Designing and processing parametric models of steady lattices

Ashish Gupta[a], Kelsey Kurzeja[a], Jarek Rossignac[a], George Allen[b], Pranav Srinivas Kumar[c], Suraj Musuvathy[c]

[a]*School of Interactive Computing, Georgia Institute of Technology*
[b]*Siemens PLM Software Inc.*
[c]*Siemens Corporation, Corporate Technology*

Abstract

Our goal is to facilitate the design, analysis, optimization, and additive manufacturing of a specific class of 3D lattices that may comprise an extremely large number of elements. We target curved lattices that exhibit periodicity and uniform geometric gradations in three directions, along possibly curved axes. We represent a lattice by a simple computer program with a carefully selected set of exposed control parameters that may be used to adjust the overall shape of the lattice, its repetition count in each direction, its microstructure, and its gradation. In our Programmed-Lattice Editor (PLE), a typical lattice is represented by a short program of 10 to 50 statements. We propose a simple API and a few rudimentary GUI tools that automate the creation of the corresponding expressions in the program. The overall shape and gradation of the lattice is controlled by three similarity transformations. This deliberate design choice ensures that the gradation in each direction is regular (i.e., mathematically steady), that each cell can be evaluated directly, without iterations, and that integral properties (such as surface area, volume, center of mass and spherical inertia) can be obtained rapidly without having to calculate them for each individual element of the lattice.

Keywords: Lattice, Periodic Structure, Steady Pattern, Microstructure, Architected Material, Additive Manufacturing, Integral Properties, Similarity Transforms.

1. Introduction

Additive manufacturing (AM) is making it possible to fabricate parts with unprecedented structural complexity and optimized mechanical properties. To realize the transformational promise of this technology, the Solid Modeling community must address several challenges [1, 2]. In this paper, we focus on two of these: *design* and *processing*. We also conjecture that the theoretical and practical results proposed here may help advance *analysis* and improve the mechanical properties of the designed artifacts.

A key design challenge is the specification of microstructures of new architected materials [3]. Engineers and scientists who design such microstructures often use periodic lattices [4, 5], because these are simple to represent, are easy to analyze, and may provide a high strength-to-weight ratio [6]. When using fine granularity, these lattices may contain billions of beams and hence cannot be designed manually one-beam-at-a-time. They may be specified by procedural models [7, 8, 9, 10] or by designing one cell and replicating it. Our goal is to facilitate the design, analysis, optimization, and additive manufacturing [11, 12] of a class of such lattices.

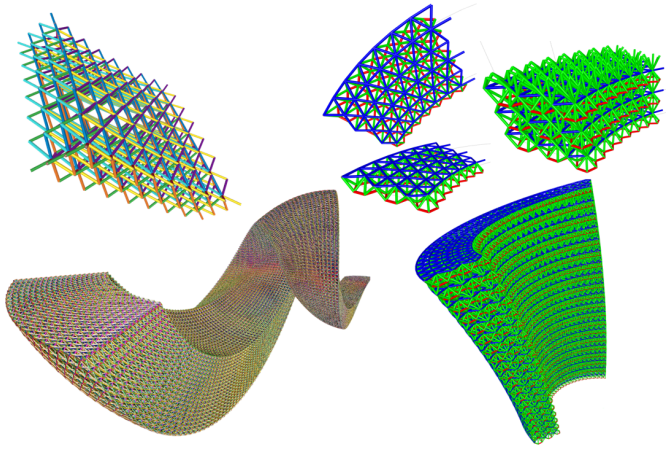


Figure 1: Steady lattices with detail shown at the top: Octet (left) with 1853280 beams and honeycomb with 57641 beams.

Specifically, we support curved lattices that exhibit periodicity and *steady* [13] shape gradation in three directions, along possibly curved axes. Steady gradation, curved overall shape, and the ability to align anisotropic mechanical properties with the different axes is essential for structural optimization in many application domains [14].

We propose to represent a lattice by a simple computer program that has a user-defined set of exposed control parameters. These may be used to adjust the overall shape, the repetition count in each direction, the microstructure,

Email addresses: ashishgupta79@gmail.com (Ashish Gupta), kkurzeja3@gatech.edu (Kelsey Kurzeja), jarek@cc.gatech.edu (Jarek Rossignac), Allen George <george.allen@siemens.com> (George Allen), Pranav Srinivas Kumar <pranav.kumar@siemens.com> (Pranav Srinivas Kumar), Suraj Musuvathy <suraj.musuvathy@siemens.com> (Suraj Musuvathy)

and the precise nature of the steady gradation.

To facilitate the writing of such programs, we propose a simple API and rudimentary GUI tools that automate the creation of program statements. In this ***Programmed-Lattice Editor (PLE)***, a typical lattice is defined by a program of 10 to 50 statements.

To illustrate the benefits of steadiness and to accelerate analysis and optimization, we propose efficient algorithms that evaluate integral properties (such as surface area, volume, center of mass and spherical inertia), without having to calculate them individually for each element of the lattice.

We make three assumptions:

(1) The lattice is made of *nodes* (balls) and *beams* (truncated cones), each smoothly connecting two balls.

(2) Nodes are arranged into a three-directional grid of *clusters*, in which all pairs of consecutive clusters in any one direction are related by the same similarity transform.

(3) Beams are grouped into *racks*, each connecting the same two nodes in each cluster or in each pair of clusters separated by constant index differences.

Examples of such lattices are shown in Fig. 1. Though the class of lattices described above have limited degrees of freedom, they offer substantial computational advantages over lattices produced by other more flexible approaches [9, 15]. These advantages are the ***key*** for viably modeling and analyzing lattices with billions or trillions of elements.

2. Terminology and notation

Research on mechanical analysis and homogenization of complex microlattices often considers periodic lattices, defined as a regular pattern of identical copies of an irreducible unit cell [16]. Authors of such analysis articles discuss lattices as being mono- or multi-atomic, i.e. having one or more nodes per cell. They also discuss lattices being one-, two-, or three-dimensional by considering a lattice to be an *assembly of unit cells* repeating in one, two or three directions respectively. We suggest a more precise terminology for several reasons.

(1) Mathematically speaking, the lattices are always three-dimensional, since they are made of solid primitives. Therefore we refer to them as rows, slabs, or bricks, based on the number of dimensions in which they repeat.

(2) In Solid Modeling, the term "assembly" refers to a set of solids with disjoint interiors. In a typical lattice, each node (ball) interferes (i.e., has a non-empty intersection) with one or more beams, and also two beams incident on a node may interfere with each other. Therefore, the lattice is not an "*assembly of nodes and beams*", but a collection or the set-theoretic union of them.

(3) The term "*assembly of identical cells*" is ambiguous and difficult to interpret in a Solid Modeling setting for two reasons. Although the dimensions of the axis-aligned cells and the distance between them, in a lattice with translational periodicity, are fixed by the periodicity of the lattice, their alignment (absolute positions) is not. For example, Fig. 2 shows three options for choosing the unit cell (black square). The top and middle ones split nodes. The bottom one splits beams. Furthermore, each one is stabbed by two instances of the orange beams. We argue that none of these three choices of cell alignment are suitable for analysis.

(4) One may define a curved lattice by deforming a regular (axis-aligned) lattice, using a mapping from Euclidean space to itself. This might lead to nodes and beams being bent themselves [15], thereby losing the truss-like properties and becoming much more difficult to analyze.

Because we support curved lattices with multiple nodes per cell, and because we define them using steady patterns [13], *cell*-based terminologies and mathematical formulations that have been proposed for simpler translational structures or deformations of these are not appropriate here. Hence, to make our presentation clear and mathematically precise, we start by defining, in this section, a set of key technical terms and notations, which we use throughout the paper. We group nodes into ***clusters*** and beams into ***racks***. The nodes (balls) in a clusters can have different radii. We construct steady patterns of clusters. We define the beams implicitly using the clusters. We then define the lattice as the union of all nodes in the clusters and of all beams in the racks.

For geometry, we use the following notation:

O, P, Q, F, P_i: isolated points.
FP: vector from F to P.
I, J, K, N, V, V_i: vectors.
$\underline{V}, \underline{FP}$: normalized (unit) versions of these vectors.
$V_1 \hat{} V_2$: angle between V_1 and V_2.
$V_1 \hat{}_N V_2$: angle from V_1 to V_2 around N
$V^\circ(\alpha, N)$: V rotated by α around N
$V^\circ(N)$: V rotate by $\pi/2$ around N

For transformations, we use the following notation:

$\boldsymbol{S}, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{M}$: orientation-preserving similarity transforms, which comprise any composition of translation, rotation, and dilation with a positive scaling factor.

$\boldsymbol{S} \cdot \mathbf{C}$: image of cluster $\mathbf{C}$ transformed by $\boldsymbol{S}$.

$\boldsymbol{S}^k$: k^{th} integer power of $\boldsymbol{S}$, i.e., the result of applying $\boldsymbol{S}$ k times.

Steady pattern: Ordered set of shapes X_i such that there exists a *base-shape*, X_0, and a similarity S, such that, for each valid i, $X_i = \boldsymbol{S}^i \cdot X_0$. More general *Steady Affine Patterns* have been defined in [13]. We focus on *Steady Similarity Patterns*, but omit the term *Similarity* for conciseness.

$\boldsymbol{F}(\mathrm{O}, I, J, K, s)$: Frame (origin O, orthonormal basis vectors I, J, K and scaling s) represents a similarity.

We group the nodes of a lattice into ***cluster***s, each having the same number $\underline{\mathrm{n}}$ of nodes, and organize these clusters into an array indexed by 3 variables. For this, we use the following terms. For conciseness, we assume that all indices (i, j, k) mentioned below are valid.

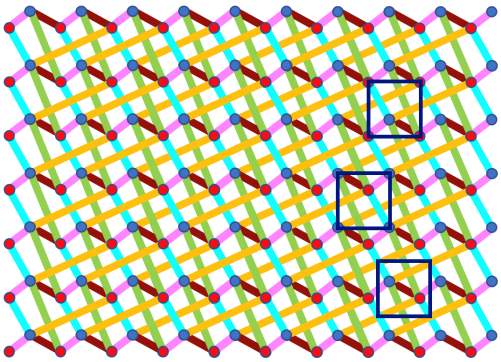
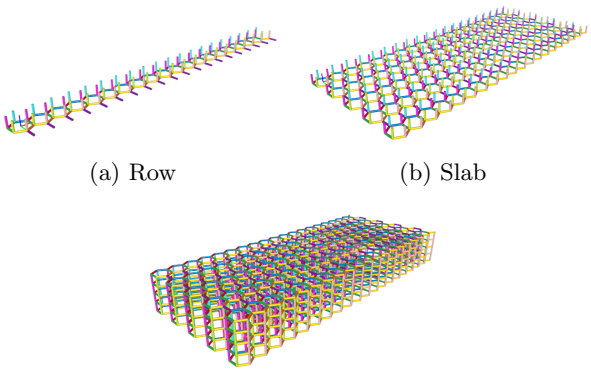


Figure 2: Lattice with two nodes (red and blue) per cluster (at different depths) and five beam racks (cyan, lime, orange, red, and magenta). The three black squares show options for defining the repeating element (irreducible unit cell).



(a) Row (b) Slab

(c) Brick

Figure 3: Lattice terminology.

Valid indices: Index i, (respectively j or k) is valid when $0 \leq i < \underline{i}$ (respectively $0 \leq j < \underline{j}$ or $0 \leq k < \underline{k}$), where $\underline{i}$ (respectively $\underline{j}$ or $\underline{k}$) is a *repetition count*. (These are: $\underline{i} = 10$, $\underline{j} = 7$ and $\underline{k} = 1$ in Fig. 2).

Node: A solid ball. Red and blue disks in Fig. 2.

Cluster C: A set of $\underline{n}$ disjoint nodes with indices $0, 1, \ldots, \underline{n} - 1$. (Red/blue node has index 0/1 in Fig. 2.)

Brick, $\mathbf{C}[,,]$: 3D array of clusters $\mathbf{C}[i, j, k]$, for which there is a ***base-cluster***, $\mathbf{C}_0$, and an array of similarities $\boldsymbol{S}[,,]$, such that $\mathbf{C}[i, j, k] = \boldsymbol{S}[i, j, k] \cdot \mathbf{C}_0$.

$\mathbf{N}[i, j, k, n]$: Particular node with index n in cluster $\mathbf{C}[i, j, k]$. (The blue node inside the top black square of Fig. 2 is $\mathbf{N}[7, 4, 0, 1]$, because it is in the 8^{th} column from the left, in 5^{th} row from the bottom, in the 1^{st} layer, and since it is the node with node-index 1.)

We define 2D and 1D subsets of a brick (See Fig. 3).

Slab, $\mathbf{C_2}[,]$: 2D array. Depending on the direction, we obtain a ***jk-slab*** $\mathbf{C_2}[j, k] = \mathbf{C}[i, j, k]$, an ***ik-slab*** $\mathbf{C_2}[i, k] = \mathbf{C}[i, j, k]$, or an ***ij-slab*** $\mathbf{C_2}[i, j] = \mathbf{C}[i, j, k]$.

Row, $\mathbf{C_1}[\,]$: 1D array. Depending on the direction, we obtain an ***i-row*** $\mathbf{C_1}[i] = \mathbf{C}[i, j, k]$, a ***j-row*** $\mathbf{C_1}[j] = \mathbf{C}[i, j, k]$, or a ***k-row*** $\mathbf{C_1}[k] = \mathbf{C}[i, j, k]$.

2.1. Steadiness of balls

We use the following terminology to distinguish ***steady*** bricks, slabs, and rows from unsteady ones. Steadiness facilitates design, accelerates processing, and may improve mechanical properties of the lattice.

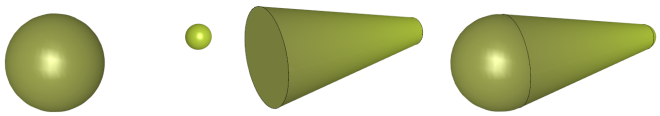
Steady row: A row for which there exist a similarity U such that $\mathbf{C_1}[i] = \boldsymbol{U^i} \cdot \mathbf{C}_0$. (The shapes $\mathbf{C_1}[i]$ form a steady pattern.)

Steady slab: A slab for which there exist similarities U and V such that $\mathbf{C_2}[i, j] = \boldsymbol{V^j} \cdot \boldsymbol{U^i} \cdot \mathbf{C}_0$. (The shapes $\mathbf{C_2}[i, j]$ form a steady pattern of steady patterns.)

Steady brick: A brick for which there exist similarities $\boldsymbol{U}$, $\boldsymbol{V}$, and $\boldsymbol{W}$ such that $\mathbf{C}[i, j, k] = \boldsymbol{W^k} \cdot \boldsymbol{V^j} \cdot \boldsymbol{U^i} \cdot \mathbf{C}_0$. (The shapes $\mathbf{C}[i, j, k]$ form a steady pattern of steady patterns of steady patterns.)

We group the beams into ***racks*** and define each rack in terms of node indices and their offsets as follows.

Beam: Truncated cone connecting two nodes with tangential continuity (Fig. 4).



(a) two balls (b) beam (c) convex hull

Figure 4: The convex hull of two nodes is their union with their beam.

Rack: The set of beams that each connects a *start-node*, $\mathbf{N}[i, j, k, s]$, to an *end-node*, $\mathbf{N}[i + x, j + y, k + z, e]$. (The lattice in Fig. 2 has 5 racks. The magenta rack, which has index 0, has beams between two balls of the same cluster. The beams in the other racks, cyan (1), lime (2), orange (3), and dark red (4), connect balls from different clusters.)

Set of racks: A set of $\underline{p}$ racks, where a rack $p \in [0, \underline{p}-1]$ is defined by five indices identifying: the start-node $s[p]$, the end-node $e[p]$, and the three start-to-end offsets $x[p]$, $y[p]$, and $z[p]$ of cluster-indices. (For example, in the lattice of Fig. 2, the rack count is $\underline{p} = 5$ and the orange rack ($p = 3$) is defined by: $s = 1, e = 0, x = 2, y = 1, z = 0$).

$\mathbf{B}[i, j, k, p]$: Particular beam in rack p that starts at node $s[p]$ of cluster $\mathbf{C}[i, j, k]$. (In Fig. 2, $\mathbf{B}[7, 4, 0, 3]$ is the orange beam that leaves from the blue node in the top black square.)

We can now define the lattice in terms of its brick and its set of racks. We distinguish unsteady lattices, steady lattices, and special cases of these.

Lattice : Union of all nodes in a brick and of all beams in a set of racks defined on these nodes. When one repetition count is 1, we call it a ***slab-lattice***. When two repetition counts are 1, we call if a ***row-lattice***.

Translational Lattice: A lattice for which $\boldsymbol{S}[i,j,k]$ is a translation by vector $i\boldsymbol{T}_i + j\boldsymbol{T}_j + k\boldsymbol{T}_k$. If vectors $\boldsymbol{T}_i$, $\boldsymbol{T}_j$ and $\boldsymbol{T}_k$ are orthogonal to each other, we call it an ***orthogonal translational lattice***. (The lattice in Fig. 2 is a orthogonal translational lattice.)

Steady Lattice: Lattice with a steady brick. (See Fig. 1 for two examples of steady lattices and their subsets.)

Finally, rather than trying to decompose the lattice into *cells*, we follow Wang et al. [17] and decompose it into an assembly of ***hubs*** (which they call *unit trusses*).

Stump of a node: Half-beam that connects to that node. It is defined by a *cut* plane that is orthogonal to the axis of the beam and, for example, equidistant from its two nodes. (See Fig. 5.)

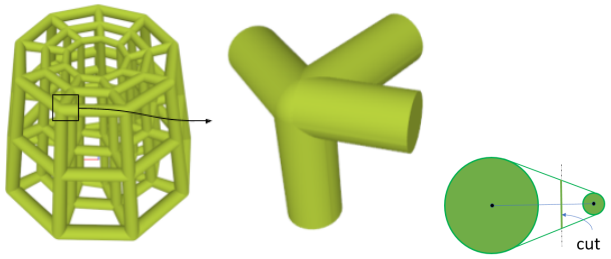


Figure 5: A lattice, one of its hubs, and a cross-section showing the cut, which is a disk that caps a stump.

Hub of a node: The union of a node with all its stumps. Observe that the lattice is the assembly of its hubs.

Clean lattice: One for which the interiors of different hubs do not intersect. This condition ensures that this decomposition is a valid assembly and leads to efficient analysis and parallelization. It also ensures that the boundary of a hub can be computed exactly.

3. Prior Art

We divide prior art into (1) Procedural models, (2) Regular lattices, (3) Conformal lattices, and (4) Irregular models.

3.1. Procedural Models

Our approach to represent the lattice by a program builds on early and more general procedural modeling paradigms [18]. For example, the MAMOUR system [19, 20] enabled to define patterns of repeated features (such as a string of uniformly spaced rivet holes around a face of a CAD model), to parameterize them, and to program parameter expressions in terms of geometric measures. Application of procedural modeling to repeated patterns and lattice structures are numerous.

The ABCSG system [7] supported a text-based representation of a scene-graph that combines solid primitives with operators (assembly, Boolean, Affine Transformation, Repetition, and Recursion) that can be edited with a keyboard or a GUI. The OCTOR system [8] provided effective tools for using the GUI to select sub-structures in such iterative or recursive patterns, so as to edit them or to specify exceptions at which instances do not follow the pattern.

Procedural methods that define geometry and coordinate systems using analytic implicit functions [9] have also been used to generate lattices.

3.2. Regular lattices

Orthogonal translational lattices (OTLs) are the most common type of lattice structures used in the design of mechanical objects [21].

They consist of copies of a cell template arranged on a regular grid. These types of lattices have also been used to design meta materials with microstructures, such as architected materials [3]. Interactive modeling and design is performed by orienting the grid, specifying cell templates, and selecting cell sizes and cell instance counts.

Solid modeling operations, such as extrude and sweep, have been applied to the design of lattice cell layouts [22].

R-Functions [9] have also been used to model OTLs and their variations, that for example include cylindrical and spherical patterns.

While the simplicity of these structures enables an efficient representation, the regularity of such structures constrains the kinds of lattices that can be modeled with this approach.

3.3. Conformal Lattices

Conformal lattices [23, 24] address the above limitation by requiring that edges or vertices of one of the faces of each cell of a base layer be aligned with iso-parametric curves of a surface. Additional layers of cells may be added by offsetting the base layer of conformal cells, or by interpolating between two base conformal layers lying on different surfaces [23, 24].

Recent work extends the conformal lattice patterning approach to the volumetric domain [15]. Here, all cells are aligned according to iso-parametric curves of trivariate B-Splines. In this approach, the geometry of the cell template is deformed according to the trivariate mapping.

The cell layout [22] may also be defined by a volumetric mesh, such as a tetrahedral or hexahedral mesh. Cell templates are deformed to match the shape of the corresponding volume mesh elements. A variant has been demonstrated in the Plato software system developed by Sandia Labs [25].

Alternatively, the vertices and edges of the volume mesh may themselves be used to define the lattice structure [22].

Implicit functions, including distance fields have also been used to define lattices [12, 22] and heterogeneous materials [26].

Warped lattice structures were created in [27] by morphing a 3D rectangular grid according to a warping function. The warping function was computed from an optimization procedure. Lattice templates were then morphed into each grid cell to create a continuous lattice structure.

Skin lattices that follow a prescribed surface may also be created [28] by trimming voxelized representation of functionally graded lattices.

These approaches broaden considerably the design space of regular lattices. Most of them require storing evaluated models of the supporting representations (such as the topology and geometry of tetrahedral meshes, the control points and knots of the control grids of splines, or the coefficients of the basis functions). The associated storage cost makes memory access expensive when processing lattices with billions of elements. More importantly, the irregularity of the patterns that they support complicates the task of identifying which cells lie near a given query (point, line, or plane) and makes geometric and integral queries expensive. Finally, we conjecture that the lack of steadiness of the patterns may often result in non-monotonic gradations of mechanical properties, which, we believe can make analysis and optimization more challenging, and ultimately may reduce the performance of the final product.

3.4. Irregular Models

A large body of literature focuses on procedural generation of non-periodic structures [29, 30]. These non-periodic lattices makes it simple to grade the geometry the microstructure, but does not offer precise control to define the geometry and topology of the microstructure. For example, procedural Voronoi Foams [10] is an example of a non-fully evaluated lattice representation. Such lattices are modeled similarly to Voronoi-based, procedural, cellular textures, which are very efficient to evaluate and store. However, the method is restricted to modeling stochastic, open-cell foams.

4. Steadiness

According to the definitions given in Sec. 2, in a Steady Lattice, each cluster can be expressed in terms of three similarities as follows: $\mathbf{C}[i,j,k] = \boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0$. In this section, we prove the steadiness of the patterns of clusters of its rows and discuss the steadiness of the corresponding patterns of its beams and hubs.

4.1. Steadiness of clusters

Consider the following two steps, through which we reformulate $\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0$:

$$\begin{aligned}\mathbf{C}[i,j,k] &= \boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0 \\ &= (\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U} \cdot \boldsymbol{V}^{-j} \cdot \boldsymbol{W}^{-k})^i \cdot (\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \mathbf{C}_0) \quad (1)\\ &= \boldsymbol{M}^i \cdot \mathbf{C}[0,j,k], \text{where } \boldsymbol{M} \text{ is a similarity.}\end{aligned}$$

The first step can be proven as follows:

- When $i = 0$, the formula is correct because both $\boldsymbol{U}^i$ and $(\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U} \cdot \boldsymbol{V}^{-j} \cdot \boldsymbol{W}^{-k})^i$ are the identity.
- When $i = 1$, $(\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U} \cdot \boldsymbol{V}^{-j} \cdot \boldsymbol{W}^{-k}) \cdot (\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \mathbf{C}_0)$ is $\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U} \cdot (\boldsymbol{V}^{-j} \cdot \boldsymbol{W}^{-k} \boldsymbol{W}^k \cdot \boldsymbol{V}^j) \cdot \mathbf{C}_0$ and simplifies to $\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U} \cdot \mathbf{C}_0$, which is correct, since $i = 1$.
- When $i > 1$, the term $\boldsymbol{V}^{-j} \cdot \boldsymbol{W}^{-k}$ on the right of each factor $(\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U} \cdot \boldsymbol{V}^{-j} \cdot \boldsymbol{W}^{-k})$ cancels with the term $\boldsymbol{W}^k \cdot \boldsymbol{V}^j$ on the left of the next factor or of $(\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \mathbf{C}_0)$.

In the second step, we simply use M to denote $(\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U} \cdot \boldsymbol{V}^{-j} \cdot \boldsymbol{W}^{-k})$ and $\mathbf{C}[0,j,k]$ to denote $\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \mathbf{C}_0$. This last expression shows that the pattern defined by clusters $\mathbf{C}[i,j,k]$ along an i-row (where j and k are fixed) is steady. Note that the base-cluster for this pattern is $\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \mathbf{C}_0$.

Similarly, we prove below that each j-row of clusters in a steady lattice is a steady pattern.

$$\begin{aligned}\mathbf{C}[i,j,k] &= \boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0 \\ &= (\boldsymbol{W}^k \cdot \boldsymbol{V} \cdot \boldsymbol{W}^{-k})^j \cdot (\boldsymbol{W}^k \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0) \quad (2)\\ &= \boldsymbol{M}^j \cdot \mathbf{C}[i,0,k]\end{aligned}$$

Finally, we prove below that each k-row of clusters a steady lattice is a steady pattern.

$$\begin{aligned}\mathbf{C}[i,j,k] &= \boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0 \\ &= (\boldsymbol{W}^k) \cdot (\boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0) \quad (3)\\ &= \boldsymbol{M}^k \cdot \mathbf{C}[i,j,0]\end{aligned}$$

4.2. Steadiness of beams

In this subsection, we prove that the set of all beams of a rack that start at a node along a k-row of clusters forms a steady pattern. For short, we say that each k-row of beams is steady.

Consider a beam rack p defined by the tuple $(s[p], e[p], x[p], y[p], z[p])$ (Sec. 2), which contains beams connecting the ball $\mathbf{N}[i,j,k,s[p]]$ in every cluster to the ball $\mathbf{N}[i+x[p], j+y[p], k+z[p], e[p]]$ for the valid range of indices. We already proved above the steadiness of k-rows of clusters, hence for balls along k-rows:

$$\begin{aligned}\mathbf{N}[i,j,k,s[p]] &= \boldsymbol{W}^k \cdot \mathbf{N}[i,j,0,s[p]] \\ \mathbf{N}[i+x[p], j+y[p], k+z[p], e[p]] &= \boldsymbol{W}^k \cdot \mathbf{N}[i,j,0,e[p]]\end{aligned} \quad (4)$$

As both the balls defining the beams in rack p follow the steady pattern defined by $\boldsymbol{W}$ along k-rows of clusters, the beam rack consists of steady k-rows of beams.

4.3. Steadiness of hubs

In Sec. 2 we defined a *hub* as the union of a ball and its stumps (half-beams) and we proved above that a steady lattice consists of steady k-rows of clusters (of balls) and steady k-rows of beams. Hence, we can conclude that a steady lattice consists of steady k-***rows of hubs***. Moreover, as all steady k-rows of clusters, beams and therefore of hubs are defined by a common similarity $\boldsymbol{W}$, a steady lattice consist of steady k-***slabs of hubs***.

In the next section we describe fast, accurate and direct evaluation of clusters in a steady lattice.

5. Computing clusters of a Steady Lattice

To compute a cluster $\mathbf{C}[i,j,k] = \boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0$ in a steady lattice, we need to compute powers of similarity transforms. We do this accurately and efficiently as follows.

5.1. Canonical decomposition of a similarity

Given a similarity $\boldsymbol{U}$, we decompose it into the canonical commutative product $\boldsymbol{R} \cdot \boldsymbol{T}$, where $\boldsymbol{R}$ and $\boldsymbol{T}$ are primitive transformations (translation, rotation and scaling), such that $\boldsymbol{U}^t = \boldsymbol{R}^t \cdot \boldsymbol{T}^t$, for any value of parameter t.

We distinguish two cases, depending on the value of m, the scaling factor of $\boldsymbol{U}$, which we assume to be strictly positive.

When $m = 1$, $\boldsymbol{U}$ is a rigid body transformation. Then, $\boldsymbol{R}$ is a rotation and $\boldsymbol{T}$ a translation by a vector parallel to the axis of $\boldsymbol{R}$ (see for example [31]). In this case, as t varies, $\boldsymbol{U}^t$ defines a ***screw motion*** and point $\boldsymbol{U}^t \cdot \mathrm{P}_0$ traces a ***helix***. (See [13] for the computation of $\boldsymbol{U}^t$.)

When $m \neq 1$, $\boldsymbol{U}$ is a similarity. Then, $\boldsymbol{R}$ is a rotation and $\boldsymbol{T}$ a dilation about a fixed point F on the axis of $\boldsymbol{R}$. In this case, as t varies, $\boldsymbol{U}^t$ defines a ***swirl motion*** and point $\boldsymbol{U}^t \cdot \mathrm{P}_0$ traces a ***concho-spiral*** [32].

Note that in special cases, the *screw* may reduce to pure translation ($\alpha = 0$) or pure rotation ($d = 0$) and the *swirl* may reduce to pure scaling ($\alpha = 0$).

The precise expressions for these two motions are given below using the notation presented in Section 2. We evaluate them for discrete values of parameter t to produce steady patterns.

$$\boldsymbol{U}^t \cdot \mathrm{P}_0 := \mathrm{F} + (td)N + (FP_0)^\circ(t\alpha, N) \ \backslash\backslash \text{Screw} \tag{5}$$

$$\boldsymbol{U}^t \cdot \mathrm{P}_0 := \mathrm{F} + m^t(FP_0)^\circ(t\alpha, N) \ \backslash\backslash \text{Swirl} \tag{6}$$

Here, F is the fixed point of the dilation, N is the unit vector along the axis of rotation, α is the angle of rotation, d is the distance of translation, and m is scaling. Please refer to Appendix A for the derivation of these parameters.

The steady pattern of shapes that are produced using patterns of similarity frames (along the above screw and swirl motions) that result from a uniform sampling of t are called ***screw-patterns*** and ***swirl-patterns***.

5.2. Direct computation of cluster balls

To compute the balls of cluster $\mathbf{C}[i,j,k]$, we transform each ball of the base-cluster, $\mathbf{C}_0$, by the appropriate similarity. To do so, we pre-compute the canonical decomposition of each of the similarities $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{W}$, that define a steady lattice. Then, we compute the center and radius of ball $\mathbf{N}[i,j,k,n]$ by transforming the center C_n and radius r_n of the corresponding ball in the base-cluster as follows, assuming that m_u, m_v and m_w are the scaling factors of $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{W}$.

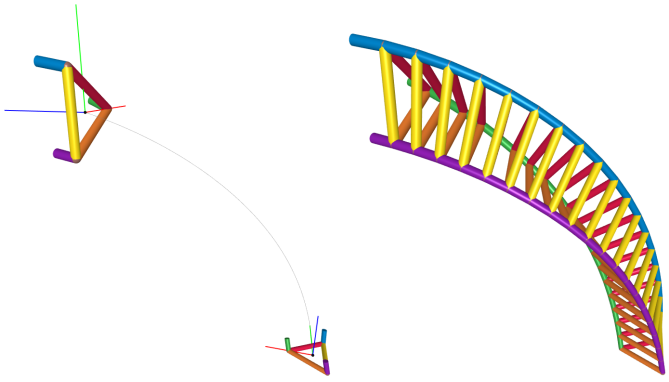
$$\mathbf{N}[i,j,k,n] = \mathit{Ball}(\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathrm{C}_n,\ m_w^k m_v^j m_u^i r_n) \tag{7}$$

5.3. Direct computation of interpolating clusters

When the designer programs the parameters of similarities $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{W}$ explicitly, we say that the corresponding cluster was defined by its incremental similarities. Although this approach may work when $\boldsymbol{U}$,$\boldsymbol{V}$ and $\boldsymbol{W}$ are simple (for example, rotations or translations about the same or orthogonal axes), it fails completely when attempting to design a pattern that interpolates the first and last cluster, which may have each been carefully placed to meet some assembly constraints.

Hence, we propose a tool that computes the incremental similarities, such that the resulting pattern interpolates the first and last key-clusters, which are related by a ***cumulative similarity***, say $\boldsymbol{U}' = \boldsymbol{U}^{\underline{i}-1}$. In this scenario, the designer may adjust $\boldsymbol{U}'$ via the parameterized program or GUI to control the overall size, shape, and scaling of the lattice using the first and last clusters in a particular direction.

Given $\boldsymbol{U}'$ and the repetition count $\underline{i}$, we use the canonical decomposition discussed above to compute the incremental similarity as $\boldsymbol{U} = (\boldsymbol{U}')^{1/(\underline{i}-1)}$, i.e. a fractional power of $\boldsymbol{U}'$. This approach always always yields a valid solution and is simpler than the one proposed in [13] for general affinities. Fig. 6 shows a steady row lattice, created by interpolating between the base frame and a cumulative similarity frame.



(a) Cumulative similarities (b) Interpolating lattice

Figure 6: Row lattice computation by interpolating clusters

6. Integral properties

In this section, we derive closed form expressions for computing specific integral properties, namely *surface area, volume, center of mass and spherical inertia* of the lattice, given the corresponding property of the first ij-slab, which is obtained by computing and then aggregating the corresponding properties for each hub in that first ij-slab.

6.1. Integral properties of a hub

Integral properties of a hub may be approximated from its voxelization or octree decomposition (see for example

[33]), from ray sampling, or from a triangulation of its boundary. For example, given that the interior side of the bounding faces of a hub is completely visible from the center C of the hub's ball, we can approximate the hub by inflating its ball, i.e. approximate that ball surface with an almost regular triangle mesh ([34]) and then push each vertex out radially away from the C until it reaches the boundary of the hub. Then, approximate the volume of the hub by the sum of the volume of the tetrahedron, one per triangle. But, this approach produces large approximation errors (Fig. 7), especially towards the far end of the stump, away from the center C. Figure 8 shows that we need a large number of rays (N) to substantially improve our approximation of each stump.

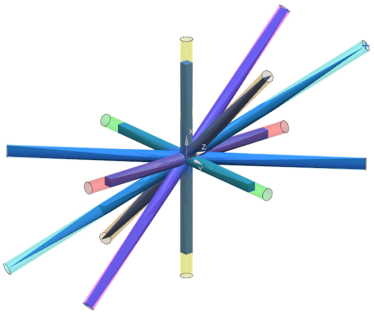


Figure 7: Hub(transparent) and its discretization

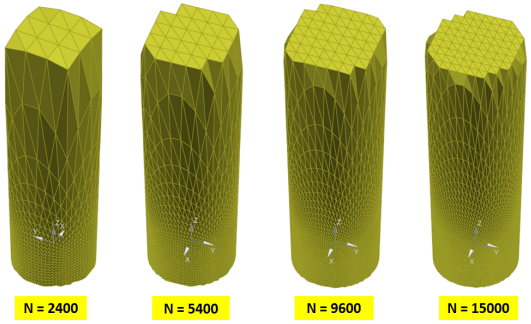
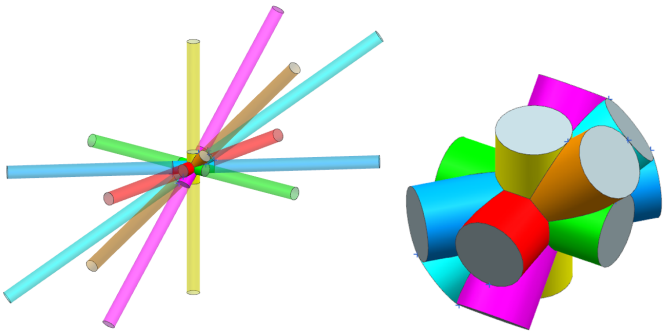


Figure 8: Stump discretization for different number of rays (N)

To improve hub's approximation with considerably less number of rays, we propose to split the hub into a ***core*** and ***clean*** portions of stumps (Fig. 9), and compute integral properties of the hub by aggregating integral properties of its core and clean portions. By *clean* portions we mean that interiors of those stump portions are pairwise disjoint. To identify the clean portion of each stump, we compute in a pairwise manner, how far the intersection of two stumps extend along the axis of the two stumps. This is reduced to a 2D problem of identifying intersection of two lines, that correspond to the profile of the conical surface of the two stumps, in the plane containing the axes of the two stumps. We then split each stump at the farthest intersection, to extract its clean portion.

The integral properties of the *core* can then be approximated from its tetrahedral discretization (Fig. 10) following the ray shooting approach described above. For the *clean* portions of the stumps (cylinders or cones), the integral properties can be computed trivially.



(a) Split hub (b) Core

Figure 9: Hub's splitting into core and clean portions of stumps

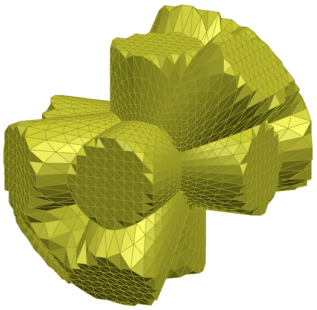


Figure 10: Discretized core

Figure 11 shows how the error in computing the volume of the hub of Figure 9a, by discretizing just its core, reduces very quickly to a very low value with just few hundred rays per hub. Thus, discretizing only it's core and not the whole hub substantially improves performance. For example, the tetrahedral discretization of the whole hub shown in Fig. 9a using close to 60000 rays, results in more than 2.5% error in volume computation, while with just 600 rays, this error is close to 0.22% if we discretize only the core.

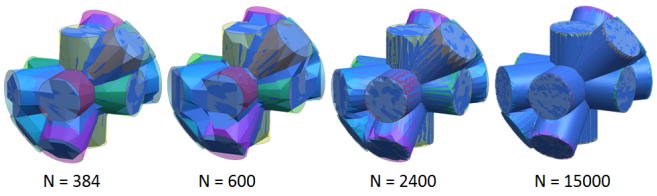
6.2. Surface area of the lattice

Let a_0 be the combined surface area of all the hubs (without the end caps of stumps) in the first ij-slab and let a_k denote the surface area of ij-slab k.

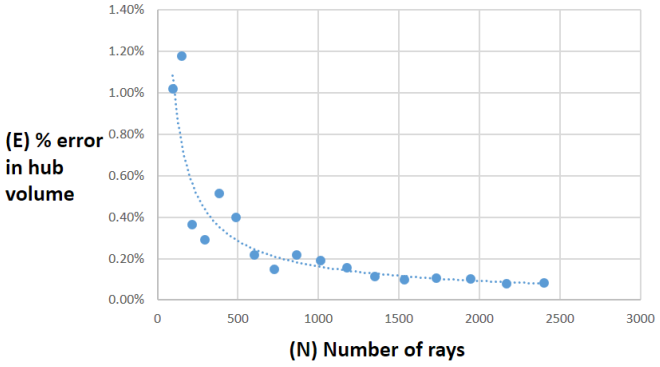
If $\boldsymbol{W}^k$ is a screw pattern, each *hub* of ij-slab 0 undergoes a rigid transformation, hence the total surface area a_L of the lattice is simply $\underline{k}a_0$.

If $\boldsymbol{W}^k$ is a swirl pattern, *hub* of of ij-slab k is scaled by m_w^k, hence the total surface area of the lattice is:

$$a_L = \sum a_k = a_0 \sum (m_w^2)^k = \frac{a_0((m_w^2)^{\underline{k}} - 1)}{(m_w^2 - 1)} \qquad (8)$$



(a) Discretized core within actual core volume in transparent.



(b) Hub's volume error vs number of rays

Figure 11: Improved approximation at small number of rays due to hub splitting

In the equation above and onwards in this section, we will use $\sum$ to denote $\sum_{k=0}^{\underline{k}-1}$ for simplicity.

6.3. Volume of the lattice

Let v_0 be the combined volume of all the hubs in the ij-slab 0 and v_k be the volume of ij-slab k.

If $\boldsymbol{W}^k$ is a screw pattern, the total volume v_L of the lattice is $\underline{k}v_0$.

If $\boldsymbol{W}^k$ is a swirl pattern, the volume of the lattice is:

$$v_L = \sum v_k = v_0 \sum (m_w^3)^k = \frac{v_0((m_w^3)^{\underline{k}} - 1)}{(m_w^3 - 1)} \tag{9}$$

6.4. Centroid of the lattice

Let G_0 be the *centroid* (center of mass) of the ij-slab 0. Then, assuming uniform material density through out the lattice, the centroid G_L of the whole lattice can be computed as the weighted average of the centroids of all the k-slabs:

$$\mathrm{G}_L = \frac{\sum v_k \boldsymbol{W}^k \cdot \mathrm{G}_0}{v_L} \tag{10}$$

As done earlier, we consider two cases.

6.4.1. Swirl pattern

In case of a swirl pattern of slabs, the volume of the slab also scales, hence the expression for centroid becomes:

$$\mathrm{G}_L = \frac{v_0 \sum (m_w^3)^k \boldsymbol{W}^k \cdot \mathrm{G}_0}{v_0((m_w^3)^{\underline{k}} - 1)/(m_w^3 - 1)} \tag{11}$$

Now, from Eq. 6:

$$\boldsymbol{W}^k \cdot \mathrm{G}_0 = \mathrm{F} + m_w^k (FG_0)^\circ(k\alpha, N) \tag{12}$$

Let $V = FG_0$, $V_1 = (V.N)N$, $V_2 = V - V_1$, $c_k = \cos(k\alpha)$ and $s_k = \sin(k\alpha)$, then:

$$\begin{aligned} V^\circ(k\alpha, N) &= (V_1 + V_2)^\circ(k\alpha, N) \\ &= V_1 + V_2^\circ(k\alpha, N) \\ &= V_1 + c_k V_2 + s_k V_2^\circ(N) \end{aligned} \tag{13}$$

Using the above expression in Eq. 11, centroid G_L can be expressed as:

$$\mathrm{G}_L = \frac{(\sum m_w^{3k})\mathrm{F} + (\sum m_w^{4k})V_1 + (\sum m_w^{4k} c_k)V_2 + (\sum m_w^{4k} s_k)V_2^\circ(N)}{(m_w^{3\underline{k}} - 1)/(m_w^3 - 1)} \tag{14}$$

While other summations in the above expression are trivial, the summations with trigonometric terms can be computed by substituting $a = m_w^4$ in the following expressions [35]:

$$\begin{aligned} \sum a^k c_k &= \frac{1 - ac_1 - a^{\underline{k}} c_{\underline{k}} + a^{\underline{k}+1} c_{\underline{k}-1}}{1 + a^2 - 2ac_1} \\ \sum a^k s_k &= \frac{as_1 - a^{\underline{k}} s_{\underline{k}} + a^{\underline{k}+1} s_{\underline{k}-1}}{1 + a^2 - 2ac_1} \end{aligned} \tag{15}$$

6.4.2. Screw pattern

In case of screw pattern of slabs, the volume of each slab remains constant, thus:

$$\mathrm{G}_L = \frac{v_0 \sum \boldsymbol{W}^k \cdot \mathrm{G}_0}{\underline{k} v_0} \tag{16}$$

From Eq. 6:

$$\boldsymbol{W}^k \cdot \mathrm{G}_0 = \mathrm{F} + (kd)N + (FG_0)^\circ(k\alpha, N) \tag{17}$$

Similar to the derivation of centroid in case of swirl rack, the centroid for the screw case can be expressed as:

$$\mathrm{G}_L = \frac{\underline{k}\mathrm{F} + \underline{k}V_1 + (d\sum k)N + (\sum c_k)V_2 + (\sum s_k)V_2^\circ(N)}{\underline{k}} \tag{18}$$

The summations with trigonometric terms can be computed by substituting $a = 1$ in Eq. 15.

6.5. Spherical moment of inertia of the lattice

The spherical moment of inertia, or simply "***spherical inertia***", of a body B about a fixed point G is $i_{\mathrm{G}} = \int_B r^2 dm$, where r is the distance of the infinitesimal mass dm from G and body's total mass is $m = \int_B dm$. For example, for a ball of uniform density ρ, center G and radius r, $i_{\mathrm{G}} = \rho(4/5)\pi r^5$ and $m = \rho(4/3)\pi r^3$. The spherical inertia of the body about a different point P can be transferred from G using $i_{\mathrm{P}} = i_{\mathrm{G}} + m|\mathrm{PG}|^2$.

We compute the spherical inertia of a steady lattice L about its centroid G_L as follows.

We assume, that we can compute the mass m, the centroid G, and the spherical inertia i for each hub of ij-slab 0 about their respective centroid. We can then compute the mass m_0, centroid G_0, and spherical inertia i_0 of the entire ij-slab 0 about its centroid G_0. The spherical inertia i_0 is computed by using the transfer formula given above and summing the results. We compute the center of mass, G_L of the lattice and then compute the spherical inertia i_L of the whole lattice about G_L, by combining the inertia of all ij-slabs as follows:

$$i_L = \sum (i_k + m_k |G_L G_k|^2) \tag{19}$$

where, i_k is the spherical inertia of ij-slab k about its centroid G_k and m_k is its mass.

To formulate a closed-form expression for this sum, we use the terms defined in the previous section. For conciseness let vector $V_3 = G_L F$.

For a swirl pattern, the spherical inertia of the lattice L about its center of mass is:

$$i_L = q_0 + m_0(q_1 + q_2 + q_3 + q_4 + q_5) \tag{20}$$

where,

$$q_0 = i_0 \sum m_w^{5k}, \;\; q_1 = |V_3|^2 \sum m_w^{3k}$$
$$q_2 = |V|^2 \sum m_w^{5k}, \;\; q_3 = 2V_3 \cdot V_1 \sum m_w^{4k}$$
$$q_4 = 2V_3 \cdot V_2 \sum (m_w^{4k} c_k), \;\; q_5 = 2V_3 \cdot V_2^{\circ}(N) \sum (m_w^{4k} s_k)$$

For a screw pattern, it is:

$$i_L = q_0 + m_0(q_1 + q_2 + q_3 + q_4 + q_5) \tag{21}$$

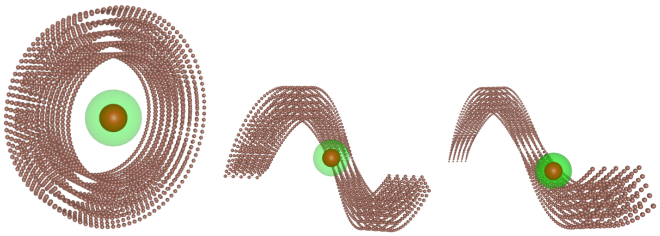
where,

$$q_0 = \underline{k} i_0, \;\; q_1 = \underline{k}(|V_3|^2 + |V|^2 + 2V_3 \cdot V_1)$$
$$q_2 = 2d(V \cdot N + V_3 \cdot N) \sum k, \;\; q_3 = d^2 \sum k^2$$
$$q_4 = 2V_3 \cdot V_2 \sum c_k, \;\; q_5 = 2V_3 \cdot V_2^{\circ}(N) \sum s_k$$

Extensions of such closed-form expressions for inertia about a given axis and hence for inertia tensors appear significantly more challenging and is a subject of the ongoing research.

6.6. Results

Fig. 12 shows results of comparing volume and centroid of a brick of clusters, computed by using brute force Vs using closed form expressions Eq. 9, 14 and 18. The green ball corresponds to result from the equation, it has volume equal to that of brick and is placed at the centroid. Similarly the red ball corresponds to result from brute force computation. The red ball's radius has been reduced by half to visualize the two balls together. Observe that the two ball centers are identical and red ball's radius is half of the green ball's radius.



(a) pure rotation (b) screw pattern (c) swirl pattern

Figure 12: Verification of equations for volume and centroid

As for performance, as an example, the closed form computation of volume of all the beams of a bent and graded brick lattice (Fig. 13) of size $10000 \times 10000 \times 10000$, with more than 24×10^{12} beams, took under 10 minutes, which if done by brute force would take several days.

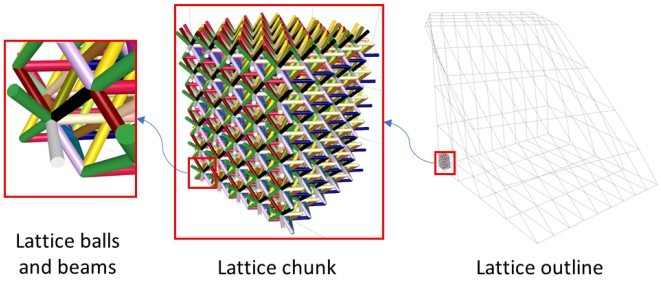


Figure 13: Volume of a large lattice

7. The Programmed Lattice Editor (PLE)

In this section, we discuss the workflow and the programming tools that we support.

7.1. Process flow and programming

Typically, the development of a lattice model evolves through the following stages of design and analysis:

1. **Programming:** A *programmer* uses the API and the GUI to edit a short program that defines a lattice. Fig. 14 shows a PLE program and the corresponding lattice for default parameter values.

2. **Rigging:** The programmer identifies a *small* set of variables as *control parameters*, provides their default values, and programs formulae to compute the values of other parameters from these. Typical control parameters include the global similarity transforms U', V', W', the repetition counts, the radii of the various nodes in the base-cluster. But other control parameters may be easily programmed and may even be used to trigger conditional branches in the creation of the lattice. Fig. 15 shows instances produced by an *LMA* program for different values of its control parameters.

3. **Compiling:** The result is *compiled* into a ***Lattice Maker Applet (LMA)***, which can use the default values or accept new values of the control parameters to generate a particular variant of the lattice.

4. **Optimizing:** A structural engineer uses the LMA inside an optimization loop that creates a model of the lattice that is suitable for analysis, performs analysis, and tweaks the control parameters. The efficient computation of mass properties discussed above may help to accelerate this process.

5. **Exporting:** The engineer then exports the optimized lattice to another CAD, or AM system for further processing, simulation, or manufacturing. Currently we export the whole lattice or a portion of it, as a text file containing list of center and radius of each ball and a list of pair of ball indices for each beam.

Note that throughout these design stages, the lattice need not be evaluated and stored in its entirety. Instead, it is represented implicitly by a parameterized LMA, which supports direct querying and lazy (on demand) evaluation for one Region-of-Interest (RoI) at a time.

7.2. GUI

PLE provides advanced GUI controls for direct manipulation of the lattice using mouse actions. The user can add or delete a ball or beam, grow or shrink a ball, or can bend, twist or scale the whole lattice via mouse clicks and drags. For example, the user can control the four frames representing the four similarities, namely $\boldsymbol{M}, \boldsymbol{U'} \cdot \boldsymbol{M}, \boldsymbol{V'} \cdot \boldsymbol{M}, \boldsymbol{W} \cdot \boldsymbol{M}$, where frame $\boldsymbol{M}$ positions the base cluster $\mathbf{C}_0$ in space and where $\boldsymbol{U'}, \boldsymbol{V'}, \boldsymbol{W'}$ are the cumulative similarities (Sec. 5.3) defining the lattice. Each frame can be dragged, rotated and scaled to modify the lattice in real time. A typical editing session for a row-lattice is shown in Fig. 16, where the end cluster (red) is dragged right, rotated around the viewing direction (blue), scaled and then rotated again respectively.

7.3. Assisted coding

The benefits of combining text editing and graphic manipulation for changing the transformations in a pattern have been demonstrated in [7]. In PLE one can use the mouse to click, one after the other, the two balls that she

```
class cParams{
  // Default values of control parameters
  float ringRad = 100.0, ballRad = 5.0, scale = 1.2;}

cLattice latticeMaker(cParams params){
  // local names of control parameters (for convenience)
  float R=params.ringRad, r = params.ballRad, s = params.scale;

  // Specify nodes in the base cluster
  cluster.addBall(R,0,0, r);

  // transform base cluster
  cluster.translate(1000,0,0);

  // define beam racks
  beams.addBeam(0,0,1,0,0); // red
  beams.addBeam(0,0,0,1,0); // green
  beams.addBeam(0,0,0,0,1); // yellow

  // define repetiotion counts
  int n1=7, n2=20, n3=50;

  // define incremental transforms
  pt p = bC.mF.O; // transformed cluster's origin, used to define U,V,W below

  U.translate(20,0,0); U.scale(s,p);
  V.rotate(2.0*PI/(float)n2, V(0,0,1.0),p);
  W.translate(0,100,0); W.scale(s); W.rotate(-4.0*PI/(float)n3, V(0,1.0,0));

  // indicate that direction 2 is cyclic
  lattice.closeDirection2();

  // create and retun lattice
  return lattice(cluster,U,V,W,n1,n2,n3);}
```

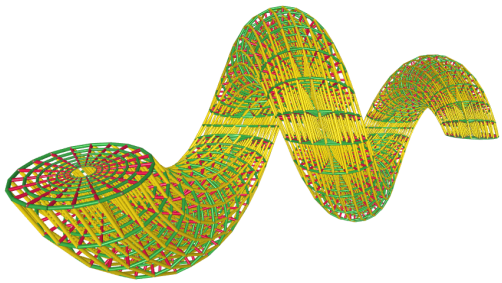
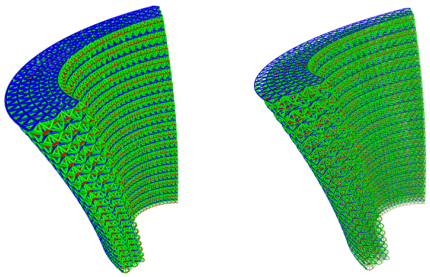
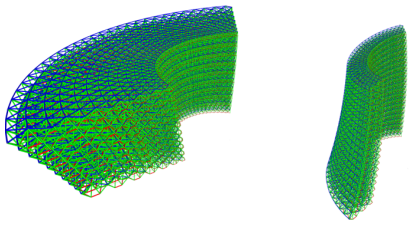


Figure 14: A program written in PLE and the corresponding lattice.



(a) Thick beams (b) Thin beams



(c) Different repetition counts (d) Different scaling

Figure 15: Parametric lattice instances.

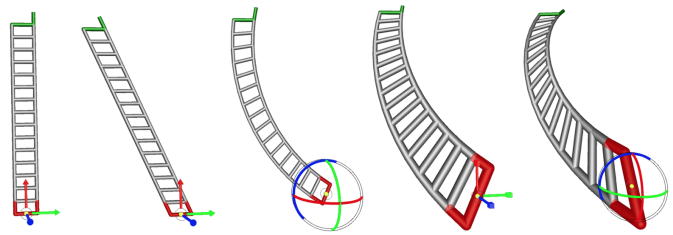


Figure 16: Interactive editing of a row-lattice.

wishes to join by a new beam. PLE then composes automatically the corresponding program statement and places it in the clipboard, for the programmer to paste at the proper place in the program. Similarly, one can manipulate a frame and a string containing the statement that creates this precise frame is inserted automatically into the clipboard.

7.4. Selective rendering

PLE supports visualizing a selected contiguous region of interest (RoI) of an otherwise large lattice. This RoI is specified by the indices of a starting cluster (i, j, k) and the span $(\delta i, \delta j, \delta k)$. The user can move and grow or shrink this region (Fig. 17). To put this RoI in the context of the overall lattice, we display the outline of the complete lattice by tracing the centroid of the base cluster $\mathbf{C}_0$ in a regularly sampled subset of rows of the 6 boundary slabs. For a given sampling of rows in each slab and of sampling of points along each row, the outline can be drawn in constant time.

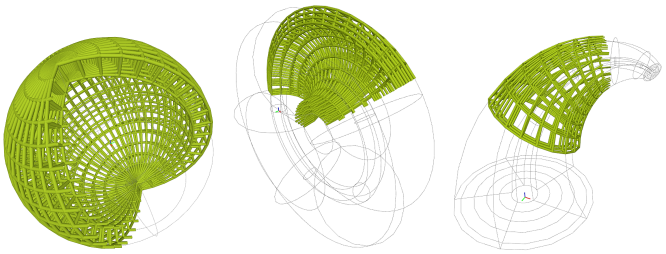
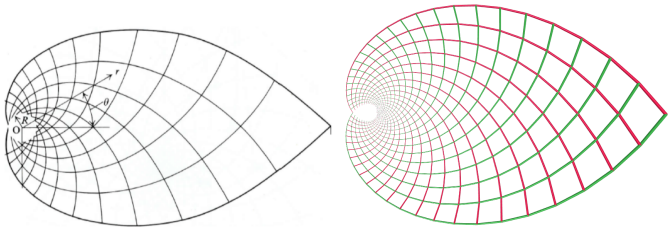


Figure 17: Selectively rendered lattices.

7.5. Examples

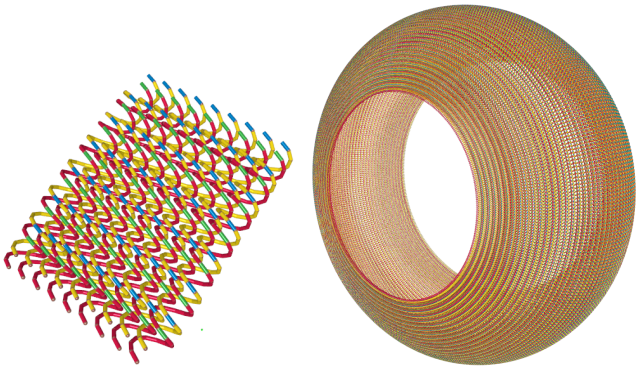
Lattices shaped like Michell lattices [36] for optimum cantilever structure, can be constructed from two spiral patterns with common fixed point [37]. Figure 18 shows such a steady lattice slab with $\boldsymbol{U}$ and $\boldsymbol{V}$ being swirl transforms.

Fig. 19 shows example of a tire shaped lattice with spring shaped elements woven together to form the internal structure. Note that this lattice is created by defining a base-cluster of just two balls, four beam racks and three similarities (screw, rotation, rotation) in a compact PLE program, similar to the one shown in Fig. 14.



(a) Michell's solution [36] (b) Steady lattice slab

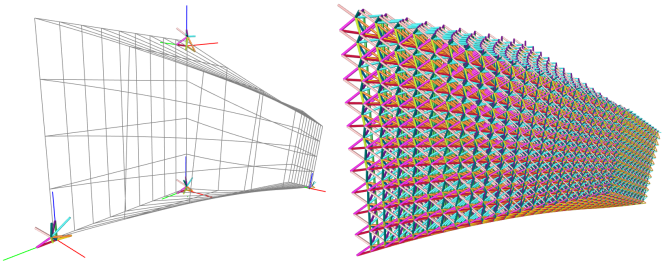
Figure 18: Steadiness in Michell lattices



(a) Weave structure (b) Spring tire

Figure 19: Woven lattices

Fig. 20 shows a brick lattice with octahedral unit cell, created by specifying three cumulative similarities. Observe that the lattice interpolates, while it twists and grades steadily.



(a) Cumulative similarities (b) Interpolating lattice

Figure 20: Steadily graded interpolating brick lattice

8. Productionisation

The initial research prototype used to validate and demonstrate the concepts described above was implemented in Processing. As a step towards the broad distribution of the results presented here, we have ported these tools into ***Mithril***, an environment for the rapid prototyping, analysis, and 3D printing of highly complex lattices with graded material structures.

The core of Mithril is written in C/C++ with interfaces to (1) CPython for interpreting construct programs, (2)

OpenGL for visualization, and (3) CUDA for parallelizing computations with GPUs. We have chosen CPython, i.e., the C implementation of Python, as our programming language for lattice programs as it is an easy-to-use, well supported, and widely used in both academic and industrial environments. A design engineer can program complex lattice structures in Python with loops, conditional statements, and standard Python libraries such as numpy. Furthermore, Python can be *extended* with custom C/C++ libraries. The C/C++ library becomes an *extension module* that can be directly imported in Python. Efficient algorithms, such as for computing mass properties, are implemented in C/C++ libraries, and exposed to Python with the Boost Python library.

In addition to extending Python, Mithril *embeds* Python where the application calls the Python interpreter to execute lattice programs and construct corresponding C++ objects. Extracting a C++ object from its Python counterpart is similarly done using Boost Python. The C++ lattice object is then used for visualization and performing advanced analyses.

With this approach, a design engineer uses the Mithril UI to program lattice structures in Python, visualize the lattices using the C++-based rendering engine, and analyze them using efficient C++-based implementations of analysis algorithms, all within the same software application. This solution provides a powerful rapid-prototyping interface for engineers and enables a very concise representation for complex lattice structures.

Fig. 21 shows a section of the 3D printed spring tire lattice constructed in Mithril.

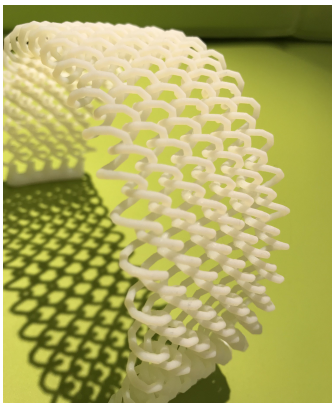


Figure 21: 3D printed lattice

9. Advantages

The advantages of the proposed representation, decomposition and computation of lattice structures are:

1. **Simple, compact and capable:** Our approach is simple, yet it has the capability to design complex bending, twisting and grading lattices (Fig. 1, 15, 19) with compact programs (Fig. 14).

2. **Directly computable:** As described in Sec. 5 the clusters of steady lattice can be computed accurately and directly, i.e. without an iterative process, by transforming the base-cluster along screw- or swirl-motions.

3. **Clean decomposition:** A *clean lattice* can be decomposed into an assembly of non-overlapping *hubs* which facilitates processing and parameterization and reduces boundary evaluation costs.

4. **Improved homogenization:** The position, orientation, and size of the clusters along any row of a steady lattice varies steadily. Since the beam patterns have a periodic definition in terms of cluster nodes, the geometry of hubs (or cells) also varies smoothly. We conjecture that this regularity may improve homogenization [38] and may also yield lattices with better mechanical properties than those of lattices obtained by warping a regular grid of cells.

5. **Efficient analysis and optimization:** Mass properties of a steady lattice can be computed accurately and efficiently, i.e. without iterating over all the elements of the lattice (Sec. 6). Similar advantage can be derived in performing geometric queries, such as PMC over a steady lattice ([39]).

   The above advantages are crucial to achieve scalability required to model and analyze lattices with billions or trillions of elements.

10. Conclusions

We propose *Programmed-Lattice Editor (PLE)* - a new design environment to program, graphically edit, parameterize, and export parametric models of lattices. We define a lattice by a three directional *brick* of clusters of balls and by a set of beam patterns. To simplify processing and validity formulation and testing, we consider a decomposition of the lattice into an assembly of quasi-disjoint hubs. The brick of cluster is fully defined by a base-cluster (small set of balls), by three similarities, and by three repetition counts. Each cluster is obtained as the image of the base-cluster by three consecutive evaluations of steady similarity patterns. The resulting steadiness of the brick allows the programmer to expose simple parameters that control the bending, twisting, and shrinking of the overall structure and consequently the gradation of its cells. More importantly, it may be used for accelerating the evaluation of mass properties. We conjecture that it may benefit the accuracy of homogenization, and hence analysis and optimization, and that it may yield material structure with smoother gradations of physical properties. We prove steadiness of all the rows of clusters, provide formulae for exploiting them in mass-property calculations, and provide formulae for computing the three similarities

via interpolation of the first and last frames in a row. We present several simple GUI tools that facilitate and accelerate the editing and visualization.

11. Acknowledgements

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] William Regli, Jarek Rossignac, Vadim Shapiro, and Vijay Srinivasan. The new frontiers in computational modeling of material structures. *Computer-Aided Design*, 77:73–85, 2016.

[2] Wei Gao, Yunbo Zhang, Devarajan Ramanujan, Karthik Ramani, Yong Chen, Christopher B Williams, Charlie CL Wang, Yung C Shin, Song Zhang, and Pablo D Zavattieri. The status, challenges, and future of additive manufacturing in engineering. *Computer-Aided Design*, 69:65–89, 2015.

[3] Tobias A Schaedler and William B Carter. Architected cellular materials. *Annual Review of Materials Research*, 46:187–210, 2016.

[4] Chen Chu, Greg Graf, and David W Rosen. Design for additive manufacturing of cellular structures. *Computer-Aided Design and Applications*, 5(5):686–696, 2008.

[5] Wenjin Tao and Ming C Leu. Design of lattice structure for additive manufacturing. In *Flexible Automation (ISFA), International Symposium on*, pages 325–332. IEEE, 2016.

[6] Tobias A Schaedler, Alan J Jacobsen, Anna Torrents, Adam E Sorensen, Jie Lian, Julia R Greer, Lorenzo Valdevit, and William B Carter. Ultralight metallic microlattices. *Science*, 334(6058):962–965, 2011.

[7] Maarten Van Emmerik, Ari Rappoport, and Jarek Rossignac. Simplifying interactive design of solid models: a hypertext approach. *The Visual Computer*, 9(5):239–254, 1993.

[8] Justin Jang and Jarek Rossignac. Octor: subset selection in recursive pattern hierarchies. *Graphical Models*, 71(2):92–106, 2009.

[9] Alexander Pasko, Oleg Fryazinov, Turlif Vilbrandt, Pierre-Alain Fayolle, and Valery Adzhiev. Procedural function-based modelling of volumetric microstructures. *Graphical Models*, 73(5):165–181, 2011.

[10] Jonàs Martínez, Jérémie Dumas, and Sylvain Lefebvre. Procedural voronoi foams for additive manufacturing. *ACM Transactions on Graphics (TOG)*, 35(4):44, 2016.

[11] Jason Nguyen, Sang-in Park, and David Rosen. Heuristic optimization method for cellular structure design of light weight components. *Int. J. Precis. Eng. Manuf*, 14(6):1071–1078, 2013.

[12] Erhan Batuhan Arisoy, Suraj Musuvathy, Lucia Mirabella, and Edward Slavin. Design and topology optimization of lattice structures using deformable implicit surfaces for additive manufacturing. In *ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2015.

[13] Jarek Rossignac and Álvar Vinacua. Steady affine motions and morphs. *ACM Transactions on Graphics (TOG)*, 30(5):116, 2011.

[14] Stephen Daynes, Stefanie Feih, Wen Feng Lu, and Jun Wei. Optimisation of functionally graded lattice structures using isostatic lines. *Materials & Design*, 2017.

[15] Gershon Elber. Precise construction of micro-structures and porous geometry via functional composition. In *International Conference on Mathematical Methods for Curves and Surfaces*, pages 108–125. Springer, 2016.

[16] Stefano Gonella and Massimo Ruzzene. Homogenization of vibrating periodic lattice structures. *Applied Mathematical Modelling*, 32(4):459–482, 2008.

[17] Hongqing Wang, Yong Chen, and David W Rosen. A hybrid geometric modeling method for large scale conformal cellular structures. In *ASME Computers and Information in Engineering Conference, Long Beach, CA, Sept*, pages 24–28, 2005.

[18] I.E. Sutherland and Lincoln Laboratory. *Sketchpad: A Man-machine Graphical Communication System*. Reports // MIT. Massachusetts Institute of Technology, Lincoln Laboratory, 1963.

[19] Jarek Rossignac, Paul Borrel, and Lee Nackman. Procedure models for design and fabrication. *Automation in the Design and Manufacture of large Marine Systems. Proceedings of the 16th annual MIT Sea Grant College Program Lecture and Seminar, 1989, Hemisphere Publishing Corp., New York (USA), 1990*, pages 147–178, 1990.

[20] Jarek Rossignac, Paul Borrel, and Lee Nackman. Interactive design with sequences of parameterized transformations. *Intelligent CAD Systems*, 2:93–125, 1989.

[21] Carl S Daily, Daniel A Lees, and Dennis Donald McKitterick. Truss structure design, January 9 2001. US Patent 6,170,560.

[22] Suraj Ravi Musuvathy. Method for creating three dimensional lattice structures in computer-aided design models for additive manufacturing, January 7 2015. US Patent 20150193559A1.

[23] Hongqing Wang and David W Rosen. Parametric modeling method for truss structures. In *ASME Computers and Information in Engineering Conference*, 2002.

[24] Jason Nguyen, S Park, David W Rosen, Luis Folgar, and James Williams. Conformal lattice structure design and fabrication. In *Solid Freeform Fabrication Symposium, Austin, TX*, pages 138–161, 2012.

[25] Miguel Alejandro Aguilovalentin, Lauren L Beghini, Brett W Clark, William Roshan Quadros, Joshua Robbins, Brett Sneed, and Thomas Eugene Voth. "PLATO" environment for designing with topology optimization. Technical report, Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States); Sandia National Laboratories, Livermore, CA, 2015.

[26] Arpan Biswas, Vadim Shapiro, and Igor Tsukanov. Heterogeneous material modeling with distance fields. *Computer Aided Geometric Design*, 21(3):215–242, 2004.

[27] Yong Chen. 3d texture mapping for rapid manufacturing. *Computer-Aided Design and Applications*, 4(6):761–771, 2007.

[28] AO Aremu, JPJ Brennan-Craddock, Ajit Panesar, IA Ashcroft, Richard JM Hague, Ricky D Wildman, and Christopher Tuck. A voxel-based method of constructing and skinning conformal and functionally graded lattice structures suitable for additive manufacturing. *Additive Manufacturing*, 13:1–13, 2017.

[29] Michael F Ashby and Tianjian Lu. Metal foams: a survey. *Science in China Series B: Chemistry*, 46(6):521–532, 2003.

[30] Xingchen Liu and Vadim Shapiro. Sample-based synthesis of functionally graded material structures. *Journal of Computing and Information Science in Engineering*, 17(3):031012, 2017.

[31] Jay Kim and Jarek Rossignac. Screw motions for the animation and analysis of mechanical assemblies. *JSME International Journal Series C*, 44(1):156–163, 2001.

[32] Khristo N Boyadzhiev. Spirals and conchospirals in the flight of insects. *The College Mathematics Journal*, 30(1):23, 1999.

[33] Yong Tsui Lee and Aristides AG Requicha. Algorithms for computing the volume and other integral properties of solids. i. known methods and open issues. *Communications of the ACM*, 25(9):635–641, 1982.

[34] John R Baumgardner and Paul O Frederickson. Icosahedral discretization of the two-sphere. *SIAM Journal on Numerical Analysis*, 22(6):1107–1115, 1985.

[35] Wolfram Research, Inc. Mathematica, Version 11.2. Champaign, IL, 2017.

[36] Anthony George Maldon Michell. Lviii. the limits of economy of material in frame-structures. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 8(47):589–597, 1904.

[37] Matthew Thomas Wojciechowski et al. Determining optimal geometries of plane stress truss structures using numerically mapped principal stress trajectories. 2016.
[38] Xingchen Liu and Vadim Shapiro. Homogenization of material properties in additively manufactured structures. *Computer-Aided Design*, 78:71–82, 2016.
[39] Kelsey Kurzeja and Jarek Rossignac. Rangefinder: Accelerating ball interference queries against steady lattice. *Manuscript submitted for publication*, 2018.

Appendix A. Decomposition parameters

Given two similarity frames $\boldsymbol{F}_0 := \boldsymbol{F}(\mathrm{O}_0, I_0, J_0, K_0, s_0)$ and $\boldsymbol{F}_n := \boldsymbol{F}(\mathrm{O}_n, I_n, J_n, K_n, s_n)$, we compute the parameters describing the screw- and swirl-motions defined by a special decomposition of the similarity $\boldsymbol{S} = \boldsymbol{F}_n \cdot \boldsymbol{F}_0^{-1}$ into the commutative product, $\boldsymbol{D} \cdot \boldsymbol{R}$, of a dilation $\boldsymbol{D}$ by scaling vector m about fixed point F and a rotation $\boldsymbol{R}$, by angle α around an axis through F with tangent N as follows. For simplicity, we assume that $\boldsymbol{S}$ is not a translation nor a screw motion, as these special cases may be detected trivially and processed separately using previously published solutions.

We observe that, when a vector V is rotated about N, its tip traces an arc in a plane perpendicular to N. Hence, vector $\delta V = V^o(\alpha, N) - V$ lies in that plane, and so do all these vectors:

$$\delta I = I_n - I_0,\ \delta J = J_n - J_0,\ \delta K = K_n - K_0 \tag{A.1}$$

Since one of them may be null, we compute N using:

$$N = \underline{\delta I \times \delta J + \delta J \times \delta K + \delta K \times \delta I} \tag{A.2}$$

We define $T = O_0 O_n$ and compute α, m, and d as follows:

$$\begin{aligned} \alpha_1 = I_0^{\hat{N}} I_n,\ \alpha_2 = J_0^{\hat{N}} J_n,\ \alpha_3 = K_0^{\hat{N}} K_n \\ \alpha = max(\alpha_1, \alpha_2, \alpha_3) \end{aligned} \tag{A.3}$$

$$m = s_n / s_0 \tag{A.4}$$

$$d = T.N \tag{A.5}$$

To compute the fixed point F, we define $V_0 = FO_0$ and $V_n = FO_n$. Hence, we have:

$$V_n = V_0 + T \tag{A.6}$$

We define $K = N$ and compute I and J forming an orthonormal triplet (I, J, K). We then write V_0 and its rotated version as follows :

$$V_0 = xI + yJ + zK,\ and \tag{A.7}$$

$$V_0^\circ(\alpha, N) = (cx - sy)I + (cy + sx)J + (V_0 - xI - yJ) \tag{A.8}$$

where,

$$\begin{aligned} x = V_0.I, y = V_0.J, z = V_0.K,\ and \\ c = \cos(\alpha),\ s = \sin(\alpha) \end{aligned}$$

For swirl motion, $V_n = mV_0^\circ(\alpha, N)$. Using A.8, we solve for x and y. Further, for swirl motion $(V_n \cdot K)/(V_0 \cdot K) = m$, we use Eq. A.6 to solve for z:

$$\begin{aligned} x = \frac{u(mc-1) + v(ms)}{(mc-1)^2 + (ms)^2},\ y = \frac{v(mc-1) - u(ms)}{(mc-1)^2 + (ms)^2} \\ z = \frac{w}{m-1},\ where\ u = T \cdot I, v = T \cdot J, w = T \cdot K \end{aligned} \tag{A.9}$$

Then, the fixed point is:

$$\mathrm{F} = \mathrm{O}_0 - (xI + yJ + zK) \tag{A.10}$$