The Construction of Conforming-to-shape Truss Lattice Structures via 3D Sphere Packing

Boris van Sosin[1,*], Daniil Rodin[1], Hanna Sliusarenko[2], Michael Bartoň[2,3], Gershon Elber[1]

Abstract

Truss lattices are common in a wide variety of engineering applications, due to their high ratio of strength versus relative density. They are used both as the interior support for other structures, and as structures on their own. Using 3D sphere packing, we propose a set of methods for generating truss lattices that fill the interior of B-rep models, polygonal or (trimmed) NURBS based, of arbitrary shape. Once the packing of the spheres has been established, beams between the centers of adjacent spheres are constructed, as spline based B-rep geometry. We also demonstrate additional capabilities of our methods, including connecting the truss lattice to (a shell of) the B-rep model, as well as constructing a tensor-product trivariate volumetric representation of the truss lattice - an important step towards direct compatibility for analysis.

Keywords: B-spline container, free-form microstructures, volumetric representation

1. Introduction and previous work

Lattice structures are ubiquitous in engineering, as they appear in plenty of objects of various scales: starting from large trusses in bridges and towers, over medium-sized light-weight metal parts of motorbikes and bicycles, up to microstructures in nano scale such as nanofibers [1, 2, 3]. Lattice structures can be regarded as a composite material, consisting of *solid materials* and *voids* [4], and the *relative density,* $\bar{\rho}$, of a lattice structure is the ratio between the volume occupied by the solid material and the total volume of the object (including the void space), $\bar{\rho} = \frac{V_{material}}{V_{total}}$. A *truss lattice* is a lattice whose infrastructural design consists of interlaced beams that are mutually cross-linked to provide higher stability and strength to a structure. Therefore, truss lattices are an attractive choice for a wide variety of engineering applications due to a high strength compared to small relative density.

The construction of truss lattice structures poses several challenges. The layout of the nodes and the design of the beams that connect them is an important issue, as it affects both the relative density, $\bar{\rho}$, and the mechanical properties of the overall structure. The nodal connectivity of the lattice must also be considered: it must be high enough to guarantee the rigidity of the lattice structure, but if the connectivity is too high, it may needlessly increase $\bar{\rho}$ [5]. Furthermore, the lattice may be required to support some other connected structure(s) of an arbitrary shape, and therefore the geometry that connects them must also be part of the overall considerations. In this paper, we address these issues by introducing truss lattice structures raising from sphere packing.

1.1. The sphere packing problem and its relation to lattice structures

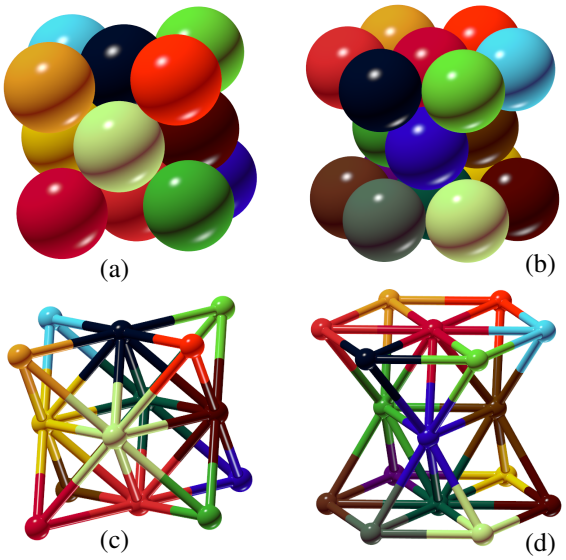

Figure 1: The two optimal patterns of sphere packing (top) and the corresponding truss lattices (bottom) which result from connecting the centers of adjacent spheres. (a) Fcc packing. (b) Hcp packing. (c) The corresponding octet cell, and (d) the corresponding hexagonal cell.

The lattice structures are closely linked to sphere packing. One finds this tight bond in nature where the hexagonal pattern appears in honeycombs [6] and the underlying lattice structure, consisting of equilateral triangles, in fact determines the shape of the honeycomb. That is, there is a certain *duality* relation between the tightly (with tangential contact) packed spheres and the lattice structure with vertices being the sphere centers and edges being the center connectors.

Two patterns of packing spheres of equal radius in 3D space, which achieve maximum packing density are *face-centered cubic* (fcc) packing, and *hexagonal close packing* (hcp) [7], see

*Corresponding author. E-mail address: sosin@cs.technion.ac.il
[1]Computer Science Department, Technion – Israel Institute of Technology, Haifa, Israel
[2]BCAM – Basque Center for Applied Mathematics, Bilbao, Basque-Country, Spain
[3]Ikerbasque – Basque Foundation for Sciences, Maria Diaz de Haro 3, 48013 Bilbao, Basque Country, Spain

Fig. 1. In an fcc cell, fourteen spheres are arranged on the vertices and on the centers of the faces of a cube. The centers of any two spheres that are in tangential contact with each other are connected by a beam. The resulting truss structure is called the *octet* cell [8]. In an hcp cell, the spheres are arranged as a triangle sandwiched between two hexagons. The hcp lattice appears frequently in nature in crystal-like structures [7].

Another simple case of using sphere packing to design truss lattices is arranging the spheres in a sheet of tetrahedral structures [9]. A tetrahedral lattice cell is formed by taking two layers of optimally-packed spheres, such as the two bottom layers of hcp, or two diagonal layers of fcc. It can therefore be regarded as a special case of both fcc and hcp. Sheets of tetrahedrons made of trusses (often called *tetrahedral-core sheets*), including non-planar variants, are used extensively in engineering and construction as support linkages and stabilizers [10, 11, 12]. We emphasize that for existing lattice systems such as [13], the beams have to be *congruent* (beams of custom-length require custom nodes which is very expensive and not supported in general). Therefore, one needs to consider sphere packing that preserves the tangential contact between the spheres, i.e., the distance between the centers of two neighboring spheres is, up to machine precision, $2r$. Note that approaches using Delaunay-like triangulations correspond to overlapping spheres and do not match this application.

It is therefore conceivable that, given a 3D geometric model, finding a tight sphere packing for its interior, and constructing a truss lattice from that sphere packing, may result in a plausible interior support structure. In this paper, we use this duality and propose a computational framework that, to the best of our knowledge, for the first time, exploits sphere packing to construct truss lattices that fill the interior of a B-rep (smooth or discrete) geometric model. A tight sphere packing results in a lattice of octet or hexagonal cells that fills the interior parts of the model, which will support the parts of the model closer to the boundary. Octet lattices are known for mechanical strength [14], and are widely used in engineering as load-bearing structures. Herein, we strive to create such octet lattices that is conforming-to-the-shape (e.g. see Figure 2).

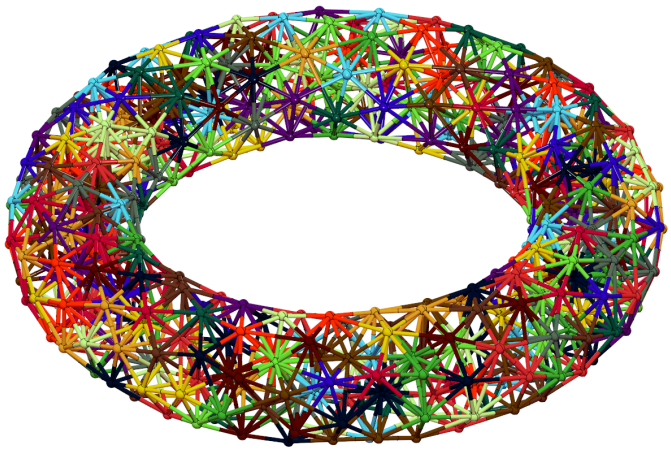

Figure 2: A conforming-to-the-shape truss construction for a torus geometry.

Our approach consists of computing an initial honeycomb sphere packing in the model, followed by an optimization cycle to include as many new spheres as possible, to finally use the duality argument to convert the sphere centers into the truss lattice, represented as either trimmed B-spline surfaces, or tensor-product B-spline trivariates.

1.2. Previous work

There are many issues in a pipeline dealing with truss lattices: starting from (i) design and their geometric representations, through (ii) physical analysis and structural mechanics, including material selection, up to (iii) the consequent fabrication. Our work is mostly related to the first class of research and therefore we will discuss mainly issues on geometric design, in this section. A 2014 survey [15] argued that, in the context of additive manufacturing, CAD tools were, at that time, ill-adapted for designed lattice structures.

Since then, several approaches to the process of designing lattice structures have been proposed. A review of lattice design methods [16], which focuses on the end-to-end process of filling a design domain (e.g. the interior of a model) with a lattice structure, categorizes these methods into two groups: *uniform*, and *conformal*. Uniform lattice approaches construct a regular lattice, with repeating elements at fixed distances from each other. The uniform lattice can be constructed without considering the boundary of the design domain, and must be restricted to fit into it. An example of a uniform-approach work is [17], which proposes to fill a geometric model with axis-aligned lattices, which then need to be clipped at the boundary of the model. Uniform-approach algorithms offer a high degree of control over the shape of the elements in the lattice, as it does not depend on the shape of the model that it fills, except near the boundary of the model, where the lattice is clipped to fit its shape. Consequently, the quality of the final result greatly depends on the initial orientation of the uniform lattice with respect to the design domain.

The conformal approach, on the other hand, attempts to construct a lattice that follows closely the shape of the design domain. This may require parameterizing the entire design domain, and using its parametrization as a deformation function for the lattice. This approach is used in [18]. A different work which attempts to construct a conformal lattice structure is [19]. It combines topological optimization with lattice construction by parameterizing the design domain using the principal stress direction, which are produced by the topological optimization process. This parametrization is then used to construct a lattice structure that conforms to the shape that results from the topological optimization. Alternatively, the design domain can be partitioned into volume cells using a meshing algorithm, and each lattice element is fitted into a cell of the mesh. This approach is used in works such as [20] and [21]. The conformal approach requires either a good volumetric parametrization, or high-quality meshing of the design domain, which is non-trivial for complex shapes. However, if employed successfully, this approach results in lattices that conform well to the boundaries of the model.

The problem of meshing is conceptually related to filling the volume of a model with a truss lattice, as both a mesh and a truss lattice consist of graph nodes with positions in Euclidean space, and edges (or beams) that connect them. The goal of meshing algorithms is, given a 2D (3D) domain, to find a triangular (tetrahedral) discretization of the domain. The requirements from the mesh depend on the application of the meshing algorithm.

Sphere packing, and its generalization, bubble packing, have been used as a tool to construct well-behaved meshes [22, 23, 24, 25]. This approach exploits the duality between the centers of packed spheres (bubbles) and a mesh. As the error of the finite element approximation is affected by the shape of the domain, regular, Delaunay-like, triangulations are preferable. Sphere packing serves to this purpose as it prevents creating degenerate triangles (tetrahedra) [22]. In this class of literature, however, the sphere arrangement allows *overlaps* and/or have small *gaps* between neighboring spheres and should not be considered

as sphere packings in the classical Kepler's sense, i.e., packing where neighboring spheres possess a *tangential contact* [26].

One of the key differences between the meshing problem and lattice structure generation is the requirement for control over the element size. In meshing algorithms, it is considered beneficial to create smaller elements near the fine details of the boundary, and smoothly transitioning into larger elements in the interior of the domain. This results in a good balance between the fitting the shape of the meshing domain and reducing the overall number of elements, thus improving the efficiency of any further algorithms that would be applied to the mesh. In lattice structure generation, however, the preferred size of the lattice elements is typically dictated by physical performance and manufacturing requirements, and globally reducing the number of elements is not necessarily desirable.

Our work looks strictly for sphere packing with tangential contact and is therefore more related to [27, 26]. While [26] considers spherical packings in unbounded domains of various dimensions, the closest work to our research is [27] that looks for a packing of the maximal number of congruent spheres into a polyhedral containers in $\mathbb{R}^3$. The mathematical formulation is based on Φ-functions and is validated by examples on simple convex polytopes which can contain some polyhedral obstacles inside, such as polygonal prisms, convex polyhedral cones and dihedral angles. In contrast, our approach supports arbitrary (non-convex) *free-form* boundary. We emphasize, however, that our work is not only about sphere packing, yet about a design of truss lattice structures that comform to arbitrary free-form geometry.

In a different setting of the 2D circle packing problem, a set of circles with non-uniform radii is given, and the objective is to find or approximate the smallest possible container into which they can be packed. The container is typically circular as well, and a practical example of such a problem is finding the smallest hole through which a bundle of electric wires of different radius can fit [28]. This problem is addressed in [28, 29]; both approaches are based on generating an initial configuartion, and then iteratively improving it by shrinking the domain, and simulating shaking the circles.

Motivated by free-form architectural design that consists of congruent and repetitive elements, packing circles and spheres was applied to curved surfaces (manifolds) [30]. A concept of a circle packing mesh is introduced. In such a triangular mesh, the incircles of the triangles form a packing, i.e., the incircles of two triangles with a common edge have the same contact point on that edge. An optimization based framework is proposed to demonstrate that circle packing meshes can comfort arbitrary free-form geometry.

A representation of lattice structures based on the union of spherical nodes *quador* (quadric of revolution) beams is proposed in [31]. This representation uses closed-form expressions for all the components of the lattice, and allows to control over the profile of the beams by adjustable parameters. The connection between the beams and the nodes is guaranteed to be smooth, and symbolic expressions for the curves at which the contact occurs are provided. In [32], an ad-hoc programming language and a user graphic interface have been introduced to design truss lattices. A whole end-to-end additive manufacturing pipeline is presented, from design to the final fabrication. Both [31, 32] require the user to explicitly define the shape of the lattice, rather than fitting it to a given geometry. In contrast, our proposed work constructs a truss lattice based on a sphere packing of the interior of the model, and therefore, attempts to fit its geometry. It does not require the user to explicitly specify the structure of the lattice. Additionally, and similarly to [32], it allows control over the resulting lattice with a small number of parameters.

Another family of relevant research deals with volumetric representation in the context isogeometric analysis. Analysis-suitable volumetric parametrizations (meshes) should respect the information coming from the boundary, such as normal vectors or curvatures, and an efficient algorithm that computes such paremetrization is introduced in [33]. A volumetric parametrization that minimizes the distortion of the mapping between the domain (unit cube) and physical domain is computed using harmonic map in [34]. Our research can also be considered as a step towards volumetric representations of curved objects composed of a specific microstructure (truss lattice).

The rest of the paper is organized as follows: in Section 2, we describe our sphere packing algorithms. In Section 3, we describe the basic truss lattice construction algorithms. In Section 4, we extend the capabilities of the truss lattice construction process, in several directions, and present our results in Section 5. Finally, we conclude and discuss future directions in Section 6.

2. The sphere packing algorithms

In this section, we describe the sphere packing algorithms. The goal of each of these algorithms is to find a configuration of as-many-as-possible congruent spheres of a given radius such that the spheres lie within a given shape. Herein, we define "within a given shape" as "center of each sphere lies within the shape".

Let us first define the problem formally. Given a bounded closed continuous domain $D \subset \mathbb{R}^3$ (with a surface boundary ∂D) and a sphere radius r, the algorithm tries to find a *configuration* $C = \{s_1, s_2, ..., s_n \in \mathbb{R}^3\}$ (a finite set of centers of spheres) such that $\forall s_i \in C,\ s_i \in D$, and $\forall s_i, s_j \in C,\ \left\|s_i - s_j\right\| \geqslant 2r$. We call the last two conditions *inclusion* and *non-penetration*, respectively, and we call any configuration that satisfies these two conditions a *valid* configuration.

Since we are aware of no algorithm that guarantees an optimal result, for a general freeform shape D, and even in 1D and 2D are at least as computationally complex as NP-hard [35], all three algorithms proposed here are heuristic, iteration-based, and produce increasingly better results. The first one (Section 2.1) attempts to find a rigid motion that will optimally position a given initial honeycomb configuration of packed spheres inside the domain. The second algorithm (Section 2.2) employs a naive repelling approach, which, however, produces good results. Finally, the third algorithm (Section 2.3) tries to iteratively improve an existing configuration, using simulated gravity forces. All three algorithms are parallelized. In Section 2.4, we briefly discuss how we compute the needed distances to the boundaries of D, and in Section 2.5, some implementation details are presented.

2.1. The "Rigid Body Motion" Algorithm

Our first packing optimization algorithm is based on moving the initial honeycomb layout *fully rigidly* such that the maximum number of spheres (i.e., sphere centers) lies inside D. In some cases, it might be desired to provide regular packing so the truss structure will consist solely of regular beams.

Consider an initial honeycomb layout that covers the whole D and also some exterior neighborhood, see Fig. 3. The *exterior neighborhood* of ∂D is defined as the space between ∂D and an outer offset of D, ε being the offsetting distance (ε is a parameter, set experimentally to $4r$ in our implementation). We categorize the sphere centers as *internal* (I), i.e., those that lie inside D, and *external* (E) which are those that lie in the exterior neighborhood of D.

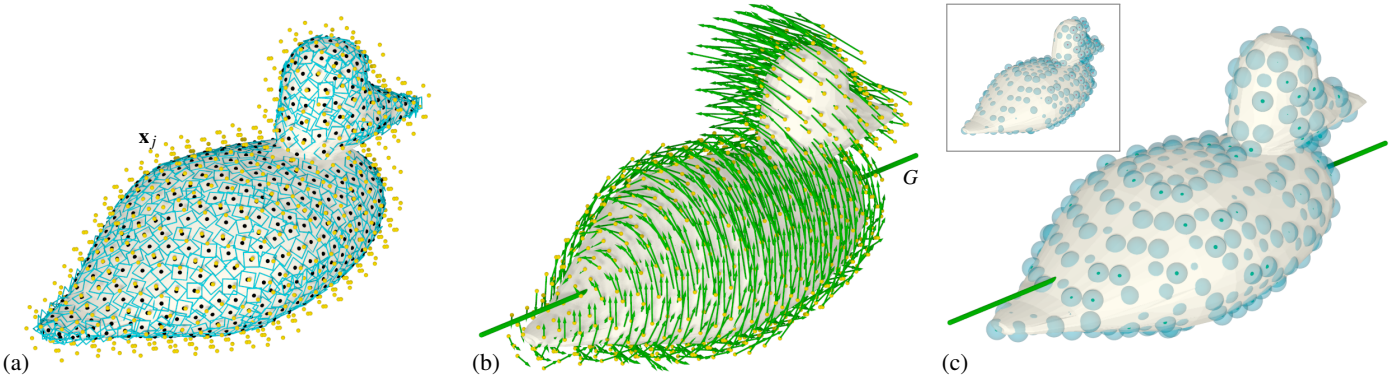

Figure 3: One iteration of "Rigid Body Motion" Algorithm. (a) The sphere centers from the exterior neighborhood (yellow) are orthogonally projected onto ∂D, resulting in a set of footpoints (black) and tangent planes (blue) of D. (b) The best instantaneous motion that moves the exterior centers towards D is computed via Eq. (4), resulting in a vector field $(\mathbf{c}, \bar{\mathbf{c}})$ (green); for visualization purposes, the vector field is scaled by factor of 60. This vector field corresponds to a helical motion with an axis G, whose Plücker coordinates $(\mathbf{g}, \bar{\mathbf{g}})$ are computed via Eq. (5). (c) The optimized honeycomb layout of the spheres (with all centers inside D). The position of the spheres before the optimization is shown top framed.

We look for a rigid body motion that moves the initial honeycomb layout such that as-many-as-possible sphere centers are located inside D. This is formulated as an optimization problem as follows. We consider the first order approximation of the rigid body motion, an *instantaneous motion* (aka a screw), defined by its projective[4] coordinates $(\mathbf{c}, \bar{\mathbf{c}}) \in \mathbb{P}^6$. Then an instantaneous velocity vector $\mathbf{v}(\mathbf{x})$ of a point $\mathbf{x} \in \mathbb{R}^3$ can be computed as

$$\mathbf{v}(\mathbf{x}) = \bar{\mathbf{c}} + \mathbf{c} \times \mathbf{x}, \tag{1}$$

see [36] for more details.

There are two main objectives: (i) to move as many as possible external spheres into D while (ii) keeping as many as possible internal spheres inside. To achieve (i), we require that the external sphere centers move towards ∂D, where the boundary is, for every point $\mathbf{x}_j$, represented by a tangent plane of D at its closest point (aka footpoint) $\mathbf{x}_j^{\perp}$. This is formulated as

$$F_E(\mathbf{c}, \bar{\mathbf{c}}) = \sum_{j=1}^{m} \left(\langle \mathbf{x}_j + \mathbf{c} \times \mathbf{x}_j + \bar{\mathbf{c}}, \mathbf{n}_j \rangle + d_j \right)^2, \tag{2}$$

where $d_j = -\langle \mathbf{x}_j^{\perp}, \mathbf{n}_j \rangle$, $<,>$ is the scalar product, and $\mathbf{n}_j$ is the boundary normal at the footpoint $\mathbf{x}_j^{\perp}$, see Fig. 4.

The second objective that the internal sphere centers stay inside D is represented by a requirement that the centers move as parallel as possible to ∂D, where the boundary is again represented by tangent planes of corresponding footpoints. This is formulated as

$$F_I(\mathbf{c}, \bar{\mathbf{c}}) = \sum_{i=1}^{n} \langle \mathbf{c} \times \mathbf{x}_i + \bar{\mathbf{c}}, \mathbf{n}_i \rangle^2. \tag{3}$$

Finally, our search for the instantaneous motion $(\mathbf{c}, \bar{\mathbf{c}})$ gives the following minimization problem

$$F(\mathbf{c}, \bar{\mathbf{c}}) = F_I(\mathbf{c}, \bar{\mathbf{c}}) + w F_E(\mathbf{c}, \bar{\mathbf{c}}), \rightarrow \min, \tag{4}$$

where the weight w controls the importance of the external centers to be brought inside D, relative to the internal points being moved inside D. In our all examples, w assumed to be in range $0.1 \leqslant w \leqslant 0.4$.

[4]The screw coordinates are points in 6-dimensional projective space $\mathbb{P}^6$, i.e., the six-tuple defines a screw up to a non-zero multiplication factor.

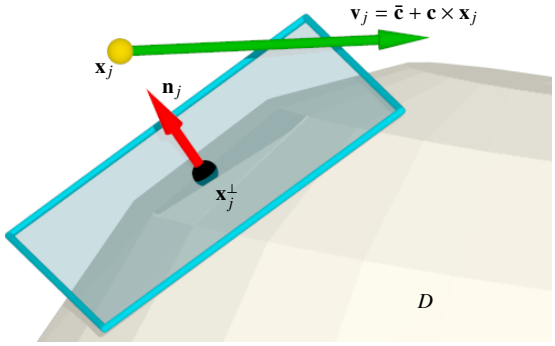

Figure 4: We look for an instantaneous motion $(\mathbf{c}, \bar{\mathbf{c}})$, that moves the external points $\mathbf{x}_j$ towards D, represented as a tangent plane (translucent) at its orthogonal projection, $\mathbf{x}_j^{\perp}$.

Eq. (4) is a linear least squares problem in $(\mathbf{c}, \bar{\mathbf{c}})$ and one therefore needs to solve only a (6×6) linear system. The solution is an instantaneous motion, that can also be interpreted as a vector field, see Fig. 3(b), that moves the centers best (in the least square sense) according to our two main objectives. We recall a known fact that moving the centers using this vector field is not sufficient as one would distort the rigidity of the honeycomb structure (as it is only the first order approximation of the rigid body motion).

To keep the structure fully rigid (i.e., to preserve that tangential contact everywhere), one needs to apply a helical motion corresponding to $(\mathbf{c}, \bar{\mathbf{c}})$. The Plücker coordinates, $(\mathbf{g}, \bar{\mathbf{g}})$, of the helical axis G are computed via

$$\mathbf{g} = \frac{\mathbf{c}}{\|\mathbf{c}\|}, \quad \bar{\mathbf{g}} = \frac{\bar{\mathbf{c}} - p\mathbf{c}}{\|\mathbf{c}\|}, \tag{5}$$

where p is the *pitch* of the helical motion, $p = \frac{\mathbf{c} \cdot \bar{\mathbf{c}}}{\|\mathbf{c}\|}$, see e.g. [37]. Finally, the desired helical motion around G is a composition of a rotation by an angle $\alpha = \arctan \|\mathbf{c}\|$ and a translation parallel to G by a distance $p\alpha$.

The whole algorithm is formalized in Algorithm 1 (and its subroutine Algorithm 2).

2.2. The "Randomization and Repulsion" Algorithm

In our second algorithm, each iteration is an attempt to find a better configuration, which is performed by filling the shape with randomly distributed spheres and letting them spread through D, under repulsion forces. The overall algorithm can be thus described as follows, while we discuss the details, in the coming sections:

Algorithm 1 Sphere packing using rigid body motion

Input:
D: The domain in $\mathbb{R}^3$ to be filled with spheres;
D_E: The exterior neighborhood of D with the offset distance ε;
$D_{tot} := D \cup D_E$;
r: The sphere radius;
Output:
The best valid configuration of spheres achieved after multiple application of the "Rigid Body Motion";
Algorithm:

1: C:= $GenerateValidHoneycomb(D_{tot}, r)$;
2: **for** $i = 1$ **to** N **do**
3: **for all** $x_i \in C$ **do**
4: C_{int}:=$GetInternalSpheres(C, r)$;
5: C_{ext}:=$C \setminus C_{int}$;
6: C_{intCl}:=$FindClosestToBoundary(C_{int}, \partial D)$;
7: C_{extCl}:=$FindClosestToBoundary(C_{ext}, \partial D)$;
8: **end for**
9: M:=$GetOptHelicalMotion(C_{intCl}, C_{extCl}, D, r)$; // Alg. 2.
10: C_{new}:=$MoveSpheres(C, M)$;
11: **end for**
12: **return** C_{new};

Algorithm 2 GetOptHelicalMotion - Optimum helical motion

Input:
D: The domain in $\mathbb{R}^3$ to be filled with spheres;
r: The sphere radius;
C: The configuration of the internal spheres;
C_{extCl}: The configuration of the closest to ∂D external spheres;
Output:
The configuration of spheres after applying a screw motion;
Algorithm:

1: $(\mathbf{c}, \bar{\mathbf{c}})$:= the screw coordinates, see Eqs.(1) – (4)
2: G := the axis of the helical motion, see Eq.(5);
3: **for all** $\mathbf{x}_i \in C \cup C_{extCl}$ **do**
4: Rotate $\mathbf{x}_i$ around G by $\alpha = \arctan \|\mathbf{c}\|$;
5: Translate $\mathbf{x}_i$ parallel to G by a distance $\frac{\mathbf{c}\cdot\bar{\mathbf{c}}}{\|\mathbf{c}\|}\alpha$;
6: **end for**
7: C: = $SpheresTranslated(D, r)$;
8: **return** C;

Algorithm 3 Sphere packing using randomization and repulsion

Input:
D: The domain in $\mathbb{R}^3$ to be filled with spheres;
r: The sphere radius;
$MaxNumRep$: The maximum number of repulsions allowed within an attempt (see Algorithm 4)
Output:
The best valid configuration of spheres achieved after multiple attempts;
Algorithm:

1: C_{best}:=$GenerateValidHoneycombFill(D, r)$; // Alg. 1
2: **while** not $Satisfied()$ **do**
3: C_{new}:=$SP1RunAttempt(|C_{best}| + 1, D, r, MaxNumRep)$; // Alg.4
4: **if** $IsValid(C_{new}, D, r)$ **then**
5: C_{best}:=C_{new};
6: **end if**
7: **end while**
8: **return** C_{best};

First, in Algorithm 3, Line 1, we generate an initial, valid honeycomb configuration, for example, using Algorithm 1. Then, in Lines 2–7, we iterate, in a search for a better configuration (one that packs more spheres) until we are *Satisfied*, which is an adjustable termination condition, in Line 2. It could mean, for example, a certain amount of time having passed. In each iteration, we *attempt* to find a configuration with a greater amount of spheres than the current best one. See the next subsection for details. The configuration resulting from each attempt can either be valid, in which case we treat it as a new best configuration, or invalid, in which case we simply continue iterating until the termination condition is met.

Algorithm 4 SP1RunAttempt - Running a single attempt

Input:
N: The number of spheres to try to fit into D on this attempt;
D: The domain in $\mathbb{R}^3$ to be filled with spheres;
r: The sphere radius;
$MaxNumRep$: The maximum number of repulsions allowed within an attempt;
Output:
The configuration of spheres resulting from this attempt;
Algorithm:

1: C:=N random sphere center points within D;
2: **for** 0 **to** $MaxNumRep$ **do**
3: C:=$SP1Repulse(C, D, r)$; // Alg. 5.
4: **if** $IsValid(C, D, r)$ **then**
5: **break**;
6: **end if**
7: **end for**
8: **return** C;

Algorithm 5 SP1Repulse - Repulsion step

Input:
C_{old}: The configuration of spheres before the repulsion;
D: The domain in $\mathbb{R}^3$ to be filled with spheres;
r: The sphere radius;
Output:
The configuration of spheres after repulsion;
Algorithm:

1: C_{new}:=ϕ;
2: **for all** $s_{old} \in C_{old}$ **do**
3: s_{new}:=s_{old};
4: **for** $s_{neighbor} \in C_{old}$ such that $\left\|s_{neighbor} - s_{old}\right\| < 2r$ **do**
5: v:=$s_{neighbor} - s_{old}$;
6: s_{new}:=$s_{new} + \alpha(2r - \|v\|)\frac{v}{\|v\|}$;
7: **end for**
8: s_{new}:=**argmin**$_{d \in D}\|d - s_{new}\|$; // make sure $s_{new} \in D$.
9: C_{new}:=$C_{new} \cup \{s_{new}\}$;
10: **end for**
11: **return** C_{new};

2.2.1. Finding a Better Configuration

The *SP1RunAttempt* function looks for a new valid configuration with N spheres. It starts with a random configuration with a desired number of spheres, which satisfies inclusion in D, but not necessarily non-penetration. Then, iterative repulsion forces are applied between the spheres, which makes the spheres spread around in the shape, aiming for the configuration in a non-penetrating state.

In Algorithm 4, in Line 1, we generate N random sphere centers within D. This can be done by generating random points within the bounding box of D and filtering out those outside D, until we find N centers. The configuration may be trapped in a local minimum, which could result in an infinite loop. Hence, in Line 2, we limit the number of repulsions by $MaxNumRep$.

In Algorithm 5, for each sphere s_{new} that penetrates other spheres (distance is less than $2r$) we move s_{new} away by a fraction of the penetration depth ($0 < \alpha < 1$). Our experiments have

Algorithm 6 Sphere packing using gravity shaking

Input:
D: The domain in $\mathbb{R}^3$ to be filled with spheres;
r: The sphere radius;
Output:
The best valid configuration of spheres achieved after multiple application of gravity;
Algorithm:

1: C:=*GenerateValidHoneycomb*(D, r);
2: **while** not *Satisfied*() **do**
3: C:=*SP2ApplyRandomGravity*(C, D, r); // Algorithm 7.
4: **end while**
5: **return** C;

Algorithm 7 SP2ApplyRandomGravity - Applying random gravity

Input:
C: The configuration of spheres to apply gravity to;
D: The domain in $\mathbb{R}^3$ to be filled with spheres;
r: The sphere radius;
Output:
The configuration of spheres after applying a gravity force once;
Algorithm:

1: $\vec{g}$:= random unit vector in $\mathbb{R}^3$;
2: **for all** $s_i \in C$ in descending order of $\langle s_i, \vec{g} \rangle$ **do**
3: *StillMoving*:=*TRUE*;
4: **while** *StillMoving* **do**
5: *StillMoving*:=*FALSE*;
6: Try move s_i in some direction $\vec{g_{pos}}$ such that $\left\langle \vec{g_{pos}}, \vec{g} \right\rangle > 0$;
7: **if** succeeded in moving s_i **then**
8: *StillMoving*:=*TRUE*;
9: **end if**
10: **end while**
11: **end for**
12: C:=*SP2TryFillFreeSpace*(C, D, r);
13: **return** C;

shown that $\alpha \cong 0.5$ is the best for convergence speed, as higher values create significant oscillations and lower values reduce the rate at which the spheres are spreading throughout D. Note that the spheres are repulsed relative to the old positions of the neighbors, which allows us to repulse each sphere in parallel. In Line 8, we make sure that the centers of the spheres never leave the shape by 'clamping' the result of the repulsion to ∂D. Clamping the position to the closest point in D allows the spheres to 'slide' along ∂D when being pushed out by their neighbors.

2.3. The "Gravity Shaking" Algorithm

Our third sphere packing optimization algorithm is based on the idea of 'shaking' an already valid configuration of spheres inside D in some preferred random directions (applying 'gravity forces' along these directions). This process is likely to create some free space on the opposite side of D that can possibly be filled with new sphere(s). Similarly to Algorithm 3, in Algorithm 6, we iterate over the configurations until we are *Satisfied*.

On each iteration of Algorithm 7, we push all the spheres along one random direction $\vec{g}$, as much as possible, in an attempt to free up some space in the opposite direction. This is a similar approach to the one used for the 2D case in [38, 28, 29].

2.4. Testing Border Conditions

For the three algorithms, described in Sections 2.1, 2.2 and 2.3, to work, we need to validate mutual non-penetration of the spheres and to compute closest points on ∂D, to ensure inclusion. Let p be the center of a sphere during the execution of our sphere packing algorithms. If D is formed by a polygonal border, this query can simply be answered by finding the closest points to p on each polygon and then choosing the closest one among them. As is shown in Section 2.5, optimizations are possible to avoid the examination of all polygons, every time.

The idea is similar for shapes formed by freeform surfaces instead of polygons: one can find the closest point on each surface patch and then choose the closest one among them. For each parametric surface patch $s(u, v)$, assuming it is C^1 continuous, there are three possibilities for the location of the closest point. The first possibility is in the interior of the surface itself, at some point $s(\hat{u}, \hat{v})$ where the tangent plane of the surface is orthogonal to the vector $p - s(\hat{u}, \hat{v})$, i.e.,

$$\left\langle p - s(\hat{u}, \hat{v}), \frac{\partial s}{\partial u}(\hat{u}, \hat{v}) \right\rangle = 0, \qquad \left\langle p - s(\hat{u}, \hat{v}), \frac{\partial s}{\partial v}(\hat{u}, \hat{v}) \right\rangle = 0. \quad (6)$$

The second possibility of distance extrema is on the boundary of the patch, formed by four curves $c_j(t)$, at some point $c_j(\hat{t})$ where the tangent of the curve is orthogonal to the vector $p - c_j(\hat{t})$, i.e.,

$$\left\langle p - c_j(\hat{t}), c_j'(\hat{t}) \right\rangle = 0, \ \ j = 1, ..., 4. \quad (7)$$

Equations (6) and (7) are solved using a solver such as [39]. The final possibility one should examine for the closest point is at the four corners of the patch.

2.5. Implementation considerations

The repulsion algorithm tries to push spheres by a fraction of their penetration depth on each step, effectively reducing the penetration depth inverse-exponentially, whenever possible. Such behavior means that reaching zero penetration depth would take infinite number of repulsions, or, when dealing with floating-point numbers, be a matter of floating-point precision errors. Since checking for zero penetration depth is fundamental in the configuration's validity check, we need to make this check with a certain degree of tolerance (epsilon). The choice of the exact tolerance is task-specific, but should be selected as high as the task allows, since increasing this tolerance would reduce the number of repulsion steps needed to accept a configuration as valid, and thus speed-up the whole process. Specifically, for truss structures, this accuracy need not be very tight.

Finding the sphere's neighbors and finding the closest point on ∂D, are two tasks that are solved numerous times, in both algorithms. Optimization of these two tasks is, therefore, of utmost importance. A naive algorithm for finding the neighbors of a single sphere would involve checking all other spheres and returning those with positive penetration depth. This naive algorithm runs in linear time, traversing all other spheres. A constant time algorithm can be implemented which involves a bounding volume hierarchy (BVH) or a partitioning of the space around D into a uniform grid with each grid cell tracking which spheres are currently inside of it. The same idea can be applied for optimizing the process of finding the closest point on ∂D. For example, if the shape is polygonal, we can pre-build a uniform grid with each grid cell knowing a small subset of polygons that is guaranteed to contain the closest point to any point within the cell.

The repulsion algorithm is parallelizable. Within each repulsion step each sphere can be processed in parallel as it does not modify any memory besides the sphere's new position. A high-performance parallel implementation is therefore possible, depending on the shape border type. Separate independent attempts can also be run in parallel, allowing a larger-scale parallelization.

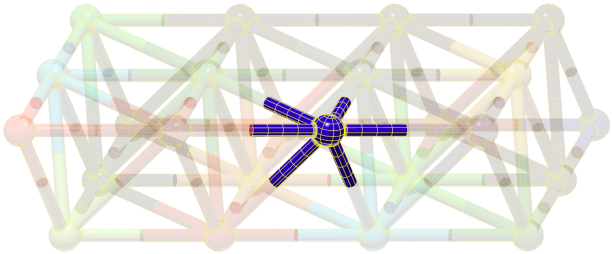

Figure 5: A node, consisting of a sphere and half-beams (in opaque dark blue), shown as part of a (translucent) truss lattice.

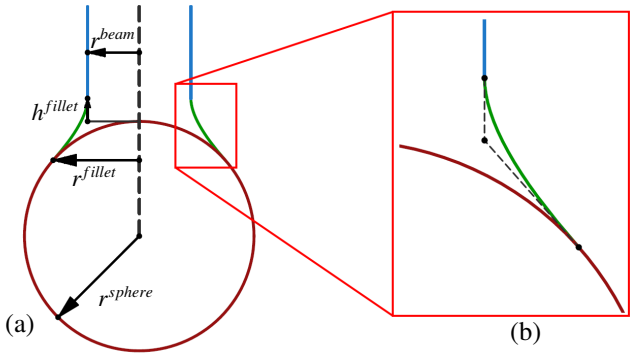

Figure 6: A cross-section of the components of a node with one half-beam. In (a), the sphere (dark red), the filleting (green), the cylindrical half-beam (light blue), and the axis of the half-beam (dashed gray) are shown. (b) A zoomed-in view of the filleting between the sphere and the half-beam. The control polygon of the filleting (dashed gray) indicates that the quadratic filleting is tangent both to the sphere and the half-beam.

The gravity algorithm can be parallelized using another approach. Whenever a new improved configuration is found (or within fixed time intervals), it is possible to run several independent instances of gravity shaking process branched from the current best one (each with its own random seed for a direction $\vec{g}$). One of these instances will probably succeed faster or better than the others, thus improving the overall performance.

Another routine that is called frequently in our rigid body motion algorithm is the computation of footpoints on a mesh ∂D and then the subsequent estimation of the normal vectors. Depending on the location of the footpoint (vertex, edge, or face), the normal vector is computed accordingly (e.g., in the case of a vertex by weighted averaging the normal vectors of the surrounding faces, the weights being the areas of the faces). This task is the bottleneck of the rigid body algorithm and was parallelized by processing every point (a sphere center) in a separate thread. Such a parallelization gave a speed-up of (almost) the number of threads. The algorithm performance could possibly be further improved by considering a better initial honeycomb configuration. Currently, the initial configuration is built without considering rotations of the honeycomb w.r.t. D.

3. Basic truss lattice construction

Once we have computed a sphere packing for the interior of the model, we need to determine the connectivity of the associated lattice, and then construct it. The lattice will be constructed from nodes consisting of a sphere (smaller than the spheres packed in Section 2, while sharing the same center), and the *half-beams* that exit from it (for an example, see Fig. 5). Each half-beam will form half the length of the connection between two adjacent sphere centers, P_i, P_j, allowing us to represent all

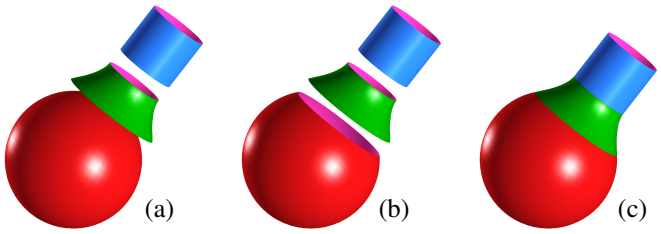

Figure 7: The three components of a node with one beam: the sphere in red, the filleting in green, the beam in blue. Magenta indicates the reverse side of the surfaces. (a) Shows an exploded view of the three components, before clipping the sphere. (b) Presents an exploded view of the components, after clipping the sphere. Finally, (c) is the assembled node.

the geometry associated with P_i independently from other points (again, for an illustration, see Fig. 5). The process for constructing the lattice is described in Algorithm 8, and the process for constructing a single node is described in Algorithm 9. These two processes depend on several parameters (see also Fig. 6):

- $d^{connect}$ - the threshold for connecting two nodes. If the sphere packing algorithm packs spheres of radius r, then we require that $d^{connect} \geq 2r$.
- r^{sphere} - the radius of the spheres at the center of the lattice nodes.
- r^{beam} - the beam radius.
- r^{fillet} - the filleting radius.
- h^{fillet} - the filleting height.

The parameters r^{fillet} and h^{fillet} together govern the computed rounding between the spheres at the centers of the nodes, and the beams in the node. We require that $r^{sphere} > r^{fillet} > r^{beam}$.

Algorithm 8 Lattice construction

Input:
$\mathcal{P} = \{P_i\}$: A set of points where the nodes will be placed (e.g., the centers of the spheres computed by a sphere packing algorithm);
$d^{connect}$: The threshold for connecting two nodes by a beam;
$(r^{sphere}, r^{beam}, r^{fillet}, h^{fillet})$: The parameters for constructing the nodes;
Output:
$\mathcal{L}$: A truss lattice, as a set of lattice nodes;
Algorithm:

```
1:  N_{P_i}:={P_j ≠ P_i | P_j ∈ P and |P_j − P_i| ≤ d^connect}; // All nodes
                                                    neighboring P_i.
2:  for all (P_i, P_j) ∈ {(P_i ∈ P, P_j ∈ P) | P_i ≠ P_j} do // Filter out
    intersecting beams.
3:      for all P_k ∈ N_{P_i}, P_m ∈ N_{P_j} do
4:          if LineSegDist(P_k − P_i, P_m − P_j) < 2r^fillet and
            |P_k − P_i| > |P_m − P_j| then
5:              N_{P_i}:=N_{P_i} − {P_k}; // Disconnect P_k from P_i.
6:              N_{P_k}:=N_{P_k} − {P_i}; // And vice-versa.
7:          end if
8:      end for
9:  end for
10: L:=φ; // The set of nodes.
11: for all P_i ∈ P do
12:     L:=L ∪ ConstructNode(P_i, r^sphere, r^beam, r^Fillet, h^fillet); // Alg. 9.
13: end for
14: return L;
```

Given a packing of spheres of radius r, two lattice nodes, P_i, P_j, are connected by a beam, if $|P_i - P_j| \leq d^{connect}$. Since for all but the most trivial models, the sphere packing is not a perfect fcc or hcp packing, we typically use $d^{connect} = 2r\sqrt{2}$. This heuristic stems from the likelihood that if four spheres are

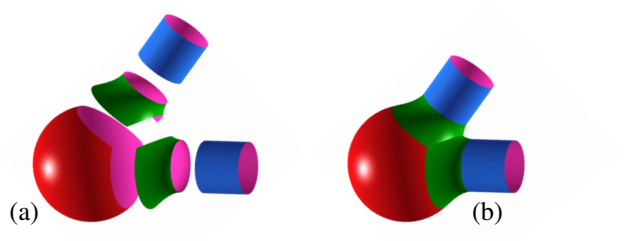

Figure 8: The clipping of two intersecting trusses. The color-coding is the same as in Fig. 7. (a) Shows an exploded view of a node with the clipped sphere and two (clipped) intersecting half-beams. (b) Is the assembled node.

arranged (approximately) in a planar quadrilateral, a beam will be formed from only one of the diagonals.

Next, we check if any of the beams (which do not share a node) intersect each other, by checking distances between two line segments in 3D (the axes of the two beams). Since the thickness of each beam is bounded by r^{fillet}, we can determine that if the distance between the axes of the beams is greater than $2r^{fillet}$, then they do not intersect. If they do intersect, we filter out the longer beams (see Algorithm 8, Lines 2–9).

With the connectivity of the truss lattice determined, all the topological information needed to construct each node unit is available. This construction process is described in Algorithm 9. Each node is constructed from a sphere S_i, and a set of half-beams $\{C_{i,k}\}$. The half-beams are constructed by creating a profile curve which consists of a filleting part which is tangent to the sphere, and a linear part, only to construct a surface of revolution from the profile curve. For an illustration of the construction of the profile curve, recall Fig. 6. The central sphere of the node is clipped by planes which form the bases of the beams, to form a trimmed surface (see Fig. 7). Each half-beam, at sphere center P_i, $C_{i,k}$, needs to be clipped at the contours of its intersections with all other half-beams $C_{i,m}$ (if such intersections exist). If the shapes $(r^{beam}, r^{fillet}, h^{fillet})$ of all the half-beams are the same, then the intersection curve is planar, and the clipping of the half-beams can be efficiently computed by finding the plane that bisects $C_{i,k}, C_{i,m}$, and clipping $C_{i,k}, C_{i,m}$ against that plane. If the shapes of the beams are not the same, the clipping must be performed by computing full surface-surface intersections. An illustration of the clipping of the intersecting beams is shown in Fig. 8.

The resulting lattice is constructed as a collection of bi-quadratic rational (trimmed) B-spline surfaces. Each node consists of a trimmed spherical surface, and half-beams that are (possibly) trimmed surfaces of revolution.

Algorithm 9 ConstructNode - Constructs a single node

Input:
P_i: The point at which the node will be constructed;
$\mathcal{N}_{P_i}$: The set of nodes neighboring P_i (see Algorithm 8);
r_i^{sphere}: The required radius of sphere i;
$r_{i,k}^{beam}$: The required radius of each beam i,k;
$r_{i,k}^{fillet}$: The required filleting radius of each beam i,k;
$h_{i,k}^{fillet}$: The required filleting height of each beam i,k;
Output:
A lattice node (consisting of a sphere $S(u,v)$, and a set of half-beams that emenate from it, $\mathcal{T}$);
Algorithm:

1: $S(u,v)$:=Sphere of radius r_i^{sphere}, centered at P_i;
2: $\mathcal{T}$:=ϕ; // The set of beams.
3: **for all** $P_j \in \mathcal{N}_{P_i}$ **do**
4: $C_{i,k}(t,r)$:= A cylinder of radius $r_{i,k}^{beam}$, height $(\frac{|P_j-P_i|}{2} - h_{i,k}^{fillet} - r_i^{sphere})$, with $(P_j - P_i)$ as its axis, ending at the point $\frac{P_i+P_j}{2}$;
5: $C_{i,k}(t,u)$:= A fillet of radius $r_{i,k}^{fillet}$ and height $h_{i,k}^{fillet}$, to be G^1-continuously joined with both $C_{i,k}(t,r)$ and S;
6: $\mathcal{T}$:=$\mathcal{T} \cup \{C_{i,k}(t,r) \cup C_{i,k}(t,u)\}$;
7: $S(u,v)$:=$Clip(S(u,v), BasePlane(C_{i,k}(t,r)))$;
8: **end for**
9: **for all** $(C_{i,k}, C_{i,m}) \in \{(C_{i,k}, C_{i,m}) \in \mathcal{T} \times \mathcal{T} \mid m \neq k\}$ **do**
10: c_{inter}:=$Intersect(C_{i,k}, C_{i,m})$; // The intersection contour.
11: **if** $c_{inter} \neq \phi$ **then**
12: $C_{i,k}$:=$Clip(C_{i,k}, c_{inter})$;
13: **end if**
14: **end for**
15: **return** $(S(u,v), \mathcal{T})$; // The clipped sphere and the set of (possibly) clipped half-beams.

4. Extensions of the construction of truss lattices

The basic algorithm for the construction of truss lattices, presented in Section 3, can be extended in several directions. We consider two such extensions here. In Section 4.1, we describe a method for connecting the boundary shell of a freeform model to a supporting truss lattice, and in Section 4.2, we adapt our algorithm to generate a tensor-product trivariate volumetric representation of the truss lattice, towards (isogeometric) analysis.

4.1. Connecting the model's shell to the truss lattice

One of the several applications of truss lattices is supporting the weight of other structures. With some simple modifications to the algorithms described in this paper, we propose a method for constructing a shell supported by a truss lattice, from a B-rep freeform model.

First, the B-rep of the *shell* must be obtained. It can be either supplied by the user, or computed from the input B-rep model. If the shell is not supplied, we compute an offset of the model, to create the interior-side of the B-rep of the shell. Given a G^1-continuous surface $S(u,v)$, we assume its normal field, $N^S(u,v)$, is oriented so that it points to the *interior* of S. The offset surface $S^{Offset}(u,v)$ is defined as $S^{Offset}(u,v) = S(u,v) + \alpha\overline{N^S}(u,v)$, where $\overline{N^S}(u,v)$ is the unit normal field, and α is the offset amount - a parameter which will govern the thickness of the shell. The computation of the precise offset surface is a complex task. It is beyond the scope of this work, as it cannot, in general, be represented as a polynomial or rational surface. Any algorithm for approximating an offset surface can be used, e.g., [40]. Then, we apply a sphere packing algorithm to the inner offset surface, and, as in the basic algorithm (described in Section 3), take the centers of the spheres, $\mathcal{P} = \{P_i\}$, as the positions of the lattice nodes. Additionally, for each $P_i \in \mathcal{P}$, we compute the nearest point Q_i on S^{Offset}. Every Q_i for which $|P_i - Q_i| < d^{connect}$ will be used to connect S^{Offset} to P_i.

The connection of the node P_i to the shell is done by intersecting S^{Offset} with a cylinder with $P_i - Q_i$ as its axis, and a radius of r^{fillet}. The cylinder is positioned to ensure that its intersection with S^{Offset} forms a closed loop. The intersection contour, c^S, forms one of the rails of the fillet, and also it is trimmed out of S^{Offset}. The other rail of the fillet is a circle, c^T, of radius r^{beam}, which is on a plane perpendicular to $P_i - Q_i$, and positioned h^{fillet} above S^{Offset} (along the $P_i - Q_i$ direction). The middle curve of this quadratic fillet is determined to ensure G^1 continuity with both S^{Offset} and the beam (which is yet to be constructed). A filleting surface is then constructed between c^S and c^T. Finally, a virtual neighbor, Q_i', is assigned to P_i at

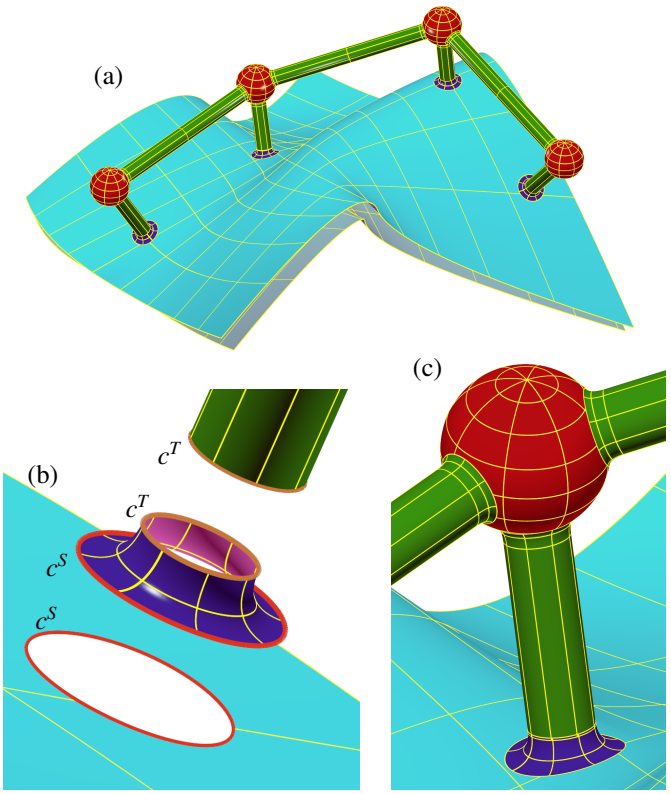

Figure 9: Three views of a configuration of four nodes (red) connected to a shell, $S - S^{Offset}$ (light blue), by beams (green). (a) The entire configuration. (b) An exploded view of the connection between the shell and a beam by a fillet (dark blue), with the fillet rails c^T (tan) and c^S (red). (c) A zoomed-in view of an assembled node, connected to the shell by filleted beams with G^1 continuity.

$Q'_i = P_i + 2(Center(c^T) - P_i)$, so that Algorithm 8 will connect the node P_i to the circle c^T by a half-beam. An illustration of the resulting filleting is shown in Fig. 9 (b), (c).

The entire process of constructing a truss lattice that is connected to a shell is detailed in Algorithm 10. An illustration of a simple set of few nodes connected to a shell is shown in Fig. 9.

4.2. *Converting the B-rep truss lattice to tensor-product trivariates*

The lattice construction processes we described until now have all used a (trimmed) B-spline surface boundary representation of the geometry. Inspired by a recent work on converting trimmed B-spline geometry to tensor-product trivariate B-spline geometry [41], we developed an algorithm for converting the lattice structure constructed by Algorithm 8, to a tensor-product trivariate B-spline representation - an important step towards being (isogeometric) analysis-ready.

Since Algorithm 8 is designed to produce a closed B-rep model with a hollowed interior, we must start by constructing a complete representation of the boundary surfaces of the node and of each half-beam. We accomplish this by constructing the interior surfaces which separate the half-beams from the sphere and from other beams, as well as the caps of the half-beams. The boundary surfaces of the node are $S(u, v)$ (defined in Algorithm 9), which is a sphere with parts of it trimmed out (see Algorithm 9, Line 7), and the bases of the half-beams. The boundary surfaces of each half-beam are $C_{i,k}$ (also defined in Algorithm 9), the base and top caps of the half-beam, and, if $C_{i,k}$ intersects with any other half-beams $C_{i,m}$, the boundary surfaces at which

Algorithm 10 Connecting the lattice to the shell

Input:
$\mathcal{P} = \{P_i\}$: A set of points where the nodes are to be placed (similarly to Algorithm 8);
$d^{connect}$: The threshold for connecting two nodes by a beam;
$(r^{sphere}, r^{beam}, r^{fillet}, h^{fillet})$: The parameters for constructing the nodes;
S^{Offset}: The B-spline boundary surface, representing the interior of the shell model;
Output:
A truss lattice;
Algorithm:

1: $\mathcal{M}$:=ϕ; // The resulting geometry.
2: **for all** $P_i \in \{P_i \in \mathcal{P} \mid Dist(P_i, S^{Offset}) \leq d^{connect}\}$ **do**
3: Q_i:=$ClosestPt(P_i, S^{Offset})$;
4: C:= A cylinder with $(P_i - Q_i)$ as its axis, centered at Q_i;
5: c^S:=$Intersect(S^{Offset}, C)$;
6: c^T:= A circle normal to $(P_i - Q_i)$, with its center positioned at a distance of h^{fillet} from Q_i along $(P_i - Q_i)$.
7: $\mathcal{M}$:=$\mathcal{M} \cup \{Fillet(c^S, c^T)\}$; // A G^1-cont. C-S^{Offset} fillet.
8: S^{Offset}:=$Trim(S^{Offset}, c^S)$;
9: $\mathcal{N}_{P_i}$:=$\{P_i + 2(Center(c^T - P_i))\}$; // The virtual neighbor, Q'_i, in the Section 4.1. $\mathcal{N}_{P_i}$ is used in Alg. 8.
10: **end for**
11: $\mathcal{M}$:=$\mathcal{M} \cup \{S^{Offset}\} \cup LatticeConstruction(\mathcal{P})$; // Alg. 8.
12: **return** $\mathcal{M}$;

$C_{i,k}$ was clipped (in Algorithm 9, Line 12). For an example of the boundary surfaces of a half-beam, see Fig. 10.

Once all boundary surfaces of the node and half-beams are constructed, they are converted (if trimmed) from trimmed B-spline surfaces to tensor-product B-spline surfaces, using an algorithm, such as the one described in [42]. Finally, we can obtain the B-spline trivariates by constructing ruled trivariates between the untrimmed surfaces that form the node (the sphere and the half-beams), and kernel points (represented as degenerate surfaces). For the surfaces of the sphere, the kernel point is the center of the sphere. Similarly, for each half-beam, we construct ruled trivariates between the boundary surfaces that form the half-beam (the half-beam from Algorithm 9, the base, the top cap, and the separating surfaces, as shown is Fig. 10) and a kernel point inside the half-beam. This kernel point is chosen near the base, to guarantee that the lines connecting the kernel point and the boundary surfaces of the half-beam are all contained inside the half-beam, ensuring that the Jacobians of the constructed trivariates are strictly positive in their interior (the trivariates can have singularities on their boundaries, which is considered tolerable in IGA applications). The overall process is detailed in Algorithm 11.

A special, but simpler, case for constructing the trivariate representations of the half-beams occurs when a half-beam does not intersect with any other half-beam. In this case, the half-beam $C_{i,k}$ is not clipped in Algorithm 9, and therefore can be constructed as a *single* ruled tensor-product trivariate between $C_{i,k}$ and a line going along the axis of the half-beam. In our experiments, tensor-product half-beam surfaces were a very common case in most of our tests. E.g., in the duck model in Fig. 11 (d), 4759 out of 12932 half-beams were tensor-products. For a comparison, see the exploded view of the node in Fig. 10(d): the two tensor-product half-beam trivariates appear in light blue, and the components of the half-beams that were trimmed and then untrimmed appear in shades of green, purple, teal and bright blue.

Algorithm 11 constructs the trivariates in such a way that they are non-overlapping, and without gaps between them. This, in

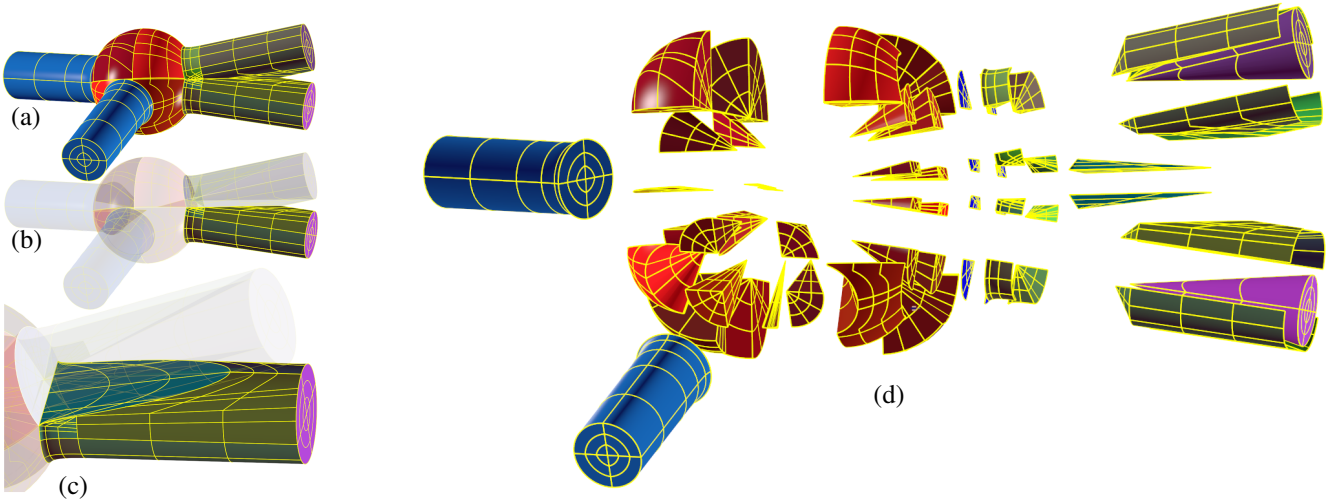

Figure 10: The boundary surfaces of a beam needed to construct a trivariate model. (a) A node with four half-beams, two of them mutually intersect (green), while the other are intersection-free (blue). The seemingly-irregular parameterization of the sphere is due to untrimming (see Algorithm 11). (b) The boundary surfaces of a beam (opaque) and (c) its zoomed-in image; the planar section separating the beam from its neighbor is shown in teal. (d) An exploded view. The components of the central node are shown in shades of red. The components of the trimmed (and untrimmed) half-beams are shown using the colors: the outer boundary of the half-beams (green), with the top caps (purple), the base caps (bright blue), the separating surfaces between the half-beams (teal), and the non-trimmed half-beams (light blue).

Algorithm 11 Trivariate node construction

Input:
$S(u,v)$: The sphere surface at the center of the node;
$\mathcal{T}$: The set of beams, as surfaces;
Output:
A lattice node, represented by compatible tensor-product trivariates;
Algorithm:

$\tilde{\mathcal{T}} := \phi$; // The trimmed surface components of all the half-beams.
for all $C_{i,k} \in \mathcal{T}$ **do**
  $\mathcal{B}_{i,k} := AllBoundarySrfs(C_{i,k})$; // A set of all the boundary surfaces of $C_{i,k}$.
  for all $C_{i,m} \in \mathcal{T} \mid m \neq k$ **do**
    $\mathcal{B}_{i,k} := Clip(\mathcal{B}_{i,k}, C_{i,m})$; // Clip $\mathcal{B}_{i,k}$ against $C_{i,m}$.
  end for
  $\tilde{\mathcal{T}} := \tilde{\mathcal{T}} \cup \{\mathcal{B}_{i,k}\}$;
end for
$\mathcal{T}^{triv} := \phi$;
// Construct all the trivariates in the half-beam:
for all $\mathcal{B}_{i,k} \in \tilde{\mathcal{T}}$ **do**
  $K := KernelPointIn(C_{i,k})$; // Find a kernel point in $C_{i,k}$.
  $\mathcal{T}^{triv} := \mathcal{T}^{triv} \cup ConstrTrivarElem(\mathcal{B}_{i,k}, K)$; // Untrim all surfaces in $\mathcal{B}_{i,k}$ (if needed), and construct ruled trivariates with K.
end for
// Construct all the boundary surfaces of the center sphere, $S(u,v)$:
$\mathcal{S}^{trim} := AllBoundarySrfs(S(u,v))$;
$\mathcal{S}^{triv} := \left\{ConstrTrivarElem(\mathcal{S}_i^{trim}, Center(S(u,v))), \forall \mathcal{S}_i^{trim} \in \mathcal{S}^{trim}\right\}$;
// Ruled trivariates between surfaces $\mathcal{S}_i^{trim}$ and sphere center.
return $\mathcal{S}^{triv} \cup \mathcal{T}^{triv}$; // The trivariates representing all the components of the node and half-beams.

addition to the guarantee that the Jacobian of the trivariates is non-negative (and strictly positive in the interior), allows precise integration operations over the entire truss lattice.

5. Results and evaluation

We have tested our proposed methods on several models, both B-spline and polygonal, and in this section we discuss the results. The graphic results are described in this section, an analysis of the quality of the resulting lattices is presented in Section 5.1 and the performance of the sphere packing algorithms is discussed in detail in Section 5.2.

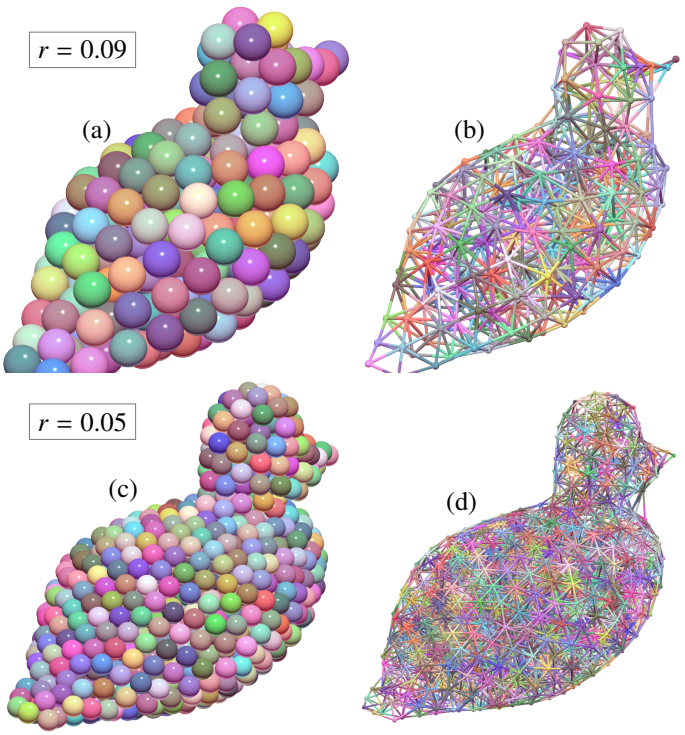

Figure 11: The results for sphere packing and truss construction for the duck model. (a) The sphere packing results with 230 spheres of $r = 0.09$ and (b) the associated truss lattice. (c,d) The analogous layout for $r = 0.05$.

First, a duck model, shown in Fig. 3, is a B-spline surface of orders 3×4 with 24×17 control points, and its outward dimensions are $2.3 \times 1 \times 1.1$. We used Algorithm 6 to produce two packing of its interior: one with 230 spheres of radius 0.09 (shown in Fig. 11 (a),(b)), and the other with 1067 spheres of radius 0.05 (shown in Fig. 11 (c),(d)). The two sphere packings demonstrate an expected result, namely that a finer packing (greater number of spheres of smaller radius) better captures the surface details of the model, such as the smooth surfaces of the head and back

of the duck. Needless to say, such control over the sphere radius has implications beyond the geometry, such as stress behavior.

Next, we created a modified version of the Stanford bunny[5] (the original Stanford bunny has holes in the bottom, making it unsuitable for sphere packing). The Stanford bunny is a polygonal model with outward dimensions of $1.7 \times 1.2 \times 1.6$. We used Algorithm 3 to pack it with spheres of radius 0.06, resulting in 885 spheres. The result is shown in Fig. 12. The bunny is a more challenging model than the duck, especially in finding a good sphere packing in narrow regions such as the ears. The truss structure, however, seems to be able to fit the geometry of the ears well, as shown in Fig. 12(e). The resulting truss lattice cannot preserve the fine surface details of the bunny. This would require using much smaller spheres, in the sphere packing algorithm. A honeycomb (regular) packing is shown in Fig. 13.

An additional model we tested is a mechanical polygonal model, with outward dimensions of $2.8 \times 1.8 \times 0.4$. This model does not have fine surface details, but its boundary is comprised of both smooth (approximately) curved surfaces and planar faces. On this model, we used Algorithm 6 with spheres of radius 0.06, resulting in 1086 spheres. The results are shown in Fig. 14. One can observe in this result, that in concave parts of the boundary, and especially near corners, some of the beams go outside the polygonal model. If we assume the mechanical model needs to fit into a space designed specifically for it, or alternatively, bolts needs to fit inside the bores (see Fig. 15 (a) and (b)), then these beams may cause the model to fail to fit correctly. To allow the user to address this issue, we offer an option to delete all beams that go outside of the boundary of the model.

We perform an approximate clipping of the beams that go outside the model, considering only the line segments that form the axes of the beams. To accomplish this, we perform two different tests. For beams which neither start nor end at spheres that lie on the boundary of the model (up to a tolerance), we simply test the line that connects the start and end points of the beam for intersection with the boundary surface. For beams which start or end at points on the boundary, we find all the contact points of the beam with the boundary surface (including tangential contact), if any. Then, for each of the segments between adjacent contact points, we determine whether the middle of the segment is inside or outside the model, using the same point-surface distance test described in Section 2.4. We show the results of removing beams that go outside the model boundary in Fig. 15 (c) and (d), respectively. Clearly, a more precise testing is feasible too. Considering the minimal distance of the center line segment of the beam to the boundary, an algebraic constraint similar to the point-surface distance test described in Section 2.4 can be built and evaluated.

Another model we present is a knot-shaped tube, with outward dimensions $1.9 \times 0.8 \times 1.9$. Its B-spline model has orders 4×4 with 10×43 control points. The results for the sphere packing and truss lattice construction are presented in Fig. 17. Similarly to the mechanical model, in the lattice that fills the knot-shaped tube there are beams that go outside the model, connecting far-apart regions. However, in particular applications that involve structural mechanics, these extra beams may be beneficial, as they could increase the structural strength of the structure. In Fig. 18, we show the knot-shaped tube with and without beams that go outside the boundary of the model.

As an additional capability, we demonstrate another feature that our framework supports, namely connecting the truss lattice to a model's shell. For this experiment, we used a sphere packing of the duck model, in which we removed all the spheres with their centers on the model boundary. Cut-out views of the results, showing half of the shell of the duck, are presented in Fig. 19.

Next, we demonstrate a truss structure with non-uniform beam radius. Recall that in Algorithm 9, each half-beam can have a different r^{beam}. We used a very coarse sphere packing of the duck model (radius of 0.17, which resulted in 46 spheres), and assigned to each beam a radius proportional to its position along the duck, back to front. This resulted in a structure with graded thickness, presented in Fig. 20. Such a degree of freedom in the truss lattice structure may have advantages in, for example, structural mechanics. Another degree of freedom to control the physical properties of the lattice structure is the relative thickness of the node versus the beam. We use the nodes of 2.5 times the thickness of the beams in our examples that is in accordance with the existing lattice systems [13].

For an additional experiment, we printed the truss lattice that results from the coarse sphere packing of the duck model, with graded-thickness beams (the same one we used in Fig. 20), in a high-end 3D printer. The results are shown in Fig. 21.

The final model we demonstrate is of a polygonal arch bridge with outward dimensions of $3 \times 0.3 \times 0.4$. The road that is supported by the bridge structure is connected to the truss lattice by passing road surface as a partial shell surface to Algorithm 10. The results are presented in Fig. 16(a-d).

In Fig. 16(e), we present an alternative solution: we constructed a truss lattice from a honeycomb sphere arrangement. The resulting truss lattice is clearly more regular.

5.1. Quality evaluation

As we discussed in Section 1.1, an ideal sphere packing results in a lattice of fcc or hcp cells, which have properties that are desirable in many engineering applications, but is not always possible inside a boundary model of general shape. To evaluate the truss lattices produced by our algorithm, we conducted a quantitative and qualitative analysis of some of our results.

The connectivity of the truss lattice is dictated by the sphere packing, and by the parameter $d^{connect}$. In fcc and hcp lattices, only spheres in tangential contact with each other are connected by beams, forming tetrahedrons. Therefore, the lengths of all beams are $2r$, where r is the radius of the spheres in the packing, and the acute angles between them are all equal to 60°. In our algorithm, we allow the user to control the maximum distance between the centers of the spheres that will be connected, via choosing the value of $d^{connect}$. Therefore, we consider a truss lattice that forms an fcc or hcp lattice in the interior of the model, but also conforms to its shape near the boundary, to be the desirable result. We also expect that a denser sphere packing with smaller spheres will better approximate the ideal result. In Figure 22, we compare the lengths and angles of the duck model packed with spheres with $r = 0.09$ (Figure 11(b)), vs. the duck packed with spheres with $r = 0.05$ (Figure 11(d)). This comparison shows that using smaller spheres in the sphere packing leads to more 60° angles between beams, which results in more tetrahedral structures. To illustrate a high-quality lattice produced by the algorithm, we show in Figure 23 the bunny model (from Figure 12) with the beams colored according to their lengths. This example shows that our algorithm does indeed construct a regular tetrahedral truss lattice in the interior of the model (the bottom of the image in Figure 23(a)), while also fitting its boundary.

Next, we present a comparison between our sphere packing approach to truss lattice construction, and a simple honeycomb

[5]Our version of the Stanford bunny was based on the model in: https://www.thingiverse.com/thing:3731/files.

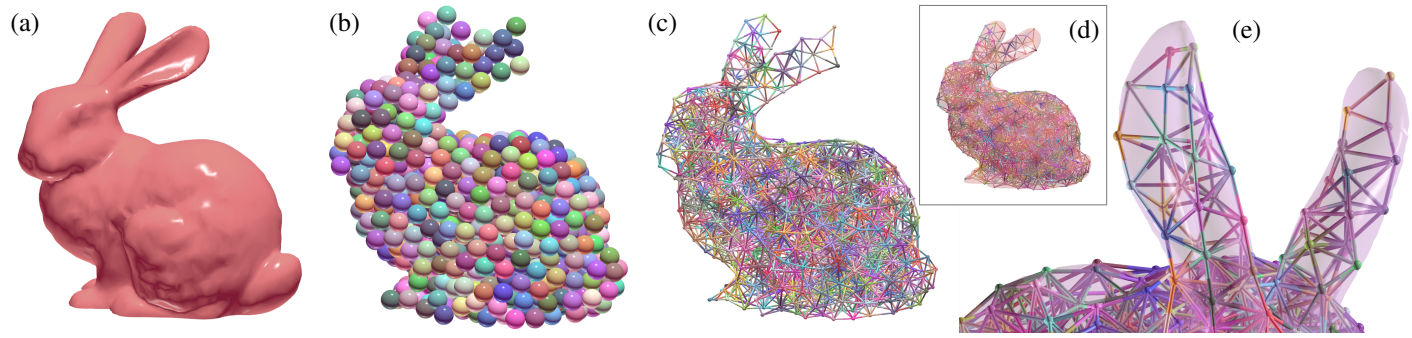

Figure 12: The Stanford bunny. (a) The input model. (b) The resulting packing and (c) the associated truss lattice. (d) The truss lattice, shown inside the translucent input model. (e) A zoomed-in view of the ears. The truss lattice manages to fit the complex geometry of the model at the ears.

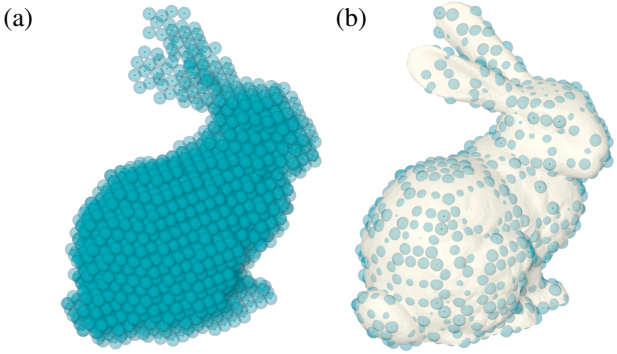

Figure 13: Honeycomb packing of the Stanford bunny. (a) The regular (honeycomb) spherical packing of 2805 spheres. (b) Interior spheres that are closer than $r = 0.04$ penetrate the boundary.

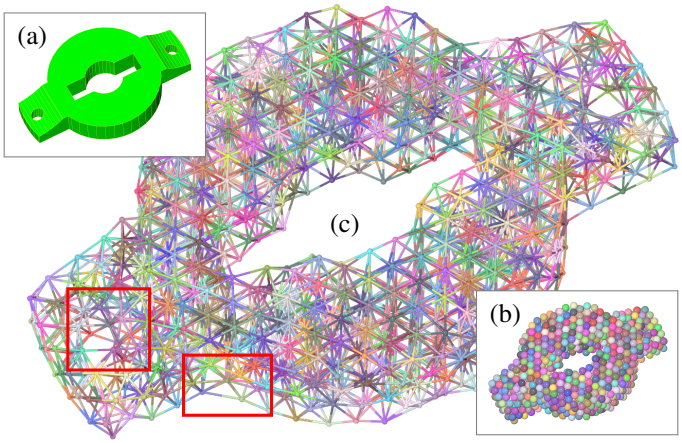

Figure 14: The mechanical model. (a) The input CAD model. (b) The resulting sphere packing and (c) the associated truss lattice. For a larger view of the marked red regions, see Fig. 15.

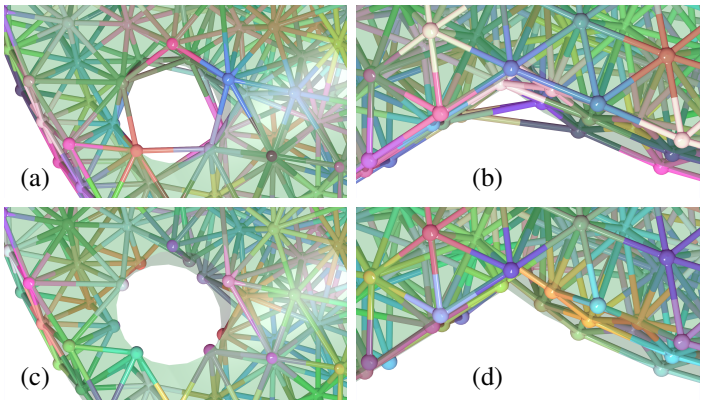

Figure 15: Zoomed-in views of the truss lattice for the mechanical model (c.f., Fig. 14). that has cavities. (a, b) The lattice shown inside the (translucent) model, with the regions where the beams go outside the boundary marked in red boxes, in Fig. 14. (c, d) The truss lattice after filtering out the beams that go outside the model's boundary.

Model	Blue (1)	Cyan (2)	Green (3)
Gravity (Fig. 24(b))	19	44	54
Honeycomb (Fig. 24(c))	12	22	77

Table 1: The numbers of tetrahedrons near the bends and in a straight section of a swept volume, for a sphere-packed truss lattice vs. a honeycomb truss lattice.

truss lattice (constructed from the honeycomb sphere packing used for the initialization of Algorithms 1, 3, and 6), clipped to the boundary. Also, we demonstrate the qualitative differences between truss lattices constructed by our algorithms and truss structures constructed via parametric deformations.

In Fig. 24, we show a swept volume of a curved cross-section (a), and the truss lattices that resulted from packing it with spheres using the gravity shaking algorithm (Alg. 6), compared to the truss lattice constructed from a honeycomb arrangement of spheres (b). Since the honeycomb arrangement does not fit the shape of the swept volume, the truss lattices has a "staircase" effect, which does not occur in the truss lattice that was constructed using sphere packing. Additionally, we computed the numbers of tetrahedrons in both lattices, at regions near the bends (highly-curved regions) of the swept volume (the box marked as (2) in Fig. 24 (b), (c)), near the boundary (the box marked as (1)), and near a relatively smooth section (the box marked as (3)). Since fcc and hcp cells are both comprised of tetrahedrons, we use the number of tetrahedrons as a heuristic approximate measure for the likelihood of fcc and hcp being formed in the truss lattice. Table 1 shows that due to the "staircase" effect, the truss lattice that was constructed via sphere packing has more tetrahedrons near the bend than the truss lattice that was constructed from the honeycomb arrangement. Near the boundary, the lattice that was constructed via sphere packing has slightly more tetrahedrons, because spheres (and therefore more lattice nodes) are packed near the curved boundary. Near the relatively smooth region of the swept volume, the number of tetrahedrons is higher in the lattice that uses the honeycomb arrangement. This is because the a honeycomb arrangement results in a maximally-dense tetrahedral mesh where it is not near the boundaries of the model.

In Fig. 25, we constructed two truss structures using parametric deformation techniques. In the first experiment, we positioned a regular grid of $4 \times 4 \times 20$ points inside the parametric space of the swept volume from Fig. 24 (a), and mapped them to Euclidean space. Then, we constructed a truss lattice using these

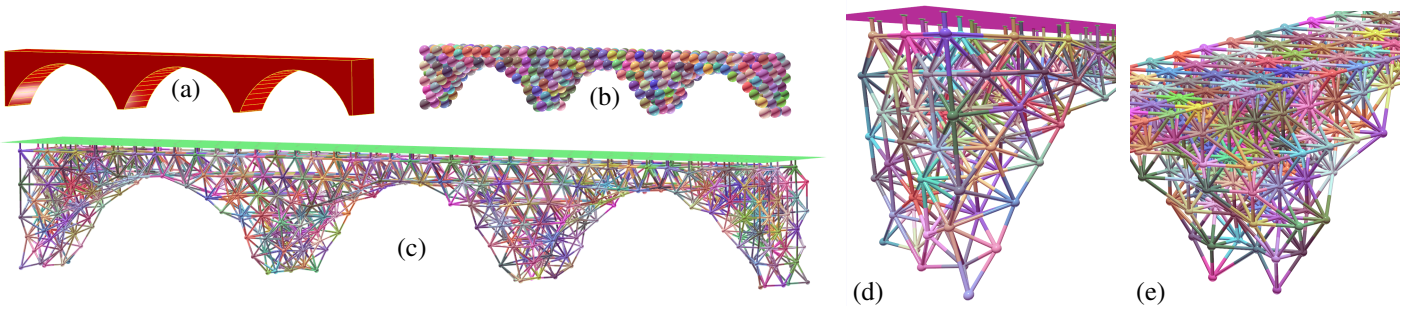

Figure 16: An arch bridge. (a) The polygonal input model. (b) The optimal sphere packing and (c) the truss lattice constructed from it. (d) A zoom-in of the optimal, yet irregular, lattice. (e) In contrast, the truss lattice constructed from the regular (honeycomb) sphere arrangement.

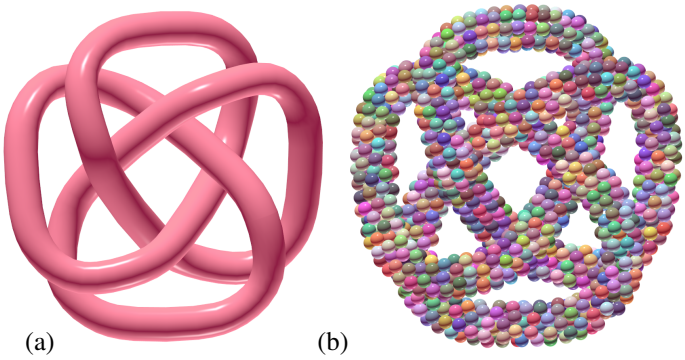

Figure 17: The knot-shaped tube model. (a) The input B-spline model. (b) The resulting sphere packing. See also Fig. 18.

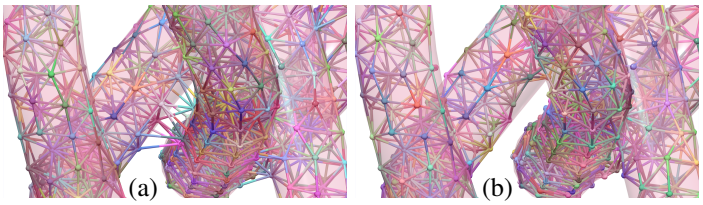

Figure 18: Zoomed-in views of the truss lattice for the knot-shaped tube, c.f. Fig. 17. (a) Due to its complex geometry, the tube (translucent) contains regions where topologically different parts are close to each other, and therefore some lattice beams go outside the tube. (b) The truss lattice after removing beams that go outside the model boundary.

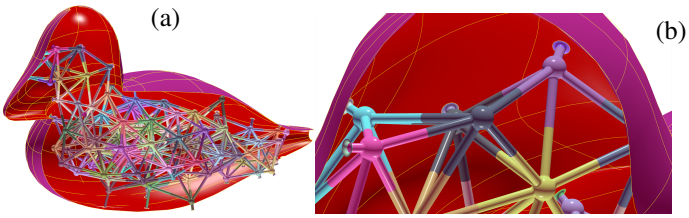

Figure 19: Connecting a truss lattice to a model's shell. (a) A cut-out view of (a low resolution) truss lattice connected to (a half of) the duck surface model. Note that some of the beams are connected to the half of the shell that is not displayed. (b) A zoomed-in view of the duck's head.

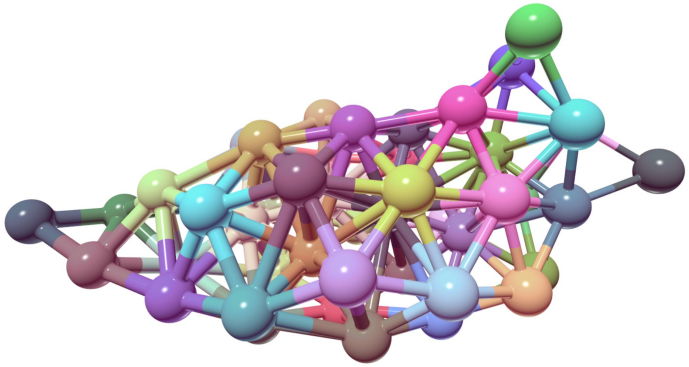

Figure 20: A truss structure with beams of graded (non-uniform) thickness, from left (thick) to right (thin), globally.

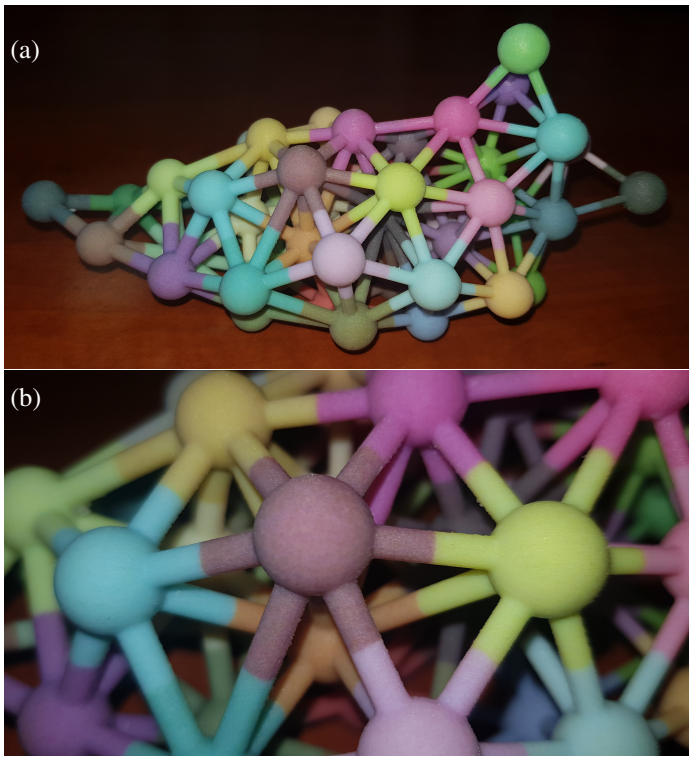

Figure 21: A 3D printed model of the simplified (46 spheres) duck, with graded-thickness beams (see Figure 20). (a) A view showing the overall shape of the duck. (b) A zoomed-in view, showing the details of the nodes. This model was printed on a Stratasys printer.

points as the lattice nodes. While the overall shape of the resulting truss lattice (Fig. 25(a)) fits the shape of the swept volume, the distances between the points are highly non-uniform. This is because the uniformly-spaced grid of points in the parametric space of the swept volume is mapped non-uniformly into Euclidean space, and as a result, in some regions the truss lattice is significantly denser than in others. This demonstrates that for all but the simplest deformation functions, controlling the positioning of the lattice node using parametric deformation is a difficult task. In Fig. 25(b), we constructed a truss lattice from a regular rectangular grid of $4 \times 4 \times 20$ cells inside the parametric space of the same swept volume. Each cell consists of a lattice node and beams that connect to the neighboring nodes, if such exist (see Fig. 25 (c)). Then applied the parametric deformation to the entire truss lattice, similarly to [18]. We, again, see that the

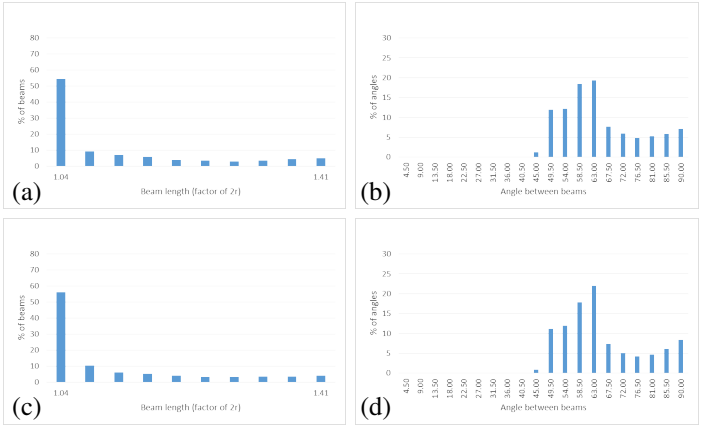

Figure 22: The distributions of the beam lengths and acute angles between beams. The beam lengths are divided into ten buckets spanning the range $[2r, d^{connect}]$. (a) The beam lengths and (b) angles of the duck model with $r = 0.09$. (c) The beam lengths and (d) angles of the duck model with $r = 0.05$.

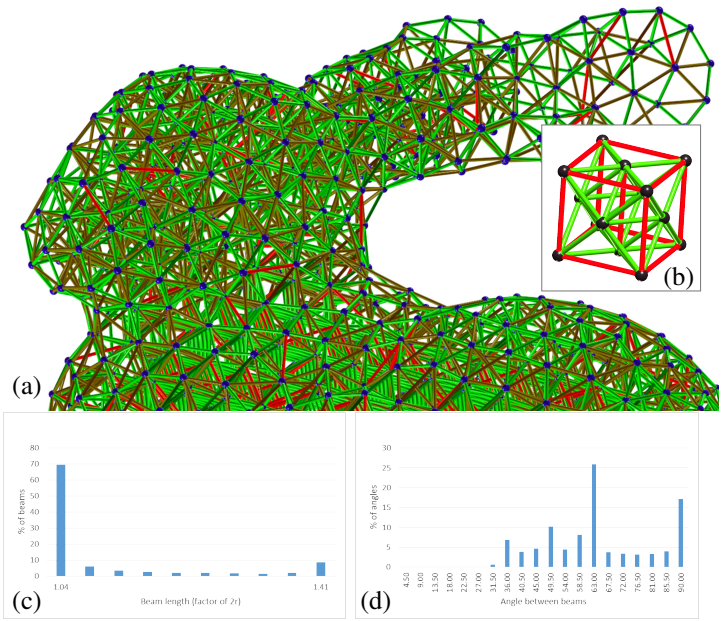

Figure 23: (a) The truss lattice constructed from the Stanford bunny, with the beams colored according to their length. Green beams correspond to the first bucket of the histogram (the first tenth of the range $[2r, d^{connect}]$), the red beams correspond to the last bucket, and the orange beams correspond to all other lengths. Note that since we used $d^{connect} = 2r\sqrt{2}$, extra beams (red) are added to the tetrahedral lattice, as in (b) the example octet cell. (c) The distributions of the beam lengths and (d) acute angles between beams.

distances between the lattice nodes are non-uniform, due to the non-uniform parametrization of the swept volume, but in this example, the lattice nodes themselves are deformed. In fact, in this example, the nodes are no longer spherical, and the beams are no longer straight. This issue has been partially addressed in [18], by proposing an adaptive subdivision of the parametric space of the deformation volume that attempts to form cells of similar size in Euclidean space. Additionally, a library of bifurcation tiles is used for connecting cells across the subdivision boundaries. However, with this strategy a certain amount of deformation is still unavoidable, and consequently, it is a less appropriate solution in cases where spherical nodes and straight beams are required (e.g. when the truss lattice needs to be load-bearing).

5.2. Performance analysis

To compare the two sphere packing algorithms, Algorithm 3 (randomization and repulsion) and 6 (gravity shaking), we measured the number of spheres they packed, starting from the same honeycomb arrangement. We performed all measurements using a 8-thread Core i7 7700K CPU. Fig. 26 shows that there is no clear winner between the two algorithms, in the general case. Algorithm 3 (shown in blue in Fig. 26) is clearly superior in case (a), where the potential number of spheres is low, especially in the short run. On the other hand, when we allow more spheres by reducing the sphere radius, Algorithm 6 (shown in orange) performs comparably in the short run and noticeably outperforms Algorithm 3, if given enough time to run. However, herein we were mostly working with cases of 1000-2000 spheres and executed each sphere packing algorithm for around 10 hours. In these particular cases, Algorithm 6 happened to always produce better final results.

As discussed in Section 2.4, our sphere packing algorithms support both polygonal and freeform surface boundaries. However, in our implementation, using freeform boundaries ended up being two to three levels of magnitude slower than their polygonal counterparts. Considering all these observations, in the result summary provided in Table 2, we mostly focus on the results achieved with the gravity shaking on polygonal models. In addition, we demonstrate the effectiveness of our algorithms on different time scales in Table 3.

6. Conclusion and future work

We have presented an end-to-end process for packing a B-rep model with spheres and a consequent construction of a truss lattice structure. Such a lattice connects the centers of the spheres with beams and fills the interior of the model. We have presented three sphere packing algorithms. Furthermore, we have proposed two extensions for the truss lattice construction algorithm: one allows connecting the truss lattice to a model's shell, and another converts the (trimmed) B-spline truss lattice to a tensor-product trivariate model. There are several directions in which our algorithms can be further explored and extended:

Isogeometric testing. While we designed our algorithm for converting the truss lattice to a trivariate model such that the generated parameterizations are (isogeometric) analysis-compatible, testing this capability is beyond the scope of this work. Analyzing the truss structures produced by our algorithm, for example, for principal stress analysis, can provide valuable insights on the properties of truss structures that result from sphere packing, and compare their performance to structures designed by human engineers, or other algorithms. We expect to explore this direction in the future.

Shape optimization. Another potential benefit of using (isogeometric) analysis in conjunction with our algorithms is optimizing the parameters controlling the shape and size of the beams and the fillets, again, for example, in the context of stress analysis. This utilizes the ability to construct beams of different thickness, that we demonstrated in our results. An analysis-optimization loop for truss structures can be a powerful tool for engineering.

Trivariate representations. Finally, a challenging extension of our algorithm is constructing a trivariate representation of a model's shell with the truss lattice connected to it. We have shown that the geometry of the nodes and beams can be converted to tensor-product trivariate B-splines. A shell of arbitrary shape and joints to beams presents a challenge. One needs to partition the shell into compatible components, that can form tensor-

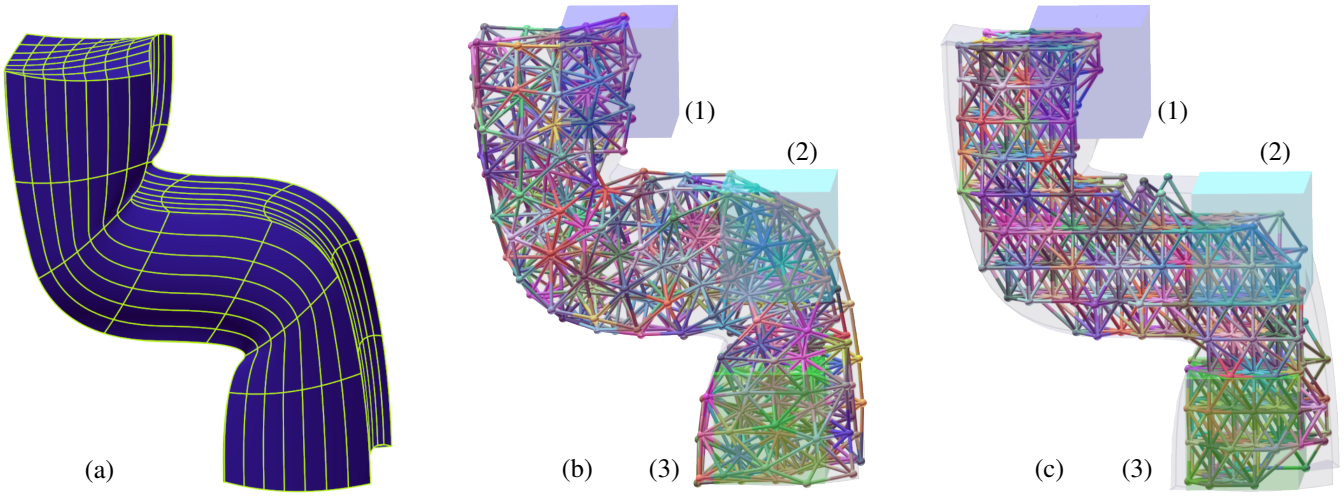

Figure 24: A comparison between truss lattices constructed using different node positioning approaches. (a) The model which was filled with the truss lattice. (b) Our sphere packing approach, using Alg. 6. (c) Regular honeycomb sphere packing. The numbers of tetrahedrons in each of the boxes numbered (1), (2), (3) are presented in Table 1.

Model	Figure	BB size	Radius	Algor.	# IniSph	# OptSph	# TrimSrf	# Trivar	Truss t	Trivar t
Duck	3(c)	$2.3 \times 1 \times 1.1$	0.05	RBM	1019	1035	12661	183596	2m4s	8m2s
Duck	11(c), (d)	$2.3 \times 1 \times 1.1$	0.05	GS	1019	1140	14072	233503	2m34s	5m15s
Duck	11(a), (b)	$2.3 \times 1 \times 1.1$	0.09	RR	170	230	2510	52937	23s	1m24s
Knot	17	$1.9 \times 0.8 \times 1.9$	0.04	GS	1034	1495	14357	303562	1m55s	11m51s
Bunny	13	$1.7 \times 1.2 \times 1.6$	0.04	RBM	2714	2805	36165	501336	6m37s	60m48s
Bunny	12	$1.7 \times 1.2 \times 1.6$	0.06	RR	817	885	10241	216024	1m43s	5m59s
Bunny		$1.7 \times 1.2 \times 1.6$	0.04	GS	2785	2990	37944	594211	7m25s	15m55s
Bridge	16	$3 \times 0.3 \times 0.4$	0.04	GS	564	666	6930	116298	56s	2m42s
CAD	14	$2.8 \times 1.8 \times 0.4$	0.06	GS	1020	1086	12576	212763	1m59s	4m55s

Table 2: Statistics on the sphere packing and truss lattice construction for various models. In the top row, BB states for the "bounding box", and the columns, in turn, show the sphere radii, algorithm applied, number of initial and optimized spheres, number of trimmed surfaces and trivariates, and finally computational times of truss and trivariate construction. In the Algorithm column, RBM indicates rigid body motion (Algorithm 1), GS indicates gravity shaking (Algorithm 6), and RR indicates randomization and repulsion (Algorithm 3).

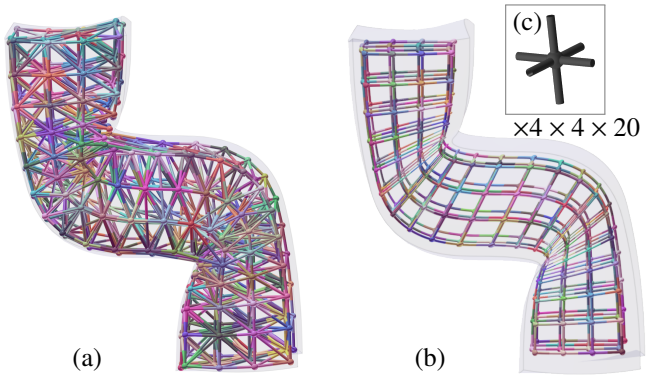

Figure 25: Constructing a truss lattice using parametric deformation techniques. Note the density of the lattice nodes near the high-curvature regions of the model. (a) A truss lattice constructed from points positioned in a regular grid in the parametric space of the swept volume (see Fig. 24 (a)), and then mapped into Euclidean space. (b) A truss lattice constructed from a regular grid of cells, inside the parametric space of the swept volume, and then deformed. (c) A single cell (before the deformation) of the truss lattice.

product trivariates with a non-negative Jacobian, and choosing kernel points accordingly.

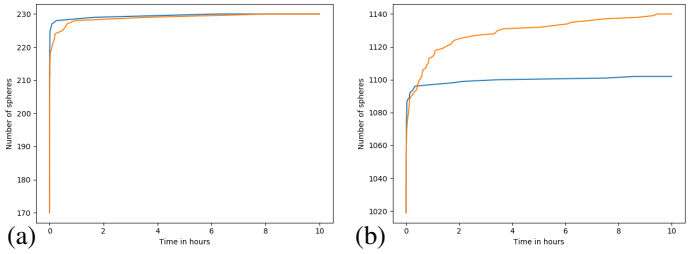

Figure 26: A comparison of the number of spheres packed by the randomization and repulsion algorithm (blue), versus the gravity shaking (orange), for the duck model over 10 hours. (a) $r = 0.09$. (b) $r = 0.05$.

Acknowledgment

This research was supported in part by the ISRAEL SCIENCE FOUNDATION (grant No. 597/18) and in part by the European Union Horizon 2020 research and innovation programme, under grant agreement No 862025.

The third author has been partially funded by the BCAM Severo Ochoa accreditation of excellence, Spain (SEV-2017-0718), and Lantek.

The fourth has been supported by Spanish Ministry of Science, Innovation and Universities: Ramón y Cajal with reference RYC-2017-22649.

Model	Radius	Algor.	Initial	1s	10s	1m	10m	1h	10h
Duck	0.05	GS	1019	1039	1050	1063	1089	1114	1140
Duck	0.09	RR	170	181	206	222	227	228	230
Knot	0.04	GS	1034	—	1309	1375	1417	1455	1495
Bunny	0.06	RR	817	—	825	830	870	878	885
Bunny	0.04	GS	2785	—	2828	2846	2881	2923	2990
Bridge	0.04	GS	564	594	612	625	641	658	666
CAD	0.06	GS	1020	1024	1030	1041	1057	1075	1086

Table 3: Statistics of time-effectiveness of the sphere packing algorithms, shown as the number of spheres achieved within a given time-frame. The models and algorithms are the same as in Table 2. Missing 1-second data in some cases means that one iteration of the algorithm takes longer than one second for that model and radius.

References

[1] L. Gil, A. Andreu, Shape and cross-section optimisation of a truss structure, Computers & Structures 79 (7) (2001) 681–689.

[2] V. R. Gervasi, D. C. Stahl, Design and fabrication of components with optimized lattice microstructures, in: Proceedings of the Solid Freeform Fabrication Symposium, Austin, TX, 2004, pp. 838–844.

[3] C. Chu, G. Graf, D. W. Rosen, Design for additive manufacturing of cellular structures, Computer-Aided Design and Applications 5 (5) (2008) 686–696.

[4] N. A. Fleck, An overview of the mechanical properties of foams and periodic lattice materials, Cell Met Polym (2004) 3–7.

[5] S. Pellegrino, C. R. Calladine, Matrix analysis of statically and kinematically indeterminate frameworks, International Journal of Solids and Structures 22 (4) (1986) 409–428.

[6] F. Nazzi, The hexagonal shape of the honeycomb cells depends on the construction behavior of bees, Scientific reports 6 (2016) 28341.

[7] J. H. Conway, N. J. A. Sloane, Sphere packings, lattices and groups, Springer Science & Business Media, 2013, Ch. 1, pp. 2,8.

[8] R. B. Fuller, Synergetics: explorations in the geometry of thinking, Estate of R. Buckminster Fuller, 1982, Ch. 420, pp. 138–140.

[9] H. G. Allen, Analysis and design of structural sandwich panels: the commonwealth and international library: structures and solid body mechanics division, Elsevier, 2013.

[10] N. Wicks, J. W. Hutchinson, Optimal truss plates, International Journal of Solids and Structures 38 (30-31) (2001) 5165–5183.

[11] F. W. Zok, S. A. Waltner, Z. Wei, H. J. Rathbun, R. M. McMeeking, A. G. Evans, A protocol for characterizing the structural performance of metallic sandwich panels: application to pyramidal truss cores, International Journal of Solids and Structures 41 (22-23) (2004) 6249–6271.

[12] G. Zhang, L. Ma, B. Wang, L. Wu, Mechanical behaviour of cfrp sandwich structures with tetrahedral lattice truss cores, Composites Part B: Engineering 43 (2) (2012) 471–476.

[13] SYMA systems, SYMA-orbit, the space-truss, `https://www.syma.com/en/systems/inspiring-systems/syma-orbit`.

[14] V. S. Deshpande, N. A. Fleck, M. F. Ashby, Effective properties of the octet-truss lattice material, Journal of the Mechanics and Physics of Solids 49 (8) (2001) 1747–1769.

[15] A. Hadi, F. Vignat, F. Villeneuve, Evaluating current cad tools performances in the context of design for additive manufacturing, in: Proceedings of Joint Conference on Mechanical, Design Engineering & Advanced Manufacturing, 2014.

[16] F. Tamburrino, S. Graziosi, M. Bordegoni, The design process of additively manufactured mesoscale lattice structures: a review, Journal of Computing and Information Science in Engineering 18 (4).

[17] S. R. Musuvathy, Method for creating three dimensional lattice structures in computer-aided design models for additive manufacturing, US Patent 9,902,114 (February 2018).

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

[19] J. Wu, W. Wang, X. Gao, Design and optimization of conforming lattice structures, IEEE Transactions on Visualization and Computer Graphics.

[20] S. Engelbrecht, L. Folgar, D. W. Rosen, G. Schulberger, J. Williams, et al., Cellular structures for optimal performance, in: Proc. SFF Symposium. Austin, 2009, pp. 831–842.

[21] J. Nguyen, S.-I. Park, D. W. Rosen, L. Folgar, J. Williams, Conformal lattice structure design and fabrication, in: Solid Freeform Fabrication Symposium, Austin, TX, 2012, pp. 138–161.

[22] K. Shimada, D. C. Gossard, Bubble mesh: automated triangular meshing of non-manifold geometry by sphere packing, in: Proceedings of the third ACM symposium on Solid modeling and applications, 1995, pp. 409–419.

[23] S. Yamakawa, K. Shimada, Anisotropic tetrahedral meshing via bubble packing and advancing front, International Journal for Numerical Methods in Engineering 57 (13) (2003) 1923–1942.

[24] J. Liu, S. Li, Y. Chen, A fast and practical method to pack spheres for mesh generation, Acta Mechanica Sinica 24 (4) (2008) 439–447.

[25] S. Lo, W. Wang, Generation of tetrahedral mesh of variable element size by sphere packing over an unbounded 3d domain, Computer methods in applied mechanics and engineering 194 (48-49) (2005) 5002–5018.

[26] J. H. Conway, C. Goodman-Strauss, N. J. Sloane, Recent progress in sphere packing, Current Developments in Mathematics 1999 (1) (1999) 37–76.

[27] Y. Stoyan, G. Yaskov, Packing congruent spheres into a multi-connected polyhedral domain, International Transactions in Operational Research 20 (1) (2013) 79–99.

[28] K. Sugihara, M. Sawai, H. Sano, D.-S. Kim, D. Kim, Disk packing for the estimation of the size of a wire bundle, Japan Journal of Industrial and Applied Mathematics 21 (3) (2004) 259–278.

[29] J. Ryu, M. Lee, D. Kim, J. Kallrath, K. Sugihara, D.-S. Kim, Voropack-d: Real-time disk packing algorithm using voronoi diagram, Applied Mathematics and Computation 375 (2020) 125076.

[30] A. Schiftner, M. Höbinger, J. Wallner, H. Pottmann, Packing circles and spheres on surfaces, in: ACM SIGGRAPH Asia 2009 papers, 2009, pp. 1–8.

[31] A. Gupta, G. Allen, J. Rossignac, Quador: Quadric-of-revolution beams for lattices, Computer-Aided Design 102 (2018) 160–170.

[32] 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.

[33] C. L. Chan, C. Anitescu, T. Rabczuk, Volumetric parametrization from a level set boundary representation with pht-splines, Computer-Aided Design 82 (2017) 29–41.

[34] M. Pan, F. Chen, W. Tong, Volumetric spline parameterization for isogeometric analysis, Computer Methods in Applied Mechanics and Engineering 359 (2020) 112769.

[35] E. D. Demaine, S. P. Fekete, R. J. Lang, Circle packing for origami design is hard, in: 5th International Conference on Origami in Science, Mathematics and Education, AK Peters/CRC Press, 2011, pp. 609–629.

[36] H. Pottmann, J. Wallner, Computational line geometry, Springer Science & Business Media, 2009.

[37] H. Pottmann, S. Leopoldseder, M. Hofer, Registration without icp, Computer Vision and Image Understanding 95 (1) (2004) 54–71.

[38] J. Machchhar, G. Elber, Dense packing of congruent circles in free-form non-convex containers, Com. Aided Geom. Design 52 (2017) 13–27.

[39] G. Elber, M.-S. Kim, Geometric constraint solver using multivariate rational spline functions, in: Proceedings of the sixth ACM symposium on Solid modeling and applications, ACM, 2001, pp. 1–10.

[40] X. Xiduo, L. Jijun, Z. Hong, A new nurbs offset curves and surfaces algorithm based on different geometry shape, in: 2009 IEEE 10th International Conference on Computer-Aided Industrial Design & Conceptual Design, IEEE, 2009, pp. 2384–2390.

[41] F. Massarwi, P. Antolin, G. Elber, Volumetric untrimming: Precise decomposition of trimmed trivariates into tensor products, Computer Aided Geo-

metric Design 71 (2019) 1–15.
[42] F. Massarwi, B. van Sosin, G. Elber, Untrimming: Precise conversion of trimmed-surfaces to tensor-product surfaces, Computers & Graphics 70 (2018) 80–91.