i=0}^{\hat{n_b}-1}(\boldsymbol{F}_{b,i}(\mathbf{P})). \tag{4}$$

Note that many cases may not need Algorithm 1 where beams connecting neighboring cells can complement each other, and we allow users to toggle the algorithm on or off.

### 4.2. Local distance calibration

We now explain how to obtain the query distance of the entire lattice from that of the unit cell. Calculating the SDF for the entire lattice directly is computationally intensive. A common practice is to leverage the localization of porous structures [14].

*Query point mapping.* Since beams in the unit cell have been well defined in 4.1, we consider constructing a mapping between it and an arbitrary cell of the lattice. Rather than mapping beams from the unit cell to a certain cell of the lattice, as widely applied in [7, 31], we map the query points from a global coordinate (denoted as $\mathbf{P}_g$) to a local one (denoted as $\mathbf{P}_l$), which is similar to the method in [53]. Based on the consistency of the query point's relative position within the original cell and the unit cell, the mapping is achieved by rounding off:

$$\mathbf{P}_l = (\mathbf{P}_g - \mathbf{V}_{min}) \bmod \mathbf{S} - 0.5 \cdot \mathbf{S}, \tag{5}$$

where $\mathbf{V}_{min}$ is the minimal coordinate of the generated lattice.

By query point mapping, $\boldsymbol{F}_b$ at location $\mathbf{P}_g$ is

$$\boldsymbol{F}_b(\mathbf{P}_g) = \boldsymbol{F}_b(\mathbf{P}_l). \tag{6}$$

The equation is reasonable in that moving the query point from a specific cell to the unit one and copying intra-cell primitives from the original cell to the specific one yield an equal $\boldsymbol{F}_b$ value.

*Distance calibration.* Since $\boldsymbol{F}_b(\mathbf{P}_g)$ considers only one cell, it is not the exact SDF value of the entire lattice, as depicted in Figure 4 (a). When the total volume of beams in a cell is small and their distribution is not uniform enough, an error occurs in which $\boldsymbol{F}_b(\mathbf{P}_g)$ represents an excessive distance. With that distance, the

**Algorithm 1 BeamAugmentation**

**Input:** Origin beam list $\mathbf{B}$ of size $n_b$, beam radius $r$, cell size $\mathbf{S}$;
**Output:** Extended beam list $\hat{\mathbf{B}}$;

$\hat{\mathbf{B}} := \mathbf{B}$
**for** $i := 0 \rightarrow n_b - 1$, $j := 0 \rightarrow 1$ **do** ▷ For each beam node
  $\mathbf{C} := (\{1,1,1\} - |\mathbf{N}[i,j]|) \odot (0.5 \cdot \mathbf{S})$
  **if** $\|\mathbf{C}\| < r$ **then** ▷ Examine the cell vertices
    **for** $k := 1 \rightarrow 7$ **do**
      $\hat{\mathbf{N}}[0] := \mathbf{N}[i,0], \hat{\mathbf{N}}[1] := \mathbf{N}[i,1]$
      **for** $\star := 0 \rightarrow 2$ **do**
        **if** $k\&(1 << \star)$ **then**
          $\hat{\mathbf{N}}[0]_\star := \hat{\mathbf{N}}[0]_\star - 2 \cdot sign(\mathbf{N}[i,j]_\star)$
          $\hat{\mathbf{N}}[1]_\star := \hat{\mathbf{N}}[1]_\star - 2 \cdot sign(\mathbf{N}[i,j]_\star)$
        **end if**
      **end for**
      $\hat{\mathbf{B}} := \hat{\mathbf{B}} \cup \{\{\hat{\mathbf{N}}[0], \hat{\mathbf{N}}[1]\}\}$
    **end for**
    continue
  **end if**
  $addByEdge := False$
  **for** $\star := 0 \rightarrow 2$ **do** ▷ Examine the cell edges
    $\hat{\mathbf{C}} := \mathbf{C}, \hat{\mathbf{C}}_\star := 0$
    **if** $\|\hat{\mathbf{C}}\| < r$ **then**
      **for** $k := 1 \rightarrow 3$ **do**
        $\hat{\mathbf{N}}[0] := \mathbf{N}[i,0], \hat{\mathbf{N}}[1] := \mathbf{N}[i,1]$
        **for** $l := 0 \rightarrow 2$ **do**
          **if** $k\&(1 << l)$ **then**
            $\hat{l} := (l + \star + 1) \bmod 3$
            $\hat{\mathbf{N}}[0]_{\hat{l}} := \hat{\mathbf{N}}[0]_{\hat{l}} - 2 \cdot sign(\mathbf{N}[i,j]_{\hat{l}})$
            $\hat{\mathbf{N}}[1]_{\hat{l}} := \hat{\mathbf{N}}[1]_{\hat{l}} - 2 \cdot sign(\mathbf{N}[i,j]_{\hat{l}})$
          **end if**
        **end for**
        $\hat{\mathbf{B}} := \hat{\mathbf{B}} \cup \{\{\hat{\mathbf{N}}[0], \hat{\mathbf{N}}[1]\}\}$
      **end for**
      $addByEdge := True$
      break
    **end if**
  **end for**
  **if** $addByEdge$ **then**
    continue
  **end if**
  **for** $\star := 0 \rightarrow 2$ **do** ▷ Examine the cell faces
    **if** $|\mathbf{C}_\star| < r$ **then**
      $\hat{\mathbf{N}}[0] := \mathbf{N}[i,0], \hat{\mathbf{N}}[1] := \mathbf{N}[i,1]$
      $\hat{\mathbf{N}}[0]_\star := \hat{\mathbf{N}}[0]_\star - 2 \cdot sign(\mathbf{N}[i,j]_\star)$
      $\hat{\mathbf{N}}[1]_\star := \hat{\mathbf{N}}[1]_\star - 2 \cdot sign(\mathbf{N}[i,j]_\star)$
      $\hat{\mathbf{B}} := \hat{\mathbf{B}} \cup \{\{\hat{\mathbf{N}}[0], \hat{\mathbf{N}}[1]\}\}$
    **end if**
  **end for**
**end for**
**return** $\hat{\mathbf{B}}$

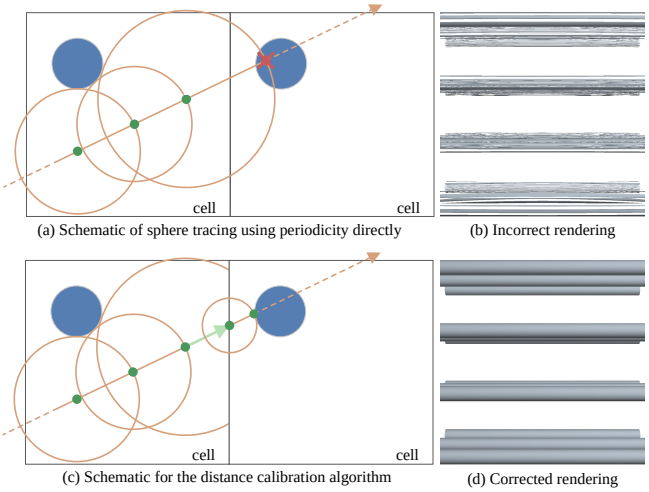
ray either enters or misses the beams in subsequent cells, resulting in a crippled rendering, as indicated in Figure 4 (b).

To address the issue, we propose Algorithm 2 to reduce potentially excessive distances to just safe distances. As illustrated in Figure 4 (c), the basic idea is that the ray can only enter a new cell by first reaching the boundary of the current cell and then

**Algorithm 2 DistanceCalibration**

**Input:** Local query point $\mathbf{P}_l$, ray direction $\mathbf{D}$, cell size $\mathbf{S}$, small number $\epsilon_t$;
**Output:** Corrected tracing distance;

1: $\hat{\mathbf{S}} := 0.5 \cdot \mathbf{S}$
2: $t := \infty$
3: **for** $\star \in \{0,1,2\}$ **do**
4:  $\hat{t} := \lfloor \frac{\hat{\mathbf{S}}_\star - \mathbf{P}_\star}{\mathbf{D}_\star} \rfloor$
5:  **if** $\hat{t} > 0$ **then**
6:  $t := min(\hat{t}, t)$
7:  **else**
8:  $\hat{t} := \lfloor \frac{-\hat{\mathbf{S}}_\star - \mathbf{P}_\star}{\mathbf{D}_\star} \rfloor$
9:  **if** $\hat{t} > 0$ **then**
10:  $t := min(\hat{t}, t)$
11:  **end if**
12:  **end if**
13: **end for**
14: **return** $min(t + \epsilon_t, \boldsymbol{F}_b(\mathbf{P}_l))$




(a) Schematic of sphere tracing using periodicity directly (b) Incorrect rendering

(c) Schematic for the distance calibration algorithm (d) Corrected rendering

Figure 4: 2D schematics of the incorrect rendering (a & b) and the solution provided by Algorithm 2 (c & d). The blue circles indicate the beam cross-sections. In this example, a cell contains only a horizontal beam.

advancing by a very small value $\epsilon_t$. We calculate the 3 possible forward distances required for the ray to reach the cell boundary plane (lines 3-13) and take their minimum (lines 6 and 10). The additional forward distance $\epsilon_t$, which guarantees that the ray enters the next cell, should neither be too small (causing the cell's boundaries to be rendered) nor too large (falling to solve the above problem). We set it equal to the termination threshold of sphere tracing, $\epsilon$, which ultimately produces the proper result in Figure 4 (d).

In the absence of distance calibration, sphere tracing the SDF efficiently for the entire lattice would also require leveraging the localization of the SDF. A direct approach involves evaluating the exact SDF considering 27 cells (the current cell containing $\mathbf{P}_g$ and its 26 neighbors). This can be reduced to evaluating 8 cells (the current cell and its 7 neighbors closest to $\mathbf{P}_g$) by taking into account the position of $\mathbf{P}_g$ within the current cell [53]. Considering that evaluating the SDF for beams is the most time-consuming part of lattice rendering, we compare the number of cells evaluated per ray for the proposed method and the 8-cell method in [53] in Figure 5. When the beams connecting the cells restore the missing parts of each other at the boundaries,

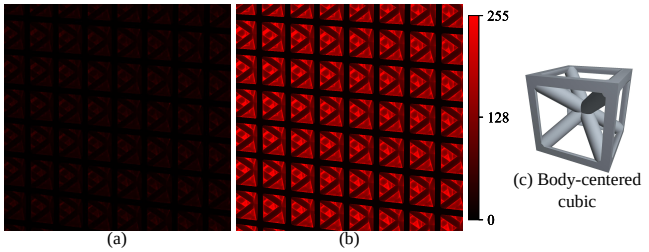



Figure 5: A comparison of the number of cell evaluations per ray between our method (a) and the 8-cell method in [53] (b). The cell type is the body-centered cubic (c). The number of evaluations is represented by the R-channel value in the ray's color, capped at a maximum of 255.

Algorithm 1 is omitted. In such cases, the cost of evaluating a cell is the same for both methods, and the proposed method requires much fewer beam evaluations for most rays. Among the widely used cell types summarized by [54] (See Figure 11), only the diamond type necessitates Algorithm 1, where the number of the beams in a cell is augmented from $12 + 4 = 16$ to $12 \times 2 + 4 \times 8 = 56$, a 3.5-fold increase.

## 5. Heterogeneous rendering for hybrid forms

We now explain the heterogeneous rendering of hybrid inputs (mesh and SDF) through tracing distance computation and the pruning-based acceleration.

### *5.1. Tracing distance for mesh shells*

We mainly explain how to obtain the tracing distances. Once it is done, the Boolean operations of intersection and union between the mesh and SDF can be achieved by simply taking maximum and minimum values.

Given $M_s$ and $M_f$, their BVHs are respectively constructed in advance before entering the rendering pipeline. The tracing distance calculation consists of two stages. First, the intersections between the ray and the mesh shells are solved before sphere tracing. Second, the tracing distances are derived from these intersections and used as inputs for sphere tracing at each iteration.

In the first phase, we follow the standard practice in ray tracing to obtain the ray-mesh intersections by traversing BVHs. We denote the ray-mesh intersection respectively for the solid part $M_s$ and the filled $M_f$ for later usage. For $M_s$, as rays never enter a solid mesh, we only record the first intersection, where the hit face is denoted as $I_s$, and its distance to the camera as $t_s$. For $M_f$, multiple intersections are stored as the lattice structure allows multiple mesh faces to be visible. We sort their distances to the camera in ascending order in a list $\mathbf{t}_f$ associated with faces $\mathbf{I}_f$.

In the second phase, the tracing distances of $M_s$ are relatively straightforward (Figure 6). When there is no intersection, a large number is returned to make the tracing stop. When the ray has not entered the mesh, the distance to the first intersection point is returned. We now mainly explain it for the region $M_f$ to be filled, as detailed via Algorithm 3 and illustrated Figures 7.

In Algorithm 3, when the query point is outside the shell mesh, we step to the next intersection (line 4). When the ray traces within the shell mesh, it returns a large negative number (line 6). When the ray hits a face before entering the mesh, the algorithm calculates and stores the face normal for shading and returns a value slightly less than $\epsilon$ (lines 10 and 11). If the ray hits a face before leaving the mesh, a large negative number is returned (line 13). The membership of the query point is determined by the parity of the remaining intersections in front of this point (lines 3 and 9): an even count indicates the exterior, while an odd count indicates the interior. When the ray has left the mesh, a large value is returned to quickly end the tracing (line 17).

**Algorithm 3 TracingDistance**

**Input:** Hit face list $\mathbf{I}_f$, intersection distance list $\mathbf{t}_f$, list size $n$, distance from the query point to the camera $t$;
**Output:** Tracing distance for $M_f$;

1: **for** $i := 0 \rightarrow n - 1$ **do** ▷ Traverse each intersection
2: **if** $t < \mathbf{t}_f[i]$ **then**
3: **if** $(n - i) \bmod 2 = 0$ **then** ▷ Outside the mesh
4: **return** $\mathbf{t}_f[i] - t$
5: **else** ▷ In the mesh
6: **return** $-\infty$
7: **end if**
8: **else if** $t = \mathbf{t}_f[i]$ **then**
9: **if** $(n - i) \bmod 2 = 0$ **then** ▷ Hit before entering
10: Calculate and store the normal of $\mathbf{I}_f[i]$
11: **return** $(1 - \epsilon) \cdot \epsilon$
12: **else** ▷ Hit before leaving
13: **return** $-\infty$
14: **end if**
15: **end if**
16: **end for**
17: **return** $\infty$ ▷ $t > \mathbf{t}_f[n - 1]$

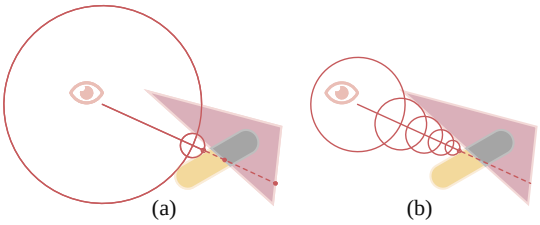



Figure 6: Comparison of heterogeneous union (a) and general sphere tracing (b).

The algorithm's efficiency comes from two main aspects. (1) The number of tracing steps: The proposed approach avoids numerous iterations that sphere tracing would typically require near mesh faces; (2) Time consumption in each iteration: Compared to approaches that compute the SDF of the mesh by utilizing complex accelerating structures at each iteration, the proposed approaches only visit BVHs once before sphere tracing.

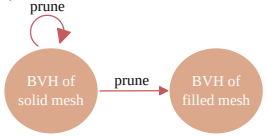
Another significant advantage of heterogeneous ray casting is its ability to preserve all details of $M_f$ and $M_s$. Methods based on explicit-implicit conversions, such as sampled level sets and marching cubes, often result in loss of details.

### *5.2. BVH pruning for rendering acceleration*

The calculation of the above ray-mesh intersection is further accelerated by pruning the BVHs during the depth-first traversal, as explained below.

For the BVH traversal of $M_s$, we follow a common pruning method: (1) always prioritize access to the node closer to the camera among the two child nodes, and (2) discard nodes where the closet distances from the camera to the bounding boxes are farther than the current $t_s$ (see Figure 8 (a)).

The situation is different for $M_f$. Since all its intersection points with the ray may be visible, the BVH traversal of $M_f$ cannot be



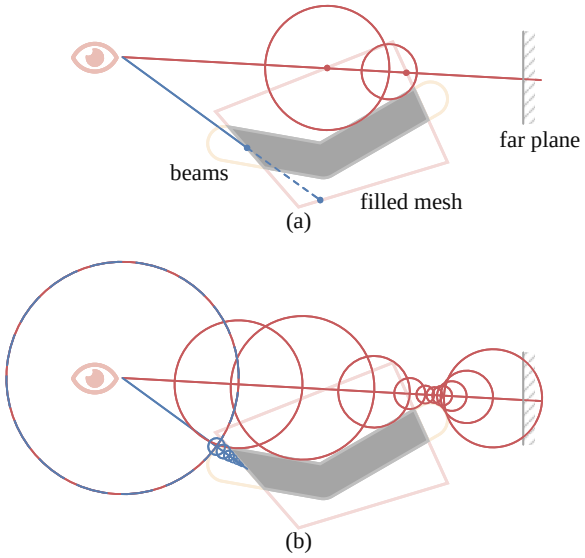



Figure 7: Comparison of heterogeneous intersection (a) and general sphere tracing (b).

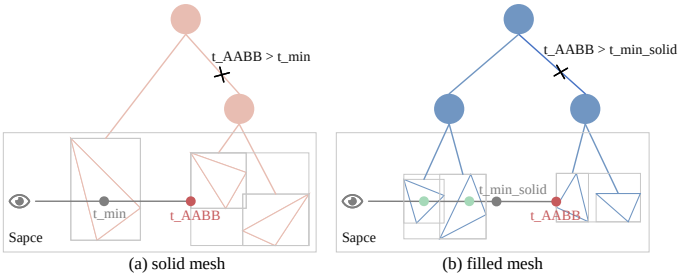



Figure 8: Schematic of pruning BVH of $M_f$ (a) and $M_s$ (b). The × symbol marks the discarded branch.

pruned using $\mathbf{t}_f$. Thanks to the prerequisite that $M_s \cap M_f = \emptyset$, we propose utilizing $t_s$ to prune BVH traversal of $M_f$. The pruning relationship between $M_s$ and $M_f$ is illustrated in the inset. $I_s$ and $t_s$ should be solved prior to $\mathbf{I}_f$ and $\mathbf{t}_f$. When traversing the BVH of $M_f$, any node with an AABB whose closest distance to the camera exceeds $t_s$ is discarded.

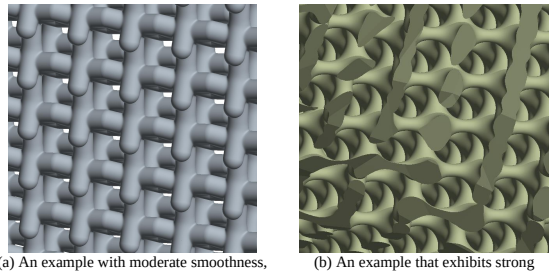
Considering the cases where the camera starts from within $M_f$, we also prune the branches where the max signed distance from the camera to the AABB along the ray is less than 0. Such nodes are discarded for both $M_f$ and $M_s$. Dropping the intersections before the ray's starting point shifts $i$ and $n$ in Algorithm 3 by an equal amount, so the parity of $n - i$ remains unchanged.

## 6. Extensions

We now extend $\Omega$ to enable control over blending, deformation, property distributions, and cell types by modifying the SDF or adjusting query points. The extended lattice covers a great number of structures that are involved in industry and academia, typically the 3D non-stochastic cellular structures and the warped-regular lattices [11, 28, 54, 25].

### 6.1. Smooth blending

A proper smooth blending improves physical attributes over sharp connections [55]. Two types of blending are studied: (1) *internal blending* within $L(C_l)$, and (2) *internal-external blending* between $L(C_l)$ and the complement of $M_f$.



(a) An example with moderate smoothness, featuring an arbitrary unit cell design that breaks the symmetry constraint.

(b) An example that exhibits strong smoothness with a unit cell design that obeys the symmetry constraint.

Figure 9: A demonstration of cases where only one of two triggers (large smoothness or asymmetry) is met, without discontinuities occurring.

#### 6.1.1. Internal blending

Internal blending focuses solely on achieving smooth transitions between beams. For SDF blending, the minimum and maximum functions used in Boolean operations are replaced by the corresponding blending functions. We use the KS function [56] to support multiple components and order invariance. This process aligns with the localization strategy in Section 4.2, considering only beams within the current cell for blending:

$$\boldsymbol{F}_{\cup I}(\mathbf{P}) = -\frac{1}{p} ln\left(\sum_{i=0}^{n_b-1} e^{-p(\boldsymbol{F}_{b,i}(\mathbf{P})-\boldsymbol{F}_b(\mathbf{P}))}\right) + \boldsymbol{F}_b(\mathbf{P}) \quad s.t.\ p > 0, \quad (7)$$

where $p$ is the smoothness term (the smaller, the smoother).

This method introduces minimal rendering overhead and performs well in most scenarios. However, in rare cases, it may cause volume discontinuities at cell boundaries due to inconsistent blending of beams connected on boundaries. This issue only arises when the value of $p$ is too small and an asymmetry along the connection direction appears: if the cells are connected in the $\star$ direction, the beams in the cell are not symmetric with respect to the plane perpendicular to the $\star$-axis and passing through the cell center. Note that although blending over larger or infinite regions can minimize or eliminate these discontinuities without this constraint, it incurs prohibitive computational costs due to increased SDF evaluations.

The internal blending performs well when the large smoothness and asymmetry do not occur simultaneously, as shown in Figure 9. Figure 10 shows the impact of $p$ and $\mathbf{S}_\star/r$ on discontinuities when the symmetry constraint is not satisfied. We fix $\mathbf{S}_\star$ at 0.2 and modify $r$ and $p$. When $p$ is small, discontinuities appear; the smaller $\mathbf{S}_\star/r$ becomes, the more pronounced these discontinuities are. Given the potential impact of discontinuities on the structure's physical properties [57, 58], we recommend a smoothing range of $p \geq 30/\mathbf{S}_\star$ when $\mathbf{S}_\star/r \geq 5$ to avoid excessive discontinuities in typical applications.

In most manufacturing scenarios, the symmetry constraint is satisfied, as illustrated by the types in 11 (excluding the diamond type) [54].

#### 6.1.2. Internal-external blending

Internal-external blending refers to smooth transitions along the truncated beams by shell $M_f$. It is achieved by a smooth union operation between $L(C_l)$ and the complement of $M_f$ using its SDF.

The SDF computation directly achieved from point to mesh query in each tracing iteration is too slow to meet our high demand of interactive rendering, although great acceleration has been achieved [33, 59]. Such a strategy is used in our slicing stage to ensure correct manufacturing.

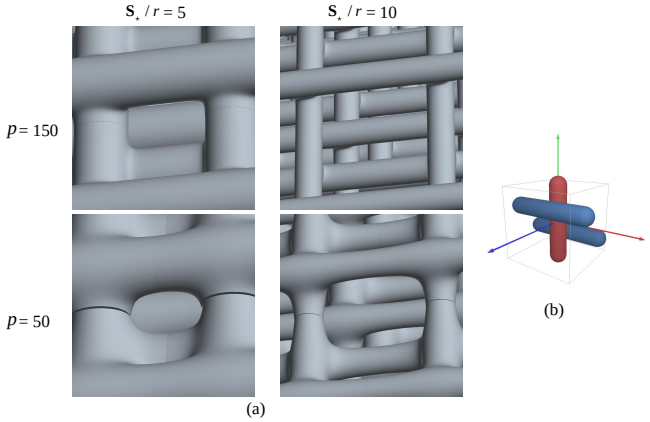



Figure 10: Comparison of results under different $\mathbf{S}_\star/r$ and $p$ values (a), using a unit cell not satisfying the constraint (b). The three components of $\mathbf{S}$ are set equal. As $\mathbf{S}_\star$ and $p$ decrease and $r$ increases, discontinuities become more pronounced.

We thus precompute a level set $\hat{M}_f$ approximating shell mesh $M_f$. Let $\boldsymbol{F}_{\hat{M}_f}(\mathbf{P})$ be the precomputed SDF value of $\hat{M}_f$ achieved by trilinear interpolation among 8 surrounding grid points. Taking $\boldsymbol{F}_{\cup I}$ and $-\boldsymbol{F}_{\hat{M}_f}(\mathbf{P})$ as inputs, the KS function gives the desired internal-external blending. During the process, we keep $M_f$ unchanged for shell rendering, thus preserving the mesh details from the intersection operations in Section 5.1.

In practice, given the detail-erasing nature of blending operations, a low-resolution level set is accurate enough. In our implementation, a grid with a fixed resolution of $128^3$ is used as the level set. Precise SDF values are presampled at the grid points.

Figure 11 shows internal-external and slight internal blending for classic unit cells [54] observed within $M_f$. Figure 12 illustrates varying degrees of internal-external blends for the body-centered cubic cell type. A $p$ value of $p \geq 2.5/\mathbf{S}_\star$ (for densities where $\mathbf{S}_\star/r \geq 5$) is recommended to maintain structure visibility. Excessive blending causes rays from within the shell to start inside the microstructure, complicating rendering.

### 6.2. Repetition with variations

We now describe how to introduce variations to repetitions while keeping the frame rate. The parameter adjustments of these variations can also be made in real-time.

#### 6.2.1. Field-directed distributions

Since AST takes a local view of the entire lattice, each query point only concerns local differences when rendering varying structures.

We explain this using a cosine trigonometric field as an example. Given the positional variable $\mathbf{P}_g$, each component on the coordinate $\star$ of a field-directed triplet is

$$\boldsymbol{T}_{tri}(\mathbf{v}_\star, \mathbf{P}_g; A, \omega, \phi) = \mathbf{v}_\star + \mathbf{v}_\star A cos(\omega \mathbf{P}_{g,\star} + \phi), \tag{8}$$

where $\mathbf{v}$, $A$, $\omega$, and $\phi$ are the triplets, amplitude, frequency, and phase, respectively. And a field-directed scalar applied to an original scalar $v$ is given by

$$\boldsymbol{T}_{sca}(v, \mathbf{P}_g; \mathbf{A}, \boldsymbol{\Omega}, \boldsymbol{\Phi}) = \frac{1}{3} \sum_{\star \in \{x,y,z\}} \boldsymbol{T}_{tri}(v, \mathbf{P}_g; \mathbf{A}_\star, \boldsymbol{\Omega}_\star, \boldsymbol{\Phi}_\star), \tag{9}$$

where $\mathbf{A}$, $\boldsymbol{\Omega}$, and $\boldsymbol{\Phi}$ are triplets representing the amplitude, frequency, and phase, respectively. They are the only parameters passed to the GPU to describe the field. The field-directed properties are evaluated at runtime during tracing. The additional memory overhead for a field-directed property is a total of $3 \times 3 \times 4 = 36$ bytes (assuming single-precision) in the case of the trigonometric field.

*Field-directed radius / internal smoothness.* It is trivial to render a lattice of field-directed radius $r$ (or smoothness term $p$) in AST, as long as the field is continuous and does not vary drastically. We just substitute $\boldsymbol{T}_{sca}(r, \mathbf{P}_g)$ for $r$ (or $\boldsymbol{T}_{sca}(p, \mathbf{P}_g)$ for $p$) when calculating $\boldsymbol{F}_b$ in Equation (1) in each tracing iteration. One exception has to be taken care of: some beams, which previously did not require beam augmentation, now do. This occurs because the enlarged portion of the beams may exceed the cell boundary. A solution to this is to substitute $r$ in line 4, 20 and 40 of Algorithm 1 with $r(1 + \frac{1}{3}\sum_\star |\mathbf{A}_\star|)$.

*Field-directed cell size.* Our mega-scale module relies on the rounding off operation in Equation (5), leveraging the consistency of cell sizes $\mathbf{S}$ across all cells. Interestingly, the rounding off still works when cell sizes vary. We simply replace the cell size $\mathbf{S}$ with $\boldsymbol{T}_{tri}(\mathbf{S}, \mathbf{P}_g)$ in Equation (5) and Algorithm 2, and the rest rendering process remains unchanged. Each query point assumes rendering a lattice with a consistent cell size, even though the cell size varies continuously across the space. The field should not vary drastically, as this can lead to rendering imperfections. Specifically, if rays advance too aggressively along the direction of field variation, they may penetrate the beam surfaces. It can be addressed by multiplying the forward distance by a decay constant. An example is illustrated in Figure 13.

#### 6.2.2. Warped lattice

Warped-regular lattices enhance the flexibility of regular lattices by allowing deformations that go beyond traditional geometric constraints. This enables them to better meet a wider range of physical and aesthetic requirements [11, 29, 61]. In warped-regular lattices, transformations are applied to the volumetric micro-elements of beams, rather than just the node centers as in steady lattices. We introduce two types of warping: *bending*, a global transformation that alters the spatial distribution of cells, and *twisting*, a local transformation that deforms the beams without changing the cell arrangement.

Deriving the form of implicit representations transformed by a rotation matrix is challenging, but it can be equivalently expressed by applying the inverse matrix to the query point $\mathbf{P}$. From this, to render a warped lattice with spatially varying rotation, we inversely warp the query point at each tracing iteration. This approach is inspired by and explored in detail for general cases in [62, 63, 64]. Note that our definitions of bend and twist differ slightly from those in traditional implicit model deformations, but they capture similar effects [62, 63].

Denote by $\boldsymbol{R}(\psi(\mathbf{P}_g))$ a function that returns the rotation matrix for beams at a rotation angle $\psi(\mathbf{P}_g)$. For the same effect, its inverse matrix $\boldsymbol{R}(\psi(\mathbf{P}_g))^{-1}$ is applied to query points. We omit details such as the axis of rotation as they are not the focus. For the bending, $\boldsymbol{R}(\psi(\mathbf{P}_g))^{-1}$ is applied to the original query point $\mathbf{P}_g$:

$$\mathbf{P}'_g = \boldsymbol{R}(\psi(\mathbf{P}_g))^{-1} \mathbf{P}_g. \tag{10}$$

By contrast, the twisting is applied to the mapped query point $\mathbf{P}_l$:

$$\mathbf{P}'_l = \boldsymbol{R}(\psi(\mathbf{P}_g))^{-1} \mathbf{P}_l. \tag{11}$$

See examples in Figure 14 on the bending and twisting, where,

$$\psi(\mathbf{P}_g) = \alpha \mathbf{P}_{g,\star}. \tag{12}$$

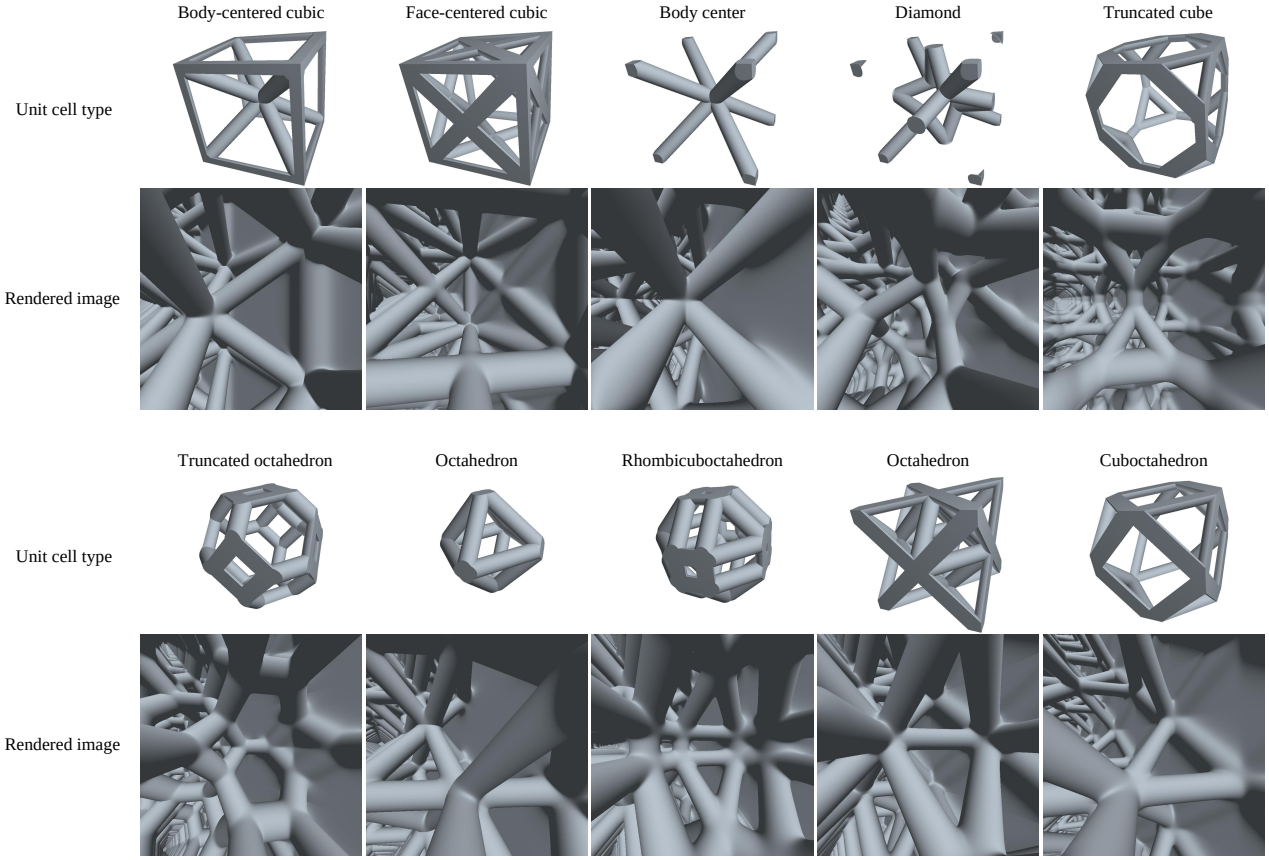



Figure 11: Rendered images for Ω using various unit cell types, with internal-external and slight internal blending enabled. The camera captures photos from within Ω. The diamond type is the only one that necessitates Algorithm 1.

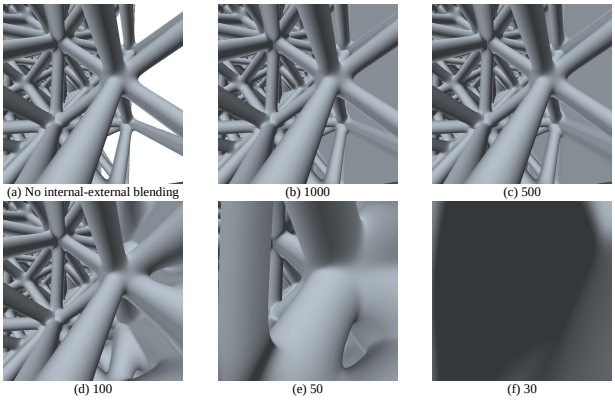



Figure 12: Comparison of internal-external blending at values of p: $+\infty$ (no internal-external blending applied), 1000, 500, 100, 50, and 30. Internal blending with $p = 800$ is simultaneously applied in all cases, and $\mathbf{S}_\star$ is fixed as 0.05.

$\alpha$ is the user-controlled coefficient for bending or twisting, and $\mathbf{P}_{g,\star}$ can be any component of $\mathbf{P}_g$.

Since the transformation is defined by a differentiable field, the Lipschitz bound of the warping can be derived using the chain rule. This bound is no larger than the product of the bounds of the field and the SDF [63]. The severity of the warping directly increases the bound. While this bound is not used in AST, it could be useful for subsequent applications.

### 6.3. Region-specific cell types

AST can easily support multiple unit cell types with specific copying rules, adding versatility to lattice design without significant computational or storage overhead.

The repetition rule for cells is highly flexible and is typically specified using their indices in the lattice. For example, the distinction can be determined by the parity of the sum of cell index components: cells with an even sum are assigned the first unit cell type, while those with an odd sum get the second type. For a given query point $\mathbf{P}_g$, the index of the cell it belongs to is calculated as:

$$\mathbf{I} = \left\lfloor \frac{\mathbf{P}_g - \mathbf{V}_{min}}{\boldsymbol{T}_{tri}(\mathbf{S}, \mathbf{P}_g)} \right\rfloor, \tag{13}$$

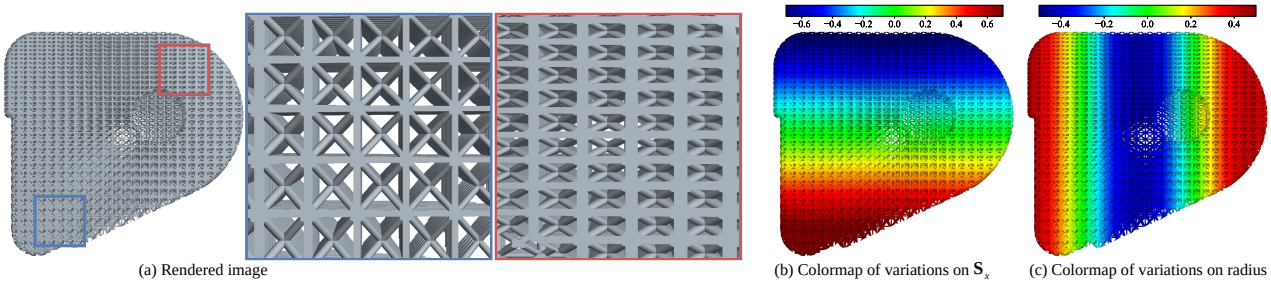
where $\lfloor\rfloor$ is a floor function.

To guarantee structural completeness, the beams from beam augmentation in a unit cell (Algorithm 1) are copied to other unit cells. And C0 continuity on adjacent boundaries should be satisfied by the cell types and the repetition rule. As a limitation, region-specific cell design should not coexist with internal blending. This is because neighboring cells of different types undergo distinct smooth blending, resulting in inconsistencies in the volume at the cell boundary. Figure 15 illustrates an example of region-specific cell types.

### 6.4. TPMS

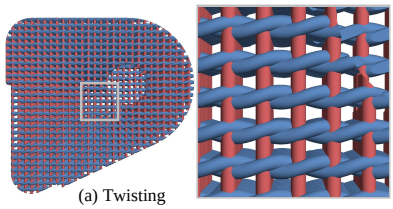
Although AST is specially designed for periodic lattices with beams as primitives, the heterogeneous rendering can accept a broader range of porous structures as inputs.

A typical example is the Triply Periodic Minimal Surface (TPMS). Two common types of TPMS, namely, Primitive ($P$)

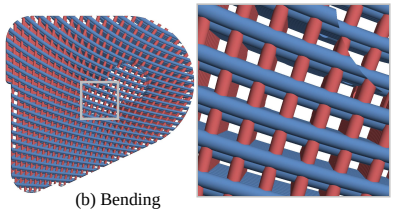



(a) Rendered image (b) Colormap of variations on $\mathbf{S}_x$ (c) Colormap of variations on radius

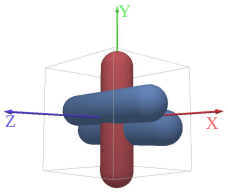
Figure 13: A filled booster model of an aircraft model sourced from the Thingi10k dataset [60] (File ID: 113419). Here, $\mathbf{S}_x$ and $r$ are controlled by trigonometric fields that vary along the $y$-axis (b) and $x$-axis (c), respectively.



(a) Twisting

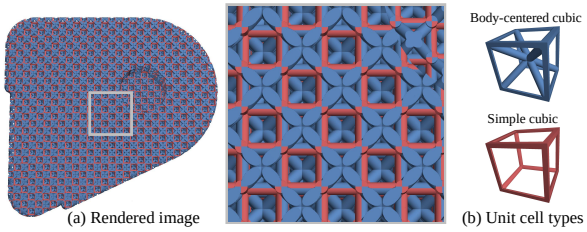


(b) Bending




(c) Unit cell design. Groups are distinguished by beam colors.

Figure 14: Results of the filled booster model with twisting (a) or bending (b) enabled, using a unit cell in (c). The beams are grouped, and the twisting is only applied to the blue group.




(a) Rendered image (b) Unit cell types

Figure 15: A rendered image of a filled booster model with region-specific cell types enabled (a). Two types of cells (b) are arranged in alternating sequences.

and Gyroid ($G$), are tested and are respectively defined as:

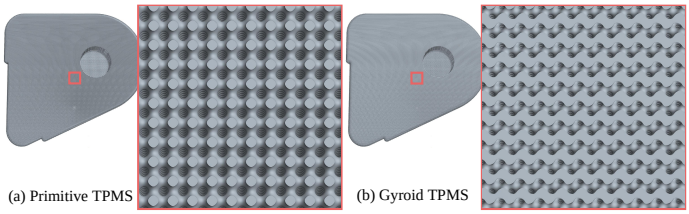
$$\boldsymbol{I}_P(\mathbf{P}) = cos(X) + cos(Y) + cos(Z) + b, \quad (14)$$

$$\boldsymbol{I}_G(\mathbf{P}) = sin(X)cos(Y) + sin(Z)cos(X) + sin(Y)cos(Z) + b, \quad (15)$$

where $X = \mathbf{\Theta}_x \mathbf{P}_{g,x}$, $Y = \mathbf{\Theta}_y \mathbf{P}_{g,y}$, $Z = \mathbf{\Theta}_z \mathbf{P}_{g,z}$, and $b$ is the level constant. $\mathbf{\Theta}_x$, $\mathbf{\Theta}_y$, $\mathbf{\Theta}_z$, and $b$ can all be edited with real-time response. We treat TPMS as a single structure due to its inherently periodic nature, and the cell-based mega-scale module is not required here. Note that $\boldsymbol{I}_P$ and $\boldsymbol{I}_G$ are not typical SDFs, and we address it by multiplying $\boldsymbol{I}_P$ or $\boldsymbol{I}_G$ by an attenuation factor to obtain a conservative distance. With this distance as the input for sphere tracing, the rest of AST remains unchanged. Figure 16 presents two examples.

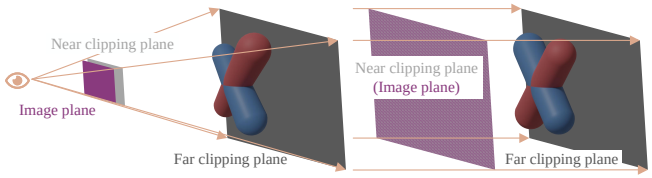
### 6.5. Slicing for fabrication

We now present the methods to generate slices for printing along any direction. Unlike the contour extraction-based methods in [35, 65], our slicing evaluates the fields directly for pixel classification, similar to the method in [66]. While it does not fully exploit the coherence of pixels and slices, it reuses the proposed rendering framework with existing GPU implementations



(a) Primitive TPMS (b) Gyroid TPMS

Figure 16: A booster model filled by $P$ or $G$ TPMSs.




(a) The perspective projection for rendering (b) The orthographic projection for slicing

Figure 17: Comparison between perspective projection for rendering and orthographic projection for slicing.

and ensures rendering consistency. The consideration of the appropriate slicing direction and supportability is beyond the scope of this paper.

We treat the slices as rendered images, modifying the proposed rendering process. First, we replace the perspective projection with an orthographic projection, as shown in Figure 17. In perspective projection, rays originate from a single viewpoint and pass through different pixels on the image plane, whereas in orthographic projection, rays are perpendicular to the image plane and start from the pixels themselves.

Second, the far clipping plane is no longer located behind the scene but instead very close to the near clipping plane. The near clipping plane now coincides with the image plane, rather than being slightly behind it (See Figure 17). The distance between the near and far planes represents the printing thickness of each slice.

Third, when internal-external blending is enabled, the SDF values of $M_f$ are no longer approximated by a low-resolution level set. Instead, they are precisely solved on the fly, reusing the BVH of $M_f$. Slicing is latency-insensitive and thus tolerates time-consuming distance queries to $M_f$.

Fourth, we adjust the slice width for a given height so that the boundary aligns with the object in the current view, thereby improving the utilization of slice pixels.

Finally, we remove the calculations on face normals and shading.

Figure 18 shows selected equally spaced slices with different slicing directions. The base model $M_f$ is sourced from the Thingi10k dataset [60] (File ID: 65149), which is a part of a gear system assembly sourced from the Thingiverse website (Thing ID: 11836). $M_s$ comprises three stand tops from this assembly. Time statistics for generating a single shell-lattice slice at differ-

ent resolutions are provided in Table 2. Higher resolutions can be easily obtained by stitching multiple slices together.

We emphasize that the contribution of this paper does not include a real-time slicing algorithm. Our slicing algorithm is simply adapted from the rendering pipeline, without the specialized optimizations seen in [35, 65, 66]. Further optimization of slicing efficiency is left for future work. Fortunately, existing GPU implementations enable the slicing algorithm to achieve reasonably good efficiency.

| Resolution | 1208 × 1024 | 2416 × 2048 | 3625 × 3072 | 4833 × 4096 | 6041 × 5120 | 7249 × 6144 |
|---|---|---|---|---|---|---|
| Runtime (*ms*) | 45.806 | 143.30 | 300.54 | 534.3258 | 829.46 | 1237.74 |

Table 2: Average time cost of generating a single slice at different resolutions.

## 7. Evaluation

The assessment of AST consists of two aspects: comparing it with existing methods and analyzing its own performance.

### *7.1. Implementation*

We implement AST using WebGPU, a cross-platform graphics and computation API, with the implementation provided by Google in 2023 [67]. All experiments were conducted on an ordinary desktop with an AMD Ryzen 4750G CPU and an NVIDIA GeForce RTX 3090 GPU.

All ray-level methods, including Algorithm 2, Algorithm 3, and extensions in Section 6 are executed on the GPU. Algorithm 1 runs on the CPU because it is applied beforehand and is low in parallelism. Complex data, such as $M_s$, $M_f$, $\hat{\mathbf{B}}$, and BVHs, are copied from CPU memory to *storage*-type GPU memory, as storage memory supports larger data blocks. All other parameters in Table 1, along with camera parameters and the model transformation matrix, use *uniform*-type GPU memory, which is more efficient but has a smaller capacity and stricter data format requirements [67]. Whenever any data is modified, we transfer only the updated data back to the GPU. When the parameters in Table 1 are modified, the shader does not need to be recompiled. In practical use, new frame requests occur only after data updates. However, in subsequent runtime tests, requests were continuously sent to the renderer to measure frame time.

### *7.2. Comparative analysis*

To assess the practical value of AST, we compare it against an STL-based pipeline and a fully implicit pipeline. For the STL-based pipeline, we opt for Marching Cubes [39] as the triangulation technology, given its widespread usage and the availability of an optimized GPU implementation from CUDA samples [68]. This pipeline is referred to as MCM and operates under CUDA Toolkit version 12.4. In the fully implicit pipeline, we replace our heterogeneous module with a presampled level set, referring to it as LSM. $M_f$ is the stand base model in Thingi10k [60]. To focus on comparing the results of lattice filling, $M_s$ is not set.

To begin with, we observe that MCM requires a minimum of approximately $16^3$ voxels per cell *to achieve visually acceptable rendering effects*, as indicated in Figure 19.

*Storage.* This comparison focuses on the overhead of graphics memory (VRAM), as it is often a limiting factor and can create bottlenecks in practical applications. Each cell in MCM maintains $16^3$ Marching Cubes voxels. Since the ability of MCM and LSM to display details is positively correlated with grid resolution, we report the VRAM usage of both methods at different resolutions, as shown in Figure 20. Although the VRAM usage of AST is independent of resolution or detail fidelity, we label this constant value in the figure for ease of comparison. The primary VRAM overheads of the MCM comprise three parts: (1) the flag information used in the Marching Cubes voxel classification, (2) the sampled SDF level set of the mesh shell during voxel classification, and (3) the vertex data during triangular mesh generation. Using our method to render the internal lattice, the VRAM overhead of LSM is almost entirely due to maintaining the level set grid. However, its VRAM overhead still increases sharply with resolution, similar to MCM. In contrast, the main VRAM overhead in AST consists only of the raw representations of mesh shells, their BVHs, and the fixed-resolution level set of $M_f$.

*Details.* We evaluate the capability of AST to accurately render the fine details of lattice structures. We set the resolution of both MCM and LSM to $512^3$ (a resolution of $1024^3$ causes programs to crash). The camera focuses on the sharp edges of $M_f$ to highlight detail rendering. The comparison with MCM is shown in Figure 21 and with LSM in Figure 22. MCM demonstrates significant limitations in rendering the fine details both of the outer shell and of the inner beams; note that these limitations also extend to slicing for manufacturing processes. Although LSM renders internal beams effectively, it eliminates the sharp edges and other detailed features of the original mesh. In contrast, AST produces high-quality rendering that captures a wide range of details, including the shell's sharp edges, the beam's smooth surfaces, and the nuanced articulation between them. These findings underscore AST's potential for high-quality visualization in applications involving large-scale shell-lattice structures.

*Runtime.* This experiment compares efficiency. Since LSM and AST render the internal lattice in the same way—this being the primary source of rendering overhead—their runtime differences are minimal (see frame time in Figure 22). Therefore, we focus on comparing AST and MCM. We conduct tests with varying cell numbers (#$C$) within the AABB of $M_f$ and different beam counts (#$B$) in the unit cell. #$C$ ranges from hundreds to billions for AST but is limited to $8^3$ to $32^3$ for MCM due to VRAM constraints. Each cell is cubic with $\mathbf{S}_\star/r = 5$. We employ four unit cell types with increasing #$B$: simple cubic, face-centered cubic, truncated cube, and rhombicuboctahedron. Tests are performed on both a stand base model (Stand) and its AABB (Cube) as $M_f$, with Cube representing a simpler geometry but higher cell count. Note that AST's maximum frame rate is capped at approximately 60 FPS due to *browser frame-locking*, setting the lower bound for measured frame time at 16.67 ms.

The results in Table 3 show that MCM's frame time increases significantly with #$C$ and #$B$, losing real-time performance when #$C$ reaches $32^3$. In contrast, AST maintains a frame time under 20 ms, unaffected by changes in #$C$ or #$B$. A higher #$B$ can reduce ray tracing iterations and speed up rendering. Despite using more beams than Stand and increasing MCM's frame time, Cube improves AST's performance by reducing ray-mesh intersection computations. This stability highlights AST's potential for real-time rendering of complex lattices.

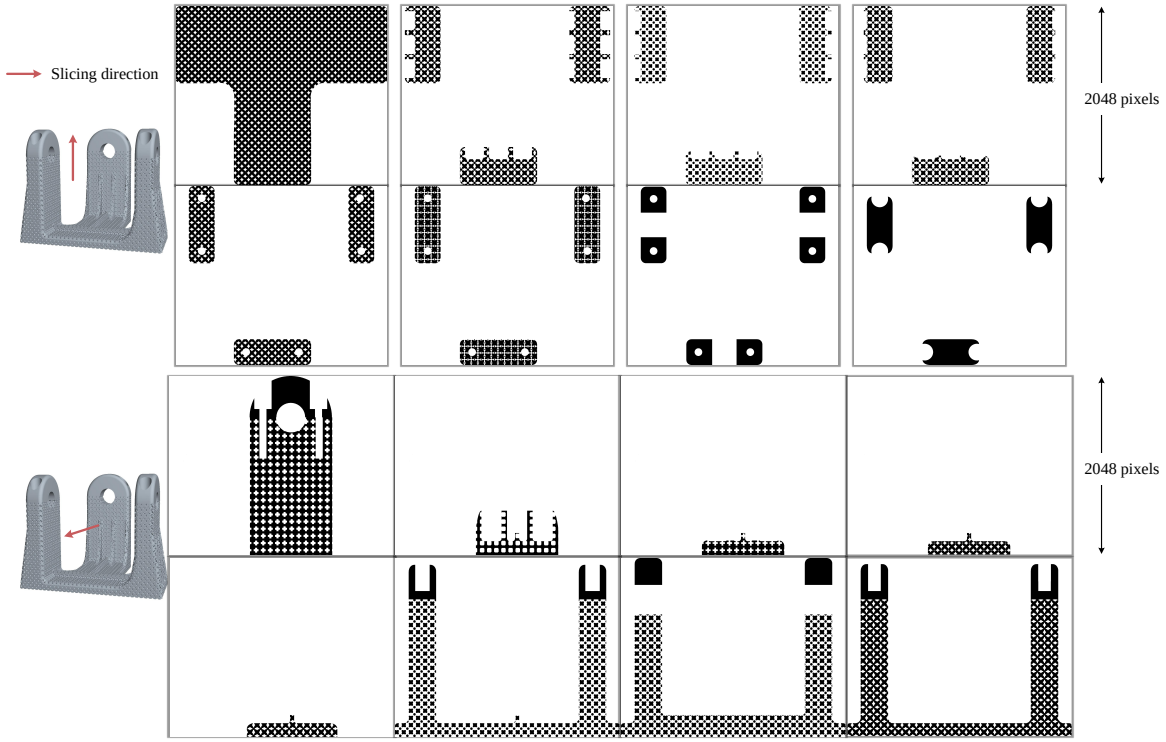



Figure 18: Eight selected slices of a partially filled stand model at 2 different directions, where the stand base is taken as $M_f$, and three stand tops as $M_s$. The slices are equally spaced along their respective direction.

| | #B | MCM | | | AST | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #C | | $8^3$ | $16^3$ | $32^3$ | $8^3$ | $16^3$ | $32^3$ | $64^3$ | $128^3$ | $256^3$ | $512^3$ | $1024^3$ | $2048^3$ |
| Stand | 12 | 15.4 | 21.2 | 166.7 | 16.9 | 17.6 | 17.9 | 18.0 | 18.3 | 18.1 | 18.3 | 17.2 | 17.3 |
| | 24 | 15.5 | 38.17 | 294.7 | 18.1 | 18.7 | 19.7 | 19.0 | 18.5 | 17.2 | 17.5 | 16.7 | 17.0 |
| | 36 | 15.6 | 60.98 | 476.2 | 18.1 | 19.7 | 18.7 | 19.4 | 18.8 | 19.48 | 19.2 | 18.1 | 18.1 |
| | 48 | 16.0 | 77.52 | 588.2 | 18.5 | 20.3 | 19.0 | 18.8 | 19.9 | 18.7 | 18.3 | 17.9 | 17.0 |
| Cube | 12 | 15.7 | 22.0 | 169.5 | 16.7 | 16.8 | 16.7 | 17.9 | 19.4 | 17.0 | 18.8 | 21.3 | 19.8 |
| | 24 | 15.9 | 42.2 | 322.6 | 16.7 | 16.7 | 16.7 | 16.9 | 17.7 | 16.7 | 26.8 | 16.7 | 16.7 |
| | 36 | 16.0 | 61.7 | 476.2 | 16.7 | 16.7 | 17.5 | 18.5 | 22.2 | 22.4 | 23.9 | 21.5 | 30.5 |
| | 48 | 16.1 | 80.6 | 625.0 | 16.7 | 16.7 | 17.3 | 19.9 | 20.4 | 31.7 | 16.7 | 16.7 | 16.7 |

Table 3: Comparisons of the frame time (ms) between AST and MCM.

### *7.3. Results and analysis*

#### *7.3.1. Rendering effects*

We demonstrate AST's rendering capabilities with a complex gear system assembly in Figure 23. The assembly is divided into three groups: the stand base and tops ($M_f$, red), a discoidal side ($M_f$, yellow), and the remaining parts ($M_s$, blue). Figures 23 (b) and (c) test the rendering effects under two conditions: (b) when various variations are set simultaneously and (c) when billions of beams are embedded.

In Figure 23 (b), the two groups have different cell types, cell sizes, and radii. Field-directed properties, internal blending, and bending are applied to the red parts. The results confirm that AST is robust and effective for rendering complex assemblies.

In Figure 23 (c), lattices with $2000^3$ cells are embedded in a bounding box, with the two $M_f$ groups occupying 4.4425% of the volume. Using the body-centered cubic type (16 beams), this results in around 5.686 billion beams. While the screen resolution is far from capturing the full complexity, a camera FoV (Field of View) of 0.15° is adequate to display the details. Note that anti-aliasing is excluded due to its high overhead and minimal improvement in detail visibility.

#### *7.3.2. Efficiency analysis*

We evaluate the computational efficiency (in terms of frame time) of the AST pipeline in rendering mega-scale shell-lattice structures with varying configurations, specifically of the lattices or the camera. Unless otherwise specified, each cell is cubic, with $\mathbf{S}_\star/r$ ranging from 5 to 10. As noted earlier, the frame time cannot go below 16.67ms.

*Cell numbers.* We measure frame time with varying cell numbers #$C$, from $1.23\times10^7$ to $6.28\times10^{10}$, at two screen resolutions: $1280\times720$ and $1920\times1080$, in Table 4. The number of triangular faces (#$F$) is fixed at 1.01K, the size of $\mathbf{B}$ (#$B$) is 12 (simple cubic unit cell type), and the field of view (FoV) is 30°.

The number of beams reaches approximately $7.54\times10^{11}$, making this the largest scale studied in lattice modeling to date. The tests are conducted on regular lattices (Regular), lattices with

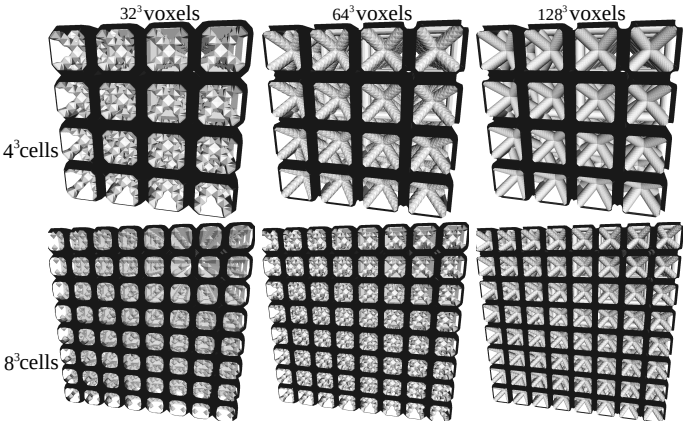



Figure 19: A cubic lattice generated and rendered using MCM, with varying cell numbers and voxel resolutions. When each cell contains only $8^3$ or $4^3$ voxels, the lattice structure shows significant fidelity loss.

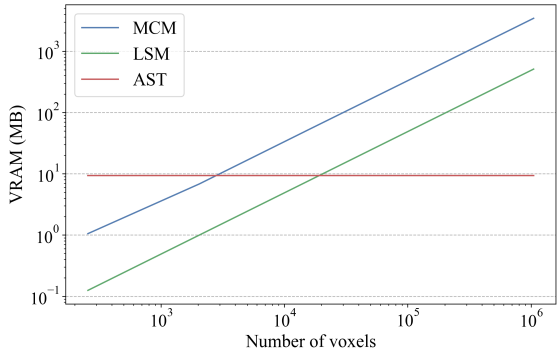



Figure 20: Comparison of VRAM overheads among AST, MCM, and LSM.

field-directed distributions and warping (Field&Warp), and lattices with internal-external blending (Blending). The FPS results demonstrate that AST's performance is largely independent of the number of cells or the variations in the lattice configuration.

| Resolution | #C | Regular | Field&Warp | Blending |
|---|---|---|---|---|
| 1280 × 720 | 12.3M | 16.69 | 16.99 | 18.43 |
| | 98.1M | 16.67 | 16.67 | 17.31 |
| | 0.785B | 16.70 | 16.69 | 16.67 |
| | 6.28B | 16.67 | 16.67 | 16.69 |
| 1920 × 1080 | 12.3M | 30.22 | 23.87 | 22.02 |
| | 98.1M | 21.49 | 21.12 | 24.58 |
| | 0.785B | 23.12 | 17.24 | 21.14 |
| | 6.28B | 18.42 | 16.67 | 17.91 |

Table 4: Frame time (ms) measured at different numbers of cells.

*Shell face numbers.* Experiments in Table 5 analyze the impact of #$F$ on frame time, with #$C$ fixed at 98.1M and a resolution of $1920 \times 1080$. The tests involve remeshing three distinct objects—cup, puppy, and foot—at scales approximately of 1K, 10K, and 100K triangular faces. As expected, the frame time shows a slight increase as #$F$ increases, showing its efficiency. The slight time increase is attributed to the increased time required to traverse the BVH. Its impact, however, is minimal due to the logarithmic time complexity, $O(\log n)$, of BVH traversal.

*Screen occupancy.* We also evaluate the frame time under varying FoVs, still using the examples in Table 5, with around 10K faces. The screen resolution is 1280×720. The results are plotted in Figure 24, which reveals an increase in frame time when the lattice occupies the entire screen at narrower FoVs.

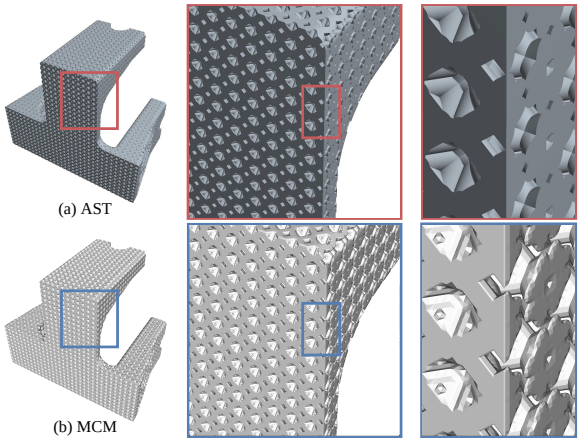



Figure 21: Comparisons between the local details of the lattices generated by AST and MCM.

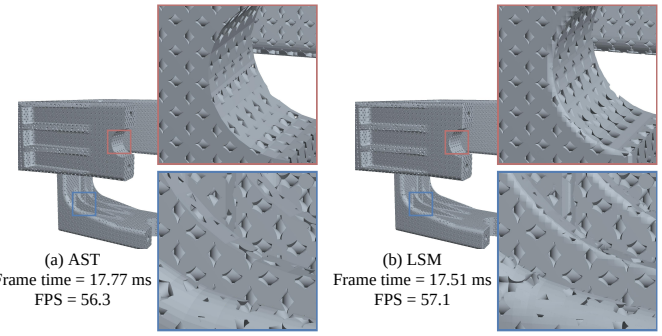



Figure 22: Comparison of AST (a) and LSM (b) in preserving mesh details, where a denser filling is used to highlight sharp edge differences. Their respective frame time/rate are also provided.

The above three results collectively demonstrate AST's capacity for real-time rendering of mega-scale lattice structures, confirming its robustness and efficiency for various industrial applications. Considering the rendering of each frame treats $C_l$ as the new input and requires no preprocessing, the frame time for editing is the same as that of rendering.

*See-through rays.* To identify bottlenecks, we analyze the impact of see-through rays by positioning the camera along the cell repetition direction with a small FoV (4°). In this setup, central rays pass through hundreds of cells without intersections. Using the body-centered cubic cell type and a $1024^2$ screen resolution, we vary the beam radius $r$ to test the effect of lattice density on frame time. The results in Table 6 show that as $\mathbf{S}_\star/r$ increases, the frame time rises significantly.

As the structure becomes sparser and see-through rays increase, many rays pass through more cells. This limits AST's applicability to sparse structures. In practice, $\mathbf{S}_\star/r > 20$ is rare due to poor mechanical properties. Appropriate camera or model rotations can further reduce such rays (Figure 25).

## 8. Conclusion and future work

The AST framework is introduced for editing mega-scale periodic shell-lattices with arbitrary cell designs filled into a mesh shell. The lattice can be warped, with blending, field-directed properties , and region-specific cell types enabled. With the mega-scale module and heterogeneous module, our method significantly reduces computational overhead compared to traditional approaches that convert between implicit and explicit rep-

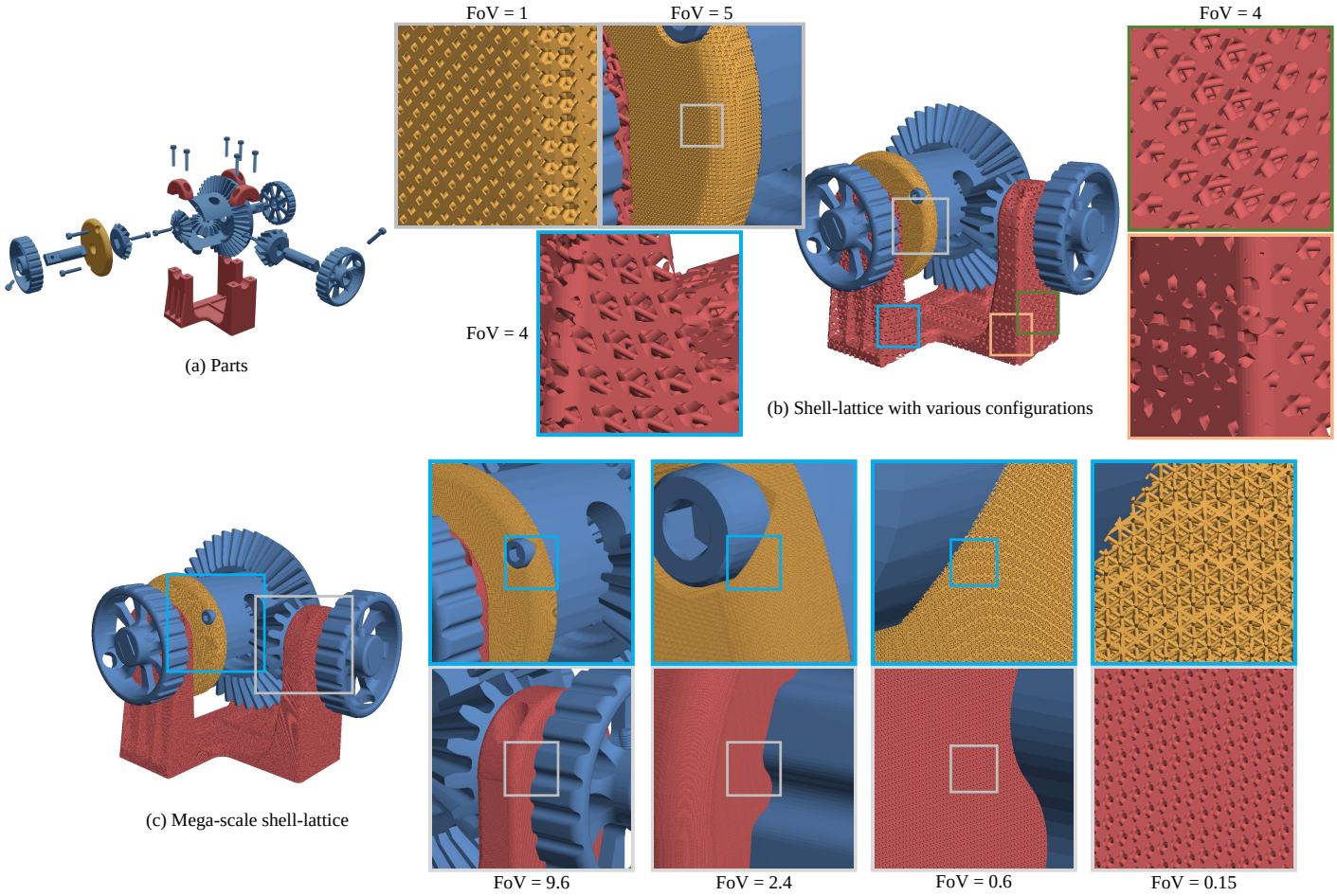



Figure 23: A gear system (a) is filled by different configurations and rendered together. In (b), the red parts take body-centered cubic as the cell type, with the radius varying along the $x$-axis, and internal blending and bending enabled. The yellow part takes octahedron as a cell type, with a smaller cell size. In (c), both parts use the body-centered cubic type and have a much smaller cell size than lattices in (b). This results in a total of approximately 5.686 billion beams. The beams remain visible as long as the camera focuses on the details (with a small FoV). All FoVs are measured in degrees.

| Model | $\#F$ | Regular | Field&Warp | Blending |
|---|---|---|---|---|
| Cup | 1014 | 21.49 | 21.12 | 24.58 |
| | 10054 | 22.58 | 21.32 | 26.10 |
| | 100032 | 22.88 | 22.38 | 26.85 |
| Puppy | 1050 | 26.48 | 19.86 | 20.71 |
| | 10228 | 26.85 | 20.34 | 22.08 |
| | 100002 | 28.51 | 20.90 | 22.12 |
| Foot | 992 | 23.82 | 18.29 | 19.39 |
| | 10420 | 24.93 | 18.63 | 19.67 |
| | 100118 | 25.84 | 19.43 | 20.76 |

Table 5: Frame time (ms) measured at different numbers of triangular faces.

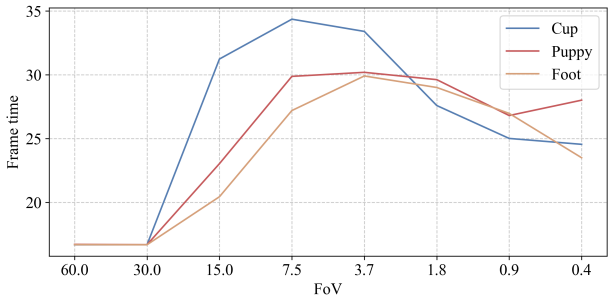


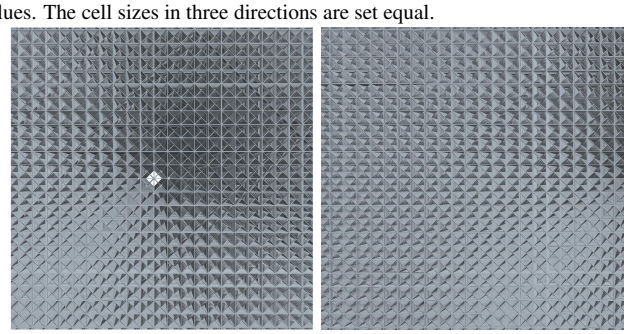

Figure 24: Frame time (ms) measured at different FoV(°).

resentations, enabling interactive editing without extensive preprocessing. Our implementation demonstrates real-time rendering of lattices with billions of beams on an RTX 3090, highlighting its potential to advance the additive manufacturing of large-scale lattices. Slicing is also achieved based on the rendering framework via adjusting rendering configurations.

Despite its efficiency in dealing with periodic lattices with warping / field-directed properties, There are still lattice types that AST can not currently cover, such as steady [7], conformal [69], and stochastic [16] lattices. These structures introduce complexities in mapping query points from lattice space to unit cell space. For instance, in conformal lattices, cells are not always hexahedral, making it challenging to map query points directly to a unit structure for repetition. Also, more advanced controls over varying radius and cell sizes—such as those determined by physical properties—have not been implemented [58]. Finally, the slicing process could be accelerated by methods that generate multiple slices simultaneously [35, 65, 66]. Developing such approaches remains one of our primary research efforts.

| $\mathbf{S}_\star/r$ | 5 | 10 | 20 | 40 |
|---|---|---|---|---|
| Runtime (ms) | 16.72 | 83.82 | 159.24 | 171.53 |
| FPS | 59.8 | 11.93 | 6.28 | 5.83 |

Table 6: Test results on the impact of see-through rays on runtime at various $\mathbf{S}_\star/r$ values. The cell sizes in three directions are set equal.



(a) Frame time = 171.53 ms FPS = 5.83

(b) Frame time = 64.02 ms FPS = 15.62

Figure 25: Frame time comparison of the same lattice at $\mathbf{S}_\star/r = 40$ from different viewpoints. Common viewpoints like (b) are less affected by see-through rays.

## ACKNOWLEDGEMENT

We would like to thank all the anonymous reviewers for their valuable comments and suggestions. The work described in this paper is supported by the NSF of China (No. 62372401).

## References

[1] Y. Zhang, F. Zhang, Z. Yan, Q. Ma, X. Li, Y. Huang, J. A. Rogers, Printing, folding and assembly methods for forming 3D mesostructures in advanced materials, Nature Reviews Materials 2 (4).

[2] S. H. Siddique, P. J. Hazell, H. Wang, J. P. Escobedo, A. A. Ameri, Lessons from nature: 3D printed bio-inspired porous structures for impact energy absorption – a review, Additive Manufacturing 58 (2022) 103051.

[3] J. Wu, O. Sigmund, J. P. Groen, Topology optimization of multi-scale structures: a review, Structural and Multidisciplinary Optimization 63 (3) (2021) 1455–1480.

[4] Y. Liu, G. Zheng, N. Letov, Y. F. Zhao, A survey of modeling and optimization methods for multi-scale heterogeneous lattice structures, Journal of Mechanical Design, Transactions of the ASME 143 (4).

[5] J. Vandenbrande, Darpa trades challenge problems, accessed: 2024-9-2. URL `solidmodeling.org/tradescp`

[6] H. Wang, D. W. Rosen, Parametric modeling method for truss structures, in: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol. 36215, 2002, pp. 759–767.

[7] A. Gupta, K. Kurzeja, J. Rossignac, G. Allen, P. S. Kumar, S. Musuvathy, Programmed-Lattice Editor and accelerated processing of parametric program-representations of steady lattices, Computer-Aided Design 113 (2019) 35–47.

[8] Y. Liu, S. Zhuo, Y. Xiao, G. Zheng, Y. F. Zhao, Rapid modeling and design optimization of multi-topology lattice structure based on unit-cell library, Journal of Mechanical Design 142 (9) (2020) 1–34.

[9] L. Chougrani, J.-P. Pernot, P. Véron, S. Abed, Lattice structure lightweight triangulation for additive manufacturing, Computer-Aided Design 90 (2017) 95–104.

[10] Q. Zou, Y. Gao, G. Luo, S. Chen, Meta-meshing and triangulating lattice structures at a large scale, Computer-Aided Design 174 (2024) 103732.

[11] A. Pasko, O. Fryazinov, T. Vilbrandt, P.-A. Fayolle, V. Adzhiev, Procedural function-based modelling of volumetric microstructures, Graphical Models 73 (5) (2011) 165–181.

[12] G. Allen, nTopology's implicit modeling technology, https://ntopology.com/resources/whitepaper-implicit-modeling-technology (2019).

[13] X. Song, B. Jüttler, Modeling and 3D object reconstruction by implicitly defined surfaces with sharp features, Computers & Graphics 33 (2009) 321–330.

[14] X. Liu, M. Meneghin, V. Shapiro, An application programming interface for multiscale shape-material modeling, Advances in Engineering Software 161 (2021) 103046.

[15] Y. Tang, G. Dong, Y. F. Zhao, A hybrid geometric modeling method for lattice structures fabricated by additive manufacturing, The International Journal of Advanced Manufacturing Technology 102 (9) (2019) 4011–4030.

[16] S. Liu, T. Liu, Q. Zou, W. Wang, E. L. Doubrovski, C. C. Wang, Memory-efficient modeling and slicing of large-scale adaptive lattice structures, Journal of Computing and Information Science in Engineering 21 (6).

[17] A. Zargarian, M. Esfahanian, J. Kadkhodapour, S. Ziaei-Rad, Numerical simulation of the fatigue behavior of additive manufactured titanium porous lattice structures, Materials Science and Engineering: C 60 (2016) 339–347.

[18] C. Bonatti, D. Mohr, Mechanical performance of additively-manufactured anisotropic and isotropic smooth shell-lattice materials: Simulations and experiments, Journal of the Mechanics and Physics of Solids 122 (2019) 1–26.

[19] O. Weeger, N. Boddeti, S.-K. Yeung, S. Kaijima, M. L. Dunn, Digital design and nonlinear simulation for additive manufacturing of soft lattice structures, Additive Manufacturing 25 (2019) 39–49.

[20] M. Li, L. Zhu, J. Li, K. Zhang, Design optimization of interconnected porous structures using extended triply periodic minimal surfaces, Journal of Computational Physics 425 (2021) 109909.

[21] W. Wang, T. Y. Wang, Z. Yang, L. Liu, X. Tong, W. Tong, J. Deng, F. Chen, X. Liu, Cost-effective printing of 3D objects with skin-frame structures, ACM Transactions on Graphics 32 (6) (2013) 1–10.

[22] L. Lu, A. Sharf, H. Zhao, Y. Wei, Q. Fan, X. Chen, Y. Savoye, C. Tu, D. Cohen-Or, B. Chen, Build-to-last: Strength to weight 3D printed objects, ACM Transactions on Graphics 33 (4) (2014) 1–10.

[23] H. Yu, J. Huang, B. Zou, W. Shao, J. Liu, Stress-constrained shell-lattice infill structural optimisation for additive manufacturing, Virtual and Physical Prototyping 15 (1) (2020) 35–48.

[24] J. Iamsamang, P. Naiyanetr, Computational method and program for generating a porous scaffold based on implicit surfaces, Computer Methods and Programs in Biomedicine 205 (2021) 106088.

[25] C. H. P. Nguyen, Y. Kim, Q. T. Do, Y. Choi, Implicit-based computer-aided design for additively manufactured functionally graded cellular structures, Journal of Computational Design and Engineering 8 (3) (2021) 813–823.

[26] J. Ding, Q. Zou, S. Qu, P. Bartolo, X. Song, C. C. Wang, STL-free design and manufacturing paradigm for high-precision powder bed fusion, CIRP Annals 70 (1) (2021) 167–170.

[27] T. A. Schaedler, W. B. Carter, Architected cellular materials, Annual Review of Materials Research 46 (1) (2016) 187–210.

[28] W. Tao, M. C. Leu, Design of lattice structure for additive manufacturing, in: 2016 International Symposium on Flexible Automation (ISFA), 2016, pp. 325–332.

[29] K. Kurzeja, J. Rossignac, Rangefinder: Accelerating ball-interference queries against steady lattices, Computer-Aided Design 112.

[30] K. Kurzeja, J. Rossignac, BeCOTS: Bent corner-operated tran-similar maps and lattices, Computer-Aided Design 129.

[31] A. Gupta, G. Allen, J. Rossignac, Exact representations and geometric queries for lattice structures with quador beams, Computer-Aided Design 115 (6058).

[32] K. Kurzeja, J. Rossignac, CTSP: CSG combinations of tran-similar two-patterns of CSG cells, Computer-Aided Design 146 (2022) 103212.

[33] C. Zong, J. Xu, J. Song, S. Chen, S. Xin, W. Wang, C. Tu, P2M: A fast solver for querying distance from point to mesh surface, ACM Transactions on Graphics 42 (4) (2023) 147:1–147:13.

[34] T. Tricard, Interval shading: using mesh shaders to generate shading intervals for volume rendering, Proceedings of the ACM on Computer Graphics and Interactive Techniques 7 (3) (2024) 43:1–43:11.

[35] S. Lefebvre, Visualizing and fabricating complex internal structures, 2017, p. 15, Technical report.

[36] F. Policarpo, M. M. Oliveira, Relief mapping of non-height-field surface details, in: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games, Association for Computing Machinery, 2006, pp. 55–62.

[37] N. Ritsche, Real-time shell space rendering of volumetric geometry, in: Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia, Association for Computing Machinery, 2006, p. 265–274.

[38] X. Liu, V. Shapiro, Multiscale shape–material modeling by composition, Computer-Aided Design 102 (2018) 194–203.

[39] W. E. Lorensen, H. E. Cline, Marching cubes: A high resolution 3d surface construction algorithm, ACM SIGGRAPH 21 (4) (1987) 163–169.

[40] L. Hao, D. Raymont, C. Yan, A. Hussein, P. Young, Design and additive manufacturing of cellular lattice structures, in: The International Conference on Advanced Research in Virtual and Rapid Prototyping (VRAP), 2011, pp. 249–254.
[41] C. Uchytil, D. Storti, A function-based approach to interactive high-precision volumetric design and fabrication, ACM Transactions on Graphics 43 (1) (2023) 3:1–3:15.
[42] J. C. Hart, Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces, Visual Computer 12 (10) (1996) 527–545.
[43] T. Plachetka, POV ray: persistence of vision parallel raytracer, in: Proc. of Spring Conf. on Computer Graphics, Budmerice, Slovakia, Vol. 123, 1998, p. 129.
[44] C. Bálint, G. Valasek, Accelerating sphere tracing, in: EG 2018 - Short Papers, The Eurographics Association, 2018, p. 4 pages.
[45] B. K. H. S. J. Korndörfer, U. G. M. Stamminger, B. Keinert, Enhanced sphere tracing, STAG: Smart Tools & Apps for Graphics 8 (4) (2014) 1–8.
[46] E. Dyllong, C. Grimm, A reliable extended octree representation of CSG objects with an adaptive subdivision depth, in: Parallel Processing and Applied Mathematics, Springer, Berlin, Heidelberg, 2008, pp. 1341–1350.
[47] O. Gourmel, A. Pajot, M. Paulin, L. Barthe, P. Poulin, Fitted BVH for fast raytracing of metaballs, Computer Graphics Forum 29 (2) (2010) 281–288.
[48] H. Grasberger, J.-L. Duprat, B. Wyvill, P. Lalonde, J. Rossignac, Efficient data-parallel tree-traversal for blobtrees, Computer-Aided Design 70 (2016) 171–181.
[49] M. J. Keeter, Massively parallel rendering of complex closed-form implicit surfaces, ACM Transactions on Graphics 39 (4) (2020) 141:1–10.
[50] C. Zanni, Synchronized-tracing of implicit surfaces, ArXiv abs/2304.09673.
[51] S. M. Rubin, T. Whitted, A 3-dimensional representation for fast rendering of complex scenes, in: Proceedings of the 7th annual conference on Computer graphics and interactive techniques, Association for Computing Machinery, New York, NY, USA, 1980, pp. 110–116.
[52] J. Amanatides, A. Woo, A fast voxel traversal algorithm for ray tracing, in: Eurographics, 1987.
[53] I. Quilez, Domain repetition, accessed: 2024-12-03.
URL `https://iquilezles.org/articles/sdfrepetition/`
[54] K.-M. Park, K.-S. Min, Y.-S. Roh, Design optimization of lattice structures under compression: Study of unit cell types and cell arrangements, Materials 15 (1) (2022) 97.
[55] M. Li, J. Hu, W. Chen, W. Kong, J. Huang, Explicit topology optimization of voronoi foams, IEEE Transactions on Visualization and Computer Graphics (2024) 1–16.
[56] G. Kreisselmeier, R. Steinhauser, Systematic control design by optimizing a vector performance index, IFAC Proceedings Volumes 12 (7) (1979) 113–117.
[57] D. Montoya-Zapata, A. Moreno, J. Pareja-Corcho, J. Posada, O. Ruiz-Salguero, Density-sensitive implicit functions using sub-voxel sampling in additive manufacturing, Metals 9 (12) (2019) 1293.
[58] J. Panetta, A. Rahimian, D. Zorin, Worst-case stress relief for microstructures, ACM Transactions on Graphics 36 (4) (2017) 1–16.
[59] E. Pujol, A. Chica, Triangle influence supersets for fast distance computation, Computer Graphics Forum 42 (6) (2023) e14861.
[60] Q. Zhou, A. Jacobson, Thingi10k: A dataset of 10, 000 3d-printing models, ArXiv abs/1605.04797.
[61] G. Elber, Precise construction of micro-structures and porous geometry via functional composition, in: Mathematical Methods for Curves and Surfaces, Springer International Publishing, Cham, 2017, pp. 108–125.
[62] A. H. Barr, Global and local deformations of solid primitives, SIGGRAPH Computer Graphics 18 (3) (1984) 21–30.
[63] B. Wyvill, A. Guy, E. Galin, Extending the CSG tree. warping, blending and boolean operations in an implicit surface modeling system, Computer Graphics Forum 18 (2) (1999) 149–158.
[64] D. Seyb, A. Jacobson, D. Nowrouzezahrai, W. Jarosz, Non-linear sphere tracing for rendering deformed signed distance fields, ACM Transactions on Graphics 38 (6) (2019) 229:1–229:12.
[65] E. Maltsev, D. Popov, S. Chugunov, A. Pasko, I. Akhatov, An accelerated slicing algorithm for Frep models, Applied Sciences 11 (15) (2021) 6767.
[66] M. Aydinlilar, C. Zanni, Transparent Rendering and Slicing of Integral Surfaces Using Per-primitive Interval Arithmetic, in: Eurographics 2022 - Short Papers, 2022, pp. 37–40.
[67] Webgpu, accessed: 2023-07-21 (2023).
URL `gpuweb.github.io/gpuweb/`
[68] Cuda samples, accessed: 2024-04-02 (2018).
URL `github.com/NVIDIA/cuda-samples`
[69] Q. Y. Hong, G. Elber, M.-S. Kim, Implicit functionally graded conforming microstructures, Computer-Aided Design 162 (2023) 103548.