CONSTRUCTIVE LATTICE GEOMETRY

A Dissertation
Presented to
The Academic Faculty

By

Kelsey Kurzeja

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing
Area of Geometry, Graphics, and Animation

Georgia Institute of Technology

May 2021

CONSTRUCTIVE LATTICE GEOMETRY

Thesis committee:

Prof. Jarek Rossignac
School of Interactive Computing
Georgia Institute of Technology

Prof. Athanassios Economou
School of Architecture
Georgia Institute of Technology

Prof. Greg Turk
School of Interactive Computing
Georgia Institute of Technology

Dr. Suraj Musuvathy
nTopology

Prof. Sehoon Ha
School of Interactive Computing
Georgia Institute of Technology

Date approved: April 14, 2020

ACKNOWLEDGMENTS

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.

I give massive thanks to my advisor, Prof. Jarek Rossignac, for contributing an enormous amount of time to my learning and to our work over the years. I have learned so much from him that I know I will use for the rest of my life.

I thank Prof. Greg Turk, Prof. Sehoon Ha, Prof. Thanos Economou, and Dr. Suraj Musuvathy for using their time and energy to review my thesis and thesis defense.

I appreciate and thank everyone that I had the opportunity to interact with at Georgia Tech. I especially thank the members of the MAGIC Lab, the Geometry Group, and the Shape Computation Lab for countless hours of ideas and discussion, and I must give a special thanks to Ashish Gupta, Mukul Sati, and Sarang Joshi, who all worked closely with me in the lab for several years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

BeCOTS Bent Corner-Operated Tran-Similar

BIQ Ball Interference Query

CAD Computer Aided Design

CLG Constructive Lattice Geometry

COTS Corner-Operated Trans-Similar

CSG Constructive Solid Geometry

DoF Degrees of Freedom

GPU Graphics Processing Unit

GUI Graphical User Interface

IPC Integral Property Calculation

LiL Lattice-in-Lattice

PLE Programmed Lattice Editor

PMC Point Membership Classification

TMI Trans-Möbius Interpolant

SUMMARY

Lattice structures are widespread in product and architectural design. Recent work has demonstrated the printing of nano-scale lattices. However, an anticipated increase in product complexity will require the storage, processing, and design of lattices with orders of magnitude more elements than current Computer-Aided Design (CAD) software can manage.

To address this, we propose a class of highly regular lattices called Steady Lattices, which due to their regularity, provide opportunities for highly compressed storage, accelerated processing, and intuitive design. Special cases of steady lattices are also presented, which provide varying degrees of compromise between design freedom and geometric regularity. For example, the commonly used regular lattices, which provide little design freedom but offer maximum regularity, are the least general form of steady lattice. We propose the 2-directional, Bent Corner-Operated Trans-Similar (BeCOTS) lattices as a useful compromise between regular lattices and fully general steady lattices. A BeCOTS lattice may be controlled by four non-coplanar points, which represent four corners of the lattice. The Trans-Similar property ensures that a BeCOTS lattice is composed of groups of beams such that each consecutive pair of groups of beams along a particular direction is related by the same similarity. Trans-Similarity also enables constant-time queries such as surface area calculation, volume calculation, and point-membership classification.

We take advantage of the regularity in steady lattices to efficiently produce and query highly complex lattice structures that we call Constructive Lattice Geometry (CLG), where CLG is an extension of traditional Constructive Solid Geometry (CSG). CLG models are periodic CSG models for which regular patterns of primitives are combined into many repeating CSG microstructures that are ultimately combined into one CSG macrostructure. We provide strategies for designing and processing recursively defined CLG models to enable the creation of CLG models composed of smaller CLG models. Parameterized steady

lattices and CLG models may be defined by a few lines of code, which facilitates lazy (on-demand) evaluation, massively parallel processing, interactive editing, and algorithmic optimization.

CHAPTER 1
INTRODUCTION AND MOTIVATION

1.1 Motivation and problem statement

New additive manufacturing techniques are enabling the fabrication of structures with unprecedentedly complexity. These provide an opportunity for engineering structures and materials with useful and novel physical properties [39] including light weight [40], recoverability from compression [40], energy absorption [11], and negative Poisson's ratio [48].

Standard Computer Aided Design (CAD) software is not capable of modeling on the scale of the microstructure for a fully manufactured object [2]. In fact, modeling microstructure is not even necessary to exceed the capabilities of fully evaluated CAD models. A cubic meter of material with just 1 structural element per cubic millimeter contains one billion total elements.

To support the engineering of such complex structures, CAD tools must be able to represent them in their entirety and must provide intuitive user interfaces for manual structure design. Additionally, CAD tools must provide efficient analysis of physical properties to support structure optimization.

The naive way to represent a structure would be to explicitly list each individual element that composes the structure. The storage requirements for this representation grow in proportion with the number of elements. For example, a structure with $\overline{m} \times \overline{m} \times \overline{m}$ elements has a storage cost of $\mathbf{O}(\overline{m}^3)$. For $\overline{m} = 1000$, one billion elements must be explicitly stored.

Similarly, for many important queries, a naive implementation has a time cost that grows at the same rate of $\mathbf{O}(\overline{m}^3)$. For example, computing the volume of a structure by explicitly computing and summing the volume of each element, assuming the elements are

pairwise disjoint.

If the structure is truly random, then improving storage and query costs compared to these naive implementations may be difficult. However, engineered structures are typically not random. Patterns in the structures can be exploited to improve both representation and query costs.

Existing geometric modeling techniques exploit translation-based periodicity to efficiently represent a limited but useful set of engineered structures that also support efficient queries [32]. These techniques store a single geometric element and then create a $\overline{u} \times \overline{v} \times \overline{w}$ grid of identical copies of the element such that each consecutive pair of elements along a single direction is related by the same translation. A structure modelled with this approach is called a Regular Structure. Regular structures only require a constant amount of storage relative to the repetition counts $\overline{u}$, $\overline{v}$, and $\overline{w}$, and useful queries on regular structures can be performed with a constant amount of time relative to the repetition counts. However, regular structures are very limited in the variety of geometries that can be modeled. For example, we are interested in increasing design freedom to model curved and graded periodic structures such that consecutive pairs of elements are not necessarily related by a translation. Curved and graded periodic structures have been modeled by deforming regular structures [32], but we wish to avoid deforming our structural elements. Furthermore, querying a deformed structure is more computationally difficult than querying the regular structures, and we want our models to support efficient geometric queries.

1.2 Overview

The key idea of our approach is to start with a simple geometric element (which we may call the **template**) that is to be repeated (possibly in multiple directions) to form an unconnected large-scale pattern of elements. The repetition of the element is such that the transformation between consecutive pairs along a direction of elements is constant, so all elements can be reconstructed if only the template and the few transformations are stored.

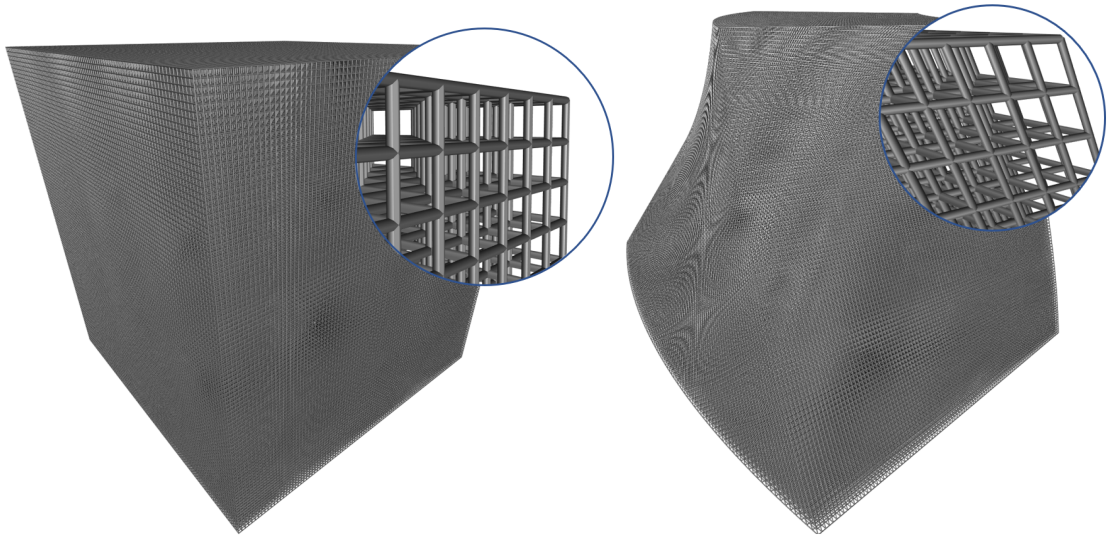


Figure 1.1: (Left) A regular lattice with 100^3 cells. (Right) A similarity steady lattice with 100^3 cells. Note that the balls and beams are not deformed in either lattice (i.e., the balls are round and the beams are conical frustums). A zoom for both lattices is shown in the top-right corners.

Note that a transformation between consecutive elements need not be a translation. Finally, neighboring elements are connected (with beams) using the same process for each element, so the connectivity information from only one element to its neighbors must be stored. Figure 1.1-Left shows an example of this approach when the consecutive transformations are translations. Figure 1.1-Right shows a more general example where the consecutive transformations are general similarities instead of translations.

Besides the reduced storage cost, the particular way in which we repeat the geometry, which we refer to as similarity steady, allows for accelerated queries, which may improve structure optimization and design.

Of course, this representation requires that the designed structure be periodic. Structures such as stochastic foams [26] and those resulting from topology optimization are typically not periodic [3]. Hence, it makes no sense to try to represent aperiodic structures using a template and repetition recipe, so other representations may be more suitable. For example, Voronoi foams have been used successfully to represent some stochastic foams [26].

The queries we primarily address are surface area calculation, volume calculation, and Point Membership Classification (PMC). A surface area calculation query returns the surface area of either the entire structure or of a selected portion of it. Calculating surface area is useful for optimizing heat transfer. Similarly, a volume calculation query returns the volume of either the entire structure or of a selected portion of it. Calculating volume is useful for optimizing material use and weight. We group the surface area calculation and volume calculation into one query group, **Integral Property Calculation (IPC)**, because, for our structures, both have the same acceleration strategies and therefore the same computational complexity.

A **Point Membership Classification (PMC)** query takes as input a point Q and returns true if and only if Q is contained by the queried structure. PMC may be useful for voxelization and for determining where to place material for 3D printing. We also discuss a useful generalization of PMC called the **Ball Interference Query (BIQ)** in chapter 7. A BIQ takes as input a query ball Q and returns true if and only if Q has a non-empty intersection with the queried structure. A BIQ with a Q of radius 0 represents a PMC query. The BIQ may be used to implement distance queries and ray intersection queries.

We validate the proposed approach experimentally, using Graphical User Interface (GUI) prototypes for creating and editing complex, periodic structures, using key queries, and discussing asymptotic and practical performance improvements.

1.3 Contributions

The goal set forth for this work was to explore novel ideas that may help us to improve our ability to design, optimize, and print large material micro-structures of unprecedented complexity.

We achieved this by proposing and validating through prototype implementations: (1) concise, parameterized, procedural representations for such structures, (2) an interactive design environment for creating and editing them, (3) and efficient algorithms that support

queries that are important for analysis, optimization, and manufacturing.

The novelty of the proposed solution includes: (1) the proposal of a new combination of simple, previously known concepts and (2) the invention of new concepts and algorithms that simplify and/or accelerate the representation, design, and processing mentioned above.

Some of the material presented in this thesis has appeared in the following peer-reviewed conferences and journal papers, and the work was performed in collaboration with listed colleagues.

- Programmed-Lattice Editor and accelerated processing of parametric program-representations of steady lattices (Ashish Gupta, Kelsey Kurzeja, Jarek Rossignac, George Allen, Pranav Srinivas Kumar, Suraj Musuvathy) [14]
- RangeFinder: Accelerating ball-interference queries against steady lattices (Kelsey Kurzeja, Jarek Rossignac) [19]
- BeCOTS: Bent Corner-Operated Tran-Similar Maps and Lattices (Kelsey Kurzeja, Jarek Rossignac) [20]
- CHoCC: Convex Hull of Cospherical Circles and Applications to Lattices (Yaohong Wu, Ashish Gupta, Kelsey Kurzeja, Jarek Rossignac) [47]

We list specific contributions along with relevant chapters and citations of relevant published work:

- The introduction of the multidirectional similarity steady patterns, maps, and fields (chapter 2 and [14]).
- The application of similarity steady patterns to modeling rectified, periodic lattices (chapter 3, chapter 6, [14], and [47]).
- Acceleration of integral property calculations (IPCs) to $\mathbf{O}(\overline{u} \times \overline{v})$ time for $\overline{u} \times \overline{v} \times \overline{w}$ steady patterns and lattices (chapter 3, chapter 6, and [14])

- The RangeFinder algorithm for accelerating Point Membership Classification (PMC) queries and Ball Interference Queries (BIQs) to $\mathbf{O}(\overline{u} \times \overline{v})$ time for $\overline{u} \times \overline{v} \times \overline{w}$ similarity steady patterns and lattices. Special cases can be further accelerated to $\mathbf{O}(\overline{u})$ and $\mathbf{O}(1)$ (chapter 7 and [19]).

- The introduction of Bent Corner-Operated Tran-Similar (BeCOTS) maps, which are a special case of 2-directional similarity steady maps and are a 3D, non-planar generalization of the planar COTS maps [35] (chapter 4 and [20]).

- The application of BeCOTS maps to the modeling of BeCOTS patterns and lattices (chapter 4 and [20]).

- Improved PMC and IPC queries to constant-time for BeCOTS patterns and lattices (chapter 4 and [20]).

- Constant-radius BeCOTS lattices, which are a novel application of BeCOTS to the modeling of lattices for which all beams have the same thickness and all beams may be joined by congruent connectors. This may be useful for reducing manufacturing costs of some architectural structures (chapter 4 and [20]).

- The BeCOTS Stacks generalization of BeCOTS maps and lattices to preserve some of the useful properties of BeCOTS lattices while modeling 3-directional structures. In particular, BeCOTS stacks may support constant-time PMC queries (chapter 6 and [20]).

- Filter-based methods of modeling hierarchical lattices by procedurally removing beams from an initial, non-hierarchical lattice. This filtering work is unpublished but was done in collaboration with Ashish Gupta (chapter 9).

- The Programmed Lattice Editor (PLE) for designing parameterized and programmed models of steady lattices through a code and graphical user interface combination (chapter 6 and [14]).

- We introduce Constructive Lattice Geometry (CLG), which is a extension of Constructive Solid Geometry (CSG) to support the design of periodic structures. CLG is also generalization of BeCOTS lattices to support the creation of periodic lattice structures where each beam is a more complex solid than just a ball or a cone-beam (chapter 5).

- We introduce Recursive CLG as a generalization of CLG for which each beam of a CLG structure may itself be a CLG structure (chapter 10).

- We propose ideas for how the similarity steady patterns, maps, and lattices may be generalized by using Möbius transformations in place of similarities (chapter 11).

- We introduce the Trans-Möbius Interpolant (TMI) generalization of COTS and special case of Möbius steady slab lattices (chapter 11).

CHAPTER 2
PRELIMINARIES

2.1 Lattices

2.1.1 Balls, beams, and hubs

Modeling with lattices allows the creation of complex geometry and topologies from simple primitives. These simple primitives simplify structure representation and analysis.

We define a **lattice** as the union of a set of nodes and a set of beams. A **node** is a ball defined by its center and radius. And, for simplicity, a **beam** is defined as the convex hull of two nodes that it connects. However, for some applications, more complex beam shapes, such as a QUADric-Of-Revolution (QUADOR), may also be used [13].

This definition of a beam, as the convex hull of two nodes, yields beams that are a union of two balls and of the conical frustum that smoothly connects the balls. Hence, we refer to these beams as **cone beams**. The conical frustum degenerates into a cylinder when the balls have equal radius.

Defining a lattice to be the union of nodes and beams is often sufficient for most applications. However, in chapter 5 we propose a useful generalization. It is sometimes useful to decompose a lattice into disjoint solids. For example, to facilitate computing the volume or surface area of an entire lattice. However, the beams of a lattice are not pairwise disjoint, and several beams may meet at a node, where the intersection between two or more beams may be complex.

So, we choose to decompose a lattice into disjoint solids called **hubs**. A hub is defined as the union of a node with all of the **half-beams** incident on it, where a half-beam is a portion of a beam obtained when the beam is cut by the plane equidistant from the beam's nodes. We define a **clean lattice** to be a lattice for which the interiors of all hubs are

pairwise disjoint, and we assume our lattices are clean. Decomposing a clean lattice into disjoint hubs is relatively simple, because neighboring hubs are connected only by a disk interface (a beam cross section). Note that we have also previously defined a half-beam as the cutting of a beam with the radical plane of its nodes.

2.1.2 Periodic lattices

The implicit definition of a beam in terms of two nodes allows the connectivity and the global geometry of a lattice to be defined separately. By specifying a set of beams as connecting certain nodes, the connectivity of a lattice may be preserved even when some of its nodes have been transformed. Similarly, the global geometry may be preserved when local connectivity is changed. By extension, the periodicity in connectivity between nodes and the periodicity of node geometry do not rely on each other and are defined separately.

A **periodic lattice** is a lattice for which both the connectivity and the node geometry is periodic. This thesis is primarily concerned with periodic lattices, so unless stated otherwise, the term lattice will refer to a periodic lattice.

The nodes of a periodic lattice are organized into an array N of $\overline{u}\times\overline{v}\times\overline{w}$ **node-groups**, where each integer $\overline{u}$, $\overline{v}$, and $\overline{w}$ is a **repetition count**. The total number of node-groups is $\overline{u}\,\overline{v}\,\overline{w}$. Each node-group contains the same number $\overline{n}$ of nodes, so a node in group $N[i,j,k]$ with ID n may be referenced as $N[i,j,k,n]$. We refer to the hub constructed on node $N[i,j,k,n]$ as $H[i,j,k,n]$. Similarly, we may refer to the union of hubs in a node-group $N[i,j,k]$ as the **hub-group** $H[i,j,k]$.

A lattice organized into an array of $\overline{u}\times\overline{v}\times\overline{w}$ node-groups is referred to as a **brick**. As special cases, a lattice organized into an array of $\overline{u}\times\overline{v}$ node-groups is a **slab** and a lattice organized into an array of $\overline{u}$ node-groups is a **row**. Alternatively, brick, slab, and row lattices may be referred to as **3-directional**, **2-directional**, and **1-directional**, respectively.

The beams of a periodic lattice are organized into a list B of $\overline{b}$ **beam-patterns**. For example, the statement $\mathrm{B}[b] = \textbf{BeamPattern}(n_x, i_\Delta, j_\Delta, k_\Delta, n_y)$ means that the b^{th} beam-

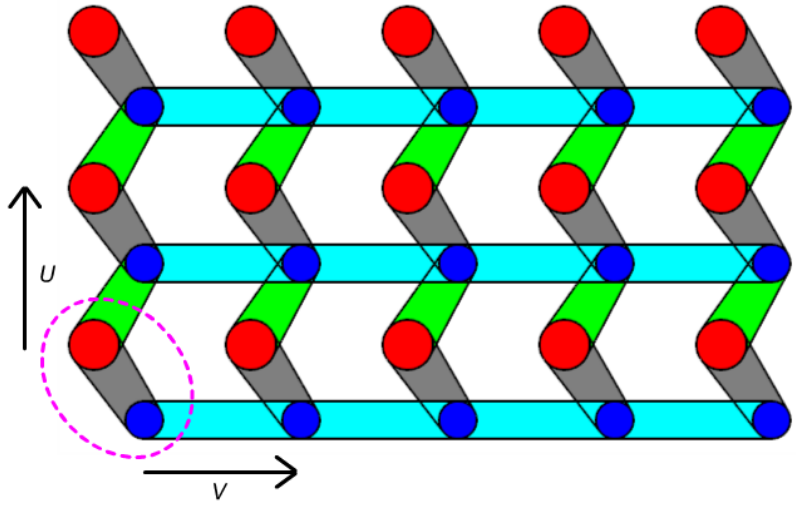


Figure 2.1: A regular slab lattice with two nodes (red and blue) per node-group. The grey beams connect nodes within a node-group, and the cyan and green beams connect nodes between node-groups. Beams with the same color belong to the same beam-pattern.

pattern of B is the set of beams that connects all pairs of nodes $\mathrm{N}[i, j, k, n_x]$ and $\mathrm{N}[i + i_\Delta, j + j_\Delta, k + k_\Delta, n_y]$ for all valid (i, j, k) triplets such that both nodes exist.

An example slab is shown in Figure 2.1 with two nodes per node-group and three beam-patterns.

For clarity and because of their exceptional properties, this thesis primarily focuses on slabs. But, we also discuss extensions to bricks in chapter 6. Our statements about slabs generally extend easily to bricks. We will state when statements made about slabs do not extend to bricks.

2.1.3 Regular slabs

A common type of periodic lattice is the **regular slab** (or its extension, the **regular brick**) [32]. A regular slab is a 2-directional periodic lattice for which each consecutive pair of node-groups along a particular direction is related by the same unit length translation, and the two translation unit vectors, $\vec{U} = <1, 0>$ and $\vec{V} = <0, 1>$, correspond to the two orthogonal directions.

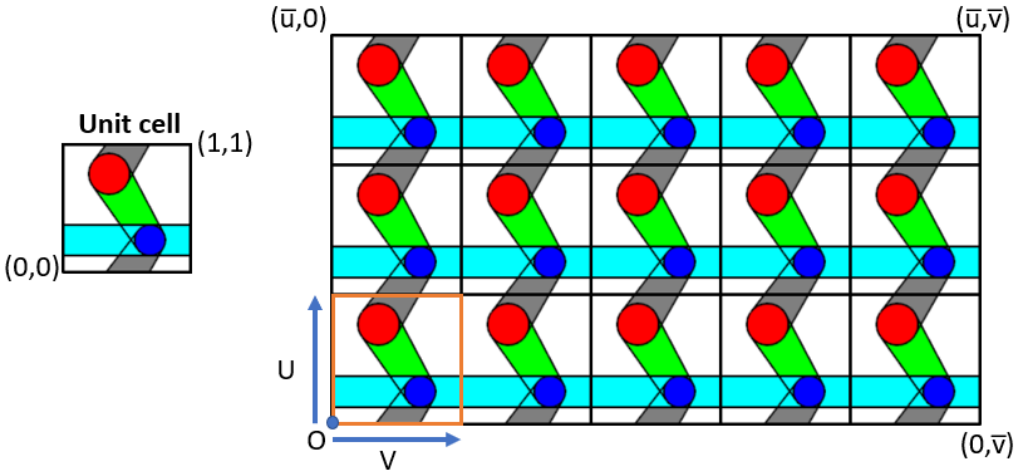


Figure 2.2: A regular lattice that is represented as a repetition of a single unit cell, where U and V are orthogonal unit-length translations. Although a unit cell typically contains most of the template node-group, the positioning of the unit cell is arbitrary. Here, the green beam is centered in the unit cell, but other choices of unit cell are possible. Some beams are cut at the grid boundary, and some applications treat the boundary cells as special cases and remove the cut beams.

The node-groups of a regular slab have the form:

$$\mathrm{N}[i,j] = \mathrm{N}[0,0] + i\vec{U} + j\vec{V} \tag{2.1}$$

A regular slab may be partitioned into a grid of $\overline{u} \times \overline{v}$ **cells** C, each of 1×1 units such that each pair of cells contains congruent sections of the lattice. Cell $\mathrm{C}[0,0]$ is called the **unit cell** and has a range of $[0,1) \times [0,1)$. Typically, the unit cell contains all or most of the **template node-group** $\mathrm{N}[0,0]$, so we associate the unit cell with the template node-group. We reference a cell with index pair (i,j) as $\mathrm{C}[i,j]$. See Figure 2.2.

One of the main benefits of modeling lattices using regular slabs is that they allow the storage cost to be reduced to a constant with respect to the repetition counts. Instead of storing all node-groups, only the template node-group must be stored, along with the two repetition counts. All other node-groups may be recovered using Equation 2.1. Pasko et al. use regular bricks to model highly complex lattices with a small storage cost [32].

Another benefit of regular slabs is that they enable the implementation of a constant-

time PMC query with respect to the repetition counts [32]. This is accomplished by exploiting the fact that the lattice geometry contained by each cell is congruent. A query point $Q = (x, y)$ may be mapped into the unit cell as $Q' = (x \bmod 1, y \bmod 1)$ such that Q and Q' have the same relationship with their corresponding cell geometry. So, PMC tests need only be performed between Q' and the geometry contained by the unit cell.

Integral properties such as volume and surface area may also be computed in constant time. For example, to compute the volume of a regular slab, simply multiply the total repetition count by the volume of the geometry contained by the unit cell.

Physical properties of regular lattices, for example the response to an applied force, may be efficiently approximated for regular lattices by material homogenization techniques, which for simulation purposes, replace a complex lattice-based material with a homogeneous material that has approximately equivalent physical properties [29]. This method depends on the regular repetition of cells throughout the simulated object.

The main drawback of modeling lattices using regular slabs is their lack of design freedom. All regular slabs have a box-shaped global geometry and a rectilinear-grid-based repetition. The limitation of having a global box-shape is sometimes addressed by "cutting" the box-shaped lattice into a different shape [4]. The cutting may be implemented with a Constructive Solid Geometry (CSG) intersection operation between the box-shaped lattice and a non-box-shaped solid. However, this only affects the global, boundary shape and not the local, internal shape of a lattice.

For simplicity, the formulation presented here assumes that each cell of a regular slab is a square. This formulation may easily be extended to allow rectangular cells [32].

2.1.4 Warped regular slabs

To address the main drawback of regular slabs, space warping techniques have been used to deform regular slabs both globally and locally. That is, space warping can modify both the boundary and the interior of a lattice. The result of such a deformation is a **warped**

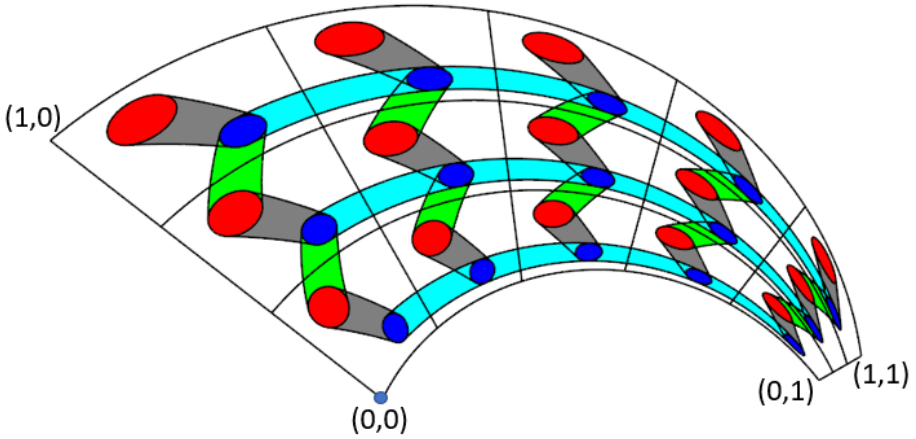


Figure 2.3: A warped regular lattice formed by a taper and bend warp of the regular lattice in Figure 2.1. Notice that the nodes are no longer circular and that the beams are no longer straight. The iso-curves of the slab map used to warp the lattice are shown.

regular slab.

Simple space warps, such as tapers, bends, and twists [6], were used by Pasko et al. to deform complex brick lattices [32]. These warps have efficient, closed-form inverses which preserve the ability to perform constant-time PMC queries on the warped lattices. The simple warps provide only a small amount of design freedom however and cannot model complex global or local shapes.

To model warped lattices with more complex global and local shapes, free-form deformations [42] may be used. For example, trivariate B-splines have been used for modeling microstructures [10]. Computing the inverse of free-form deformations is in general more difficult than for the simple space warps, which may make PMC less efficient and more difficult to implement. Iterative subdivision-based solvers may be used to compute the inverse of B-spline warps [33].

The primary drawback of modeling with warped regular lattices is that not only is the global shape deformed, but the local details throughout the lattice are also deformed. The local shapes in a regular lattice are all balls and cone beams, but after warping, the local shapes may become distorted. An example warped slab lattice, with deformed nodes and beams, is shown in Figure 2.3.

The deformation may have several negative effects on the ability to process the lattice. For example, computing the surface area or the volume of a lattice composed of balls and cone beams is already non-trivial, and the deformation only further complicates these computations, because the simplicity of the undeformed ball and cone composition helps to accelerate and improve the results for integral property computations on lattices [14]. Furthermore, regular lattices allow these integral property computations to be completed in constant-time. For a deformed lattice, this is generally not possible because the integral properties of the lattice cells are not related to each other in any simple way, so the integral properties must each be computed for each cell separately. We do however present, in chapter 4, a class of deformed slab lattices for which constant-time IPC is possible.

Lastly, for deformed lattices, predicting physical properties such as the stiffness of a manufactured part may be much more difficult than for regular lattices. To evaluate such properties in a regular lattice, homogenization techniques [29] may be used to exploit the fact that all parts of the lattice have the same local structure. However, any two different portions of a deformed lattice will likely have a different local structure, making the use of homogenization more difficult. Often, the entire lattice will have to be considered during analysis. And, since the local shapes are complex, full blown finite element analysis techniques must be used.

2.2 Space warps as maps

Warping is commonly formulated in terms of a map, so we formulate our warps as maps. When warping a 2D slab lattice, we use a **slab map** $\mathbf{M}(u, v) = P_2$ that takes the parameter-space coordinates (u, v) to a point P_2, either in 2D or 3D space depending on the application. So, a 2D regular slab lattice L is warped by transforming every point of L, with coordinates (u, v), by the map $\mathbf{M}(u, v)$. For simplicity, we typically assume the coordinates u and v are in the range $[0, 1)$ and that the regular lattice has been scaled to fit in the range. The parameter space is divided into $\overline{u} \times \overline{v}$ congruent cells, and the image of a

map is divided into $\overline{u} \times \overline{v}$ **tiles** that are the image of the cells. The sides of the tiles are the iso-curves of the map. An example slab map warping of the regular lattice from Figure 2.1 is shown in Figure 2.3 with the tiles of the map overlaid. Similarly, for a 3D brick lattice, a **brick map** $\mathbf{M}(u, v, w) = P_3$ takes (u, v, w) to a 3D point P_3.

Note that, for this map formulation of a warp, it does not make sense to try to warp a 3D slab lattice by a slab map, because the slab map requires at least 3 coordinates to describe each point of a 3D lattice. And, our formulation of brick maps (described in chapter 6) do not make sense for warping 3D slab lattices. However, the inability to warp a 3D slab lattice by these maps is not an issue for us, because, in practice, we never actually warp the balls or beams of a regular lattice. As will be discussed in chapter 3, we only use the form of a slab map as inspiration for constructing non-regular, steady slab lattices, and later, we take inspiration from the form of a brick map for constructing brick lattices.

When a map is planar, we choose to demonstrate it in 2D space, for simplicity. A lattice in such a 2D space (before it is warped) is a union of disks (nodes) and of isosceles trapezoids (beams). However, some of our maps may be non-planar, so we will of course demonstrate lattices based on these non-planar maps as a union of balls (nodes) and cone frustums (beams) in 3D space.

2.3 Rectified warped lattices

The most glaring flaw of the warped lattices is that the local geometry, of nodes and of beams, is no longer a union of balls and conical frustums. So, in fact, the warped lattices do not adhere to our original definition of a lattice.

It may be desirable to correct these local warps without removing the global warp. Such a correction results in a **rectified warped lattice**, which is a union of balls and of conical frustums produced by a **rectification process**. These meet our original definition of a lattice. Such a post-processing rectification has been used by Wang and Rosen for lattices warped by trivariate Bezier solids [45].

Unfortunately, it is not clear how to choose the appropriate rectification for a lattice under a general warp, let alone perform (or even define) the best rectification. In fact, it is not clear what it would mean to have a good rectification for a lattice under a simple warp, such as a bilinear map. Under a bilinear map, the local geometry may fold over itself and it may be stretched in wildly different ways for different parts of the map. In essence, the rectification problem is to fit a perfect ball to a distorted version of a ball, for all nodes in a lattice, then to join the rectified nodes with straight beams. This may violate some design constraints, for example, minimum beam length.

For a general warp, rectification is a global problem. For example, adjusting the length of one beam (to correct a design constraint that was violated after rectification) requires adjusting at least one node, which affects the lengths of other beams. However, some special lattices have a regularity that ensures global constraint satisfaction given constraint satisfaction on only a small piece of the lattice. For example, in a regular lattice, all beams are longer than a minimum length l if the beams associated with the unit cell are longer than l. In this thesis, we propose special types of lattices that have such a nice regularity and that are more general than the regular lattices.

2.4 2D Similarities

We use the term **similarity** to refer to a similarity transformation. Similarities are assumed to be orientation-preserving, unless stated otherwise. That is, we assume that similarities do not change the directions of angles.

Similarities play a central role in this thesis because they provide us with a means of generating more general regularities than the repeating translations, of regular lattices, without deforming the local geometry of nodes and beams.

2.4.1 A composition of 2D primitive similarities

In 2D, a similarity may either be 1) a **translation**, **Translation**$(\vec{T})$, by vector $\vec{T}$ or 2) a composition of a **rotation**, **Rotation**(α, F), by angle α around fixed point F with a **dilation**, **Dilation**(s, F), by scaling factor s on F.

The rotation and the dilation are commutative because they share the same fixed point F. For conciseness, we refer to the composition of a rotation and a dilation about the same fixed point as a **spiral transformation**,

$$\begin{aligned} \textbf{Spiral}(\alpha, s, F) &= \textbf{Dilation}(s, F) \circ \textbf{Rotation}(\alpha, F) \\ &= \textbf{Rotation}(\alpha, F) \circ \textbf{Dilation}(s, F) \end{aligned} \tag{2.2}$$

because a repeated application of a constant spiral transformation generates logarithmic spiral patterns.

A spiral transformation may degenerate into a pure rotation or a pure dilation. Likewise, a translation may degenerate into an identity when $\vec{T}$ is the zero vector.

We refer to translations, rotations, and dilations as **primitive similarities** because all similarities may be formulated as a composition of them.

2.4.2 Frame representation of 2D similarities

A similarity transformation may be represented by an orthogonal coordinate **frame** $\{O, \vec{X}, \vec{Y}\}$ where O is an origin point and $\vec{X}$ and $\vec{Y}$ are vectors with the constraint $\vec{Y} = \textbf{Rotate}(\pi/2) \circ \vec{X}$, where **Rotate**$(\alpha)$ denotes a vector rotation by angle α.

A point $P = (x, y)$ is mapped into the frame as $P' = O + x\vec{X} + y\vec{Y}$, where P' represents the transformation of P by a similarity.

Note, however, that this representation has an ambiguity such that the frame may represent multiple different spiral transformations that differ in rotation angle by an integer multiple of 2π.

2.4.3 Point pair representation of 2D similarities

A simpler yet equivalent representation of similarities is as a pair of points A and B. This representation has the benefit of facilitating an intuitive GUI for manipulating similarities.

The point pair representation can be converted into the frame representation by letting $O = A$, $\vec{X} = \overrightarrow{AB}$, and $\vec{Y} = \textbf{Rotate}(\pi/2) \circ \vec{X}$.

Likewise, the frame representation may easily be converted into a point pair representation by letting $A = O$ and $B = O + \vec{X}$.

2.4.4 Canonical representation of 2D similarities

Although the point pair representation doubles as a convenient control scheme, we prefer a different representation for computation.

When representing a translation, we simply store the translation vector $\vec{T}$. When representing a spiral transformation, we store the fixed point F, the rotation angle α, and the scaling factor s. This representation allows representing rotations of greater than 2π radians.

Additionally, this representation is convenient because it enables the following simple, closed-form formulas for computing real powers of similarities,

$$\textbf{Translation}(\vec{T})^t = \textbf{Translation}(t\vec{T}) \tag{2.3}$$

$$\textbf{Spiral}(\alpha, s, F)^t = \textbf{Spiral}(t\alpha, s^t, F) \tag{2.4}$$

With this formulation, computing the inverse of a power of a similarity is as simple as substituting t with $-t$. A continuous variation of t yields a continuously varying similarity.

2.4.5 Degrees of freedom in a 2D similarity

The 2D similarities have 4 Degrees of Freedom (DoF). This may be seen either 1) in the point pair where each point has 2 DoF or 2) in the canonical representation where F has 2 DoF and both α and s have 1 DoF.

2.4.6 Computing the 2D similarity between two point pairs

As a GUI control scheme, it is useful to compute the similarity between two ordered-pairs of points. We want the similarity **U** that takes pair $\{A, B\}$ to pair $\{C, D\}$, i.e. $\{C, D\} = \mathbf{U} \circ \{A, B\}$. We refer to such a similarity **U** as **Similarity**$(\{A, B\}, \{C, D\})$.

It is easy to detect and handle the case where **U** is a translation (when $\overrightarrow{AB} = \overrightarrow{CD}$ and $\vec{T} = \overrightarrow{AC}$). However, handling the spiral transformation case is not as straightforward, so here we focus on this case.

When $\overrightarrow{AB} \neq \overrightarrow{CD}$, $\mathbf{U} =$ **Spiral**(α, s, F) with

$$\begin{aligned} \alpha &= \textbf{Angle}(\overrightarrow{AB}, \overrightarrow{CD}) \\ s &= |\overrightarrow{CD}|/|\overrightarrow{AB}| \\ F &= \textbf{Dilation}(|\overrightarrow{BD}|/|\overrightarrow{BG}|, B) \circ \textbf{Rotation}(\textbf{Angle}(\overrightarrow{BG}, \overrightarrow{BD}), B) \circ A \\ &= \textbf{Spiral}(\textbf{Angle}(\overrightarrow{BG}, \overrightarrow{BD}), |\overrightarrow{BD}|/|\overrightarrow{BG}|, B) \circ A \end{aligned} \tag{2.5}$$

where $G = A + \overrightarrow{CD}$.

The forumlae for α and s are simply justified, respectively, as the change in angle and as the scaling factor between $\{A, B\}$ and $\{C, D\}$. The formula for F may be intuitively understood as the result of transforming A by the spiral transformation that takes triangle $\mathrm{T}_1 = \{A, B, G\}$ to triangle $\mathrm{T}_2 = \{F, B, D\}$. T_1 and T_2 are similar because **Angle**$(\overrightarrow{AB}, \overrightarrow{AG}) =$ **Angle**$(\overrightarrow{FB}, \overrightarrow{FD})$ and $|\overrightarrow{AB}|/|\overrightarrow{AG}| = |\overrightarrow{FB}|/|\overrightarrow{FD}|$. These relationships are visualized in Figure 2.4.

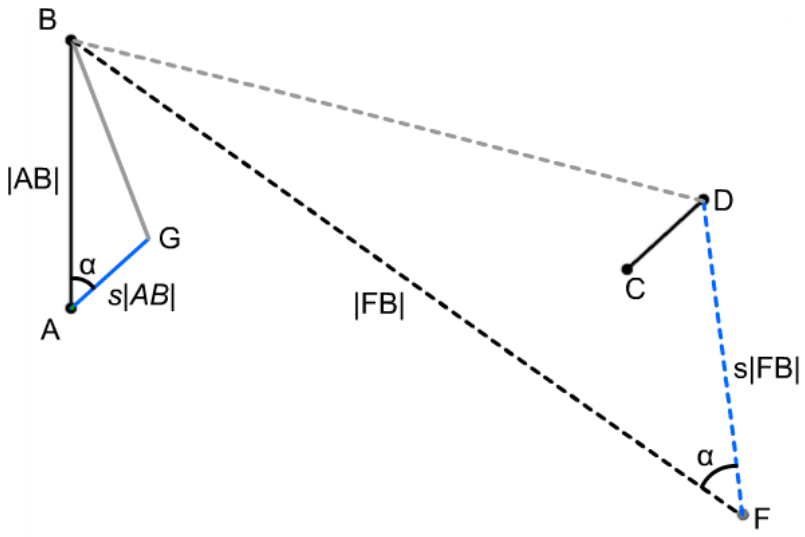


Figure 2.4: The triangles BAG and BFD are similar, which allows F to be constructed by rotating and dilating point A around B such that the same transformation would align BAG to BFD.

2.5 3D similarities

2.5.1 A composition of 3D primitive similarities

So far, we have several useful representations of planar similarities, and we have a useful control scheme for graphically specifying them. However, similarities need not be limited to the plane, so here we generalize the planar similarity representations, and related computations, to non-planar 3D similarities.

In 3D, like in 2D, all similarities may be formulated as a composition of the primitive similarities: 1) **Translation**$(\vec{T})$ by a 3D vector $\vec{T}$, 2) **Dilation**(s, F) by a scaling factor s and a 3D fixed point F, and 3) **Rotation**$(\alpha, Q, \vec{R})$ by a rotation angle α around the axis through point Q in direction $\vec{R}$.

Then, the composition of a 3D similarity has a strong parallel to the composition of a 2D similarity. A 3D similarity may either be

1. a **screw transformation**,

$$\begin{aligned} \textbf{Screw}(\vec{T}, \alpha, Q) &= \textbf{Translation}(\vec{T}) \circ \textbf{Rotation}(\alpha, \vec{Q}, \underline{\vec{T}}) \\ &= \textbf{Rotation}(\alpha, Q, \underline{\vec{T}}) \circ \textbf{Translation}(\vec{T}) \end{aligned} \tag{2.6}$$

2. a **swirl transformation**,

$$\begin{aligned} \textbf{Swirl}(s, \alpha, F, \vec{R}) &= \textbf{Dilation}(s, F) \circ \textbf{Rotation}(\alpha, F, \vec{R}) \\ &= \textbf{Rotation}(\alpha, F, \vec{R}) \circ \textbf{Dilation}(s, F) \end{aligned} \tag{2.7}$$

where we use underline to denote vector normalization, so $\underline{\vec{T}}$ is the normalized translation vector.

Notice that both cases are a commutative composition of two primitive similarities, one of which is always a rotation and the other is either a translation or a dilation. Interestingly, the translation may be viewed as a special case of a dilation where the fixed point approaches infinity, and this viewpoint unifies the screw case with the swirl case.

A screw may degenerate into a pure translation, and a swirl may degenerate into a pure dilation. Either may degenerate into a pure rotation or into an identity.

2.5.2 Frame representation of 3D primitive similarities

A 3D similarity may also be represented by an orthogonal coordinate frame $\{O, \vec{X}, \vec{Y}, \vec{Z}\}$, where O is a 3D origin point and $\vec{X}$, $\vec{Y}$, and $\vec{Z}$ are 3D vectors with the constraints $|\vec{X}| = |\vec{Y}| = |\vec{Z}|$, $\vec{X} \bullet \vec{Y} = 0$ and $\vec{Z} = \underline{\vec{X} \times \vec{Y}}$.

The point $P' = O + x\vec{X} + y\vec{Y} + z\vec{Z}$ represents the transformation of a 3D point $P = (x, y, z)$ by a similarity.

2.5.3 An attempt at a point pair representation of 3D primitive similarities

It would be useful to have an 3D analogue of the 2D point pair representation of similarities. As a naive attempt, we might try to compute a unique frame representation from only two points A and B. However, this does not work. If we let $O = A$ and $\vec{X} = \overrightarrow{AB}$, then we would quickly find that a valid $\vec{Y}$ could point in any direction orthogonal to $\vec{X}$ with length $|\overrightarrow{AB}|$.

The attempt at a point pair representation is lacking one degree of freedom, so a 3D similarity has 7 DoF. Although it is not as nice as the 2D case, this 7th DoF could be exposed to someone specifying a similarity, along with a 3D point pair, as a reasonable control scheme.

2.5.4 Canonical representation of 3D primitive similarities

Like for the 2D case, our canonical representation stores only the necessary parameters for either the screw or swirl, depending on the case, and enables the following closed-form formulas for computing powers of similarities,

$$\textbf{Screw}(\vec{T}, \alpha, Q)^t = \textbf{Screw}(t\vec{T}, t\alpha, Q)) \tag{2.8}$$

$$\textbf{Swirl}(s, \alpha, F, \vec{R})^t = \textbf{Swirl}(s^t, t\alpha, F, \vec{R})) \tag{2.9}$$

2.5.5 Computing the 3D similarity between two frames

Given two similar 3D frames, **A** and **B**, it is useful to be able to compute the similarity **S** that takes **A** to **B**. Rossignac and Vinacua solve the more general problem of computing the affine transformation between two 3D affine frames [36]. However, implementing the general affine solution is tricky, so we present a simpler (but non-trivial) solution for the useful special case of two frames related by a similarity. Here, we describe how to compute

the canonical representation where we explicitly store the screw and swirl parameters.

Let $\{O_A, \vec{X}_A, \vec{Y}_A, \vec{Z}_A\}$ represent frame **A**. Likewise let $\{O_B, \vec{X}_B, \vec{Y}_B, \vec{Z}_B\}$ represent frame **B**. **S** is a screw transformation if the scaling factor $s = |\vec{X}_B|/|\vec{X}_A| = 1$, otherwise **S** is a swirl transformation with scaling factor s. First, we consider the screw case. Then, we consider the swirl case, where we reuse some of the values computed for the screw case. The following computation of the direction of the axis of rotation $\vec{R}$ is inspired by Kim and Rossignac [18].

$$\begin{aligned}
\vec{N}_1 &= (\underline{\vec{X}_B} - \underline{\vec{X}_A}) \times (\underline{\vec{Y}_B} - \underline{\vec{Y}_A}) \\
\vec{N}_2 &= (\underline{\vec{Y}_B} - \underline{\vec{Y}_A}) \times (\underline{\vec{Z}_B} - \underline{\vec{Z}_A}) \\
\vec{N}_3 &= (\underline{\vec{Z}_B} - \underline{\vec{Z}_A}) \times (\underline{\vec{X}_B} - \underline{\vec{X}_A}) \\
\vec{R} &= \underline{\vec{N}_1 + \vec{N}_2 + \vec{N}_3}
\end{aligned} \tag{2.10}$$

Let P be any plane with normal $\vec{R}$, and let **Project**(Q, P) denote the closest projection of a point Q onto P. Consider the following points: $U_A =$ **Project**(O_A, P), $V_A =$ **Project**$(O_A + \vec{X}_A, \mathrm{P})$, $U_B =$ **Project**(O_B, P), and $V_B =$ **Project**$(O_B + \vec{X}_B, \mathrm{P})$. Notice that the points O_A and $O_A + \vec{X}_A$ project to the same point on P if $\vec{X}_A$ is parallel with $\vec{R}$. For this special case, consider the alternative projection choices $V_A =$ **Project**$(O_A + \vec{Y}_A, \mathrm{P})$ and $V_B =$ **Project**$(O_B + \vec{Y}_B, \mathrm{P})$. We compute the rotation angle α and the planar fixed point F_2 for the planar transformation **Similarity**$(\{U_A, V_A\}, \{U_B, V_B\})$. The computed F_2, when placed on plane P, is a point on the axis through which **S** rotates, and α is the rotation angle by which **S** rotates. $\vec{T} = (\overrightarrow{O_A O_B} \cdot \vec{R})\vec{R}$ is the translation component of the screw transformation.

Now we consider the swirl case, assuming that we have already computed everything needed for the screw case (except for $\vec{T}$, which is not needed). Although, the computed fixed point F_2 is the fixed point for the planar spiral, it is not the fixed point for the non-planar swirl. However, F_2 does lie on the axis of rotation, and the swirl fixed point F also

lies on the axis of rotation. We compute the swirl fixed point as $F = F_2 - h\vec{R}$, where $h = (\overrightarrow{O_A O_B} \cdot \vec{R})/(s-1)$.

If $\vec{R}$ is the zero vector, then the screw case degenerates into a translation and the swirl case degenerates into a dilation. The translation is by vector $\vec{T} = \overrightarrow{O_A O_B}$, and the dilation is by scaling factor s about fixed point $F = (sO_A - O_B)/(s-1)$.

2.6 Similarity steady patterns, maps, and fields

2.6.1 Steady patterns (rows)

As mentioned at the beginning of this section, similarities are central to this thesis because they provide a method of generating useful regularities in non-regular lattices.

Consider an initial shape $\mathrm{P}[0]$. If we apply a similarity $\mathbf{U}$ to it, we get $\mathrm{P}[1] = \mathbf{U} \circ \mathrm{P}[0]$. If we repeat this, we get $\mathrm{P}[2] = \mathbf{U} \circ \mathbf{U} \circ \mathrm{P}[0] = \mathbf{U}^2 \circ \mathrm{P}[0]$, and so on. We call the pattern P of shapes a **Similarity Steady Pattern**, where the i^{th} instance of the pattern may be written as $\mathrm{P}[i] = \mathbf{U}^i \circ \mathrm{P}[0]$. We have provided a closed-form expression for computing powers of similarities, so this is a closed-form expression for generating any instance of a similarity steady pattern. Any arbitrary instance can be generated efficiently in constant time.

For conciseness, we will use **steady pattern** to refer to a similarity steady pattern, unless stated otherwise.

A steady pattern may be designed by two frames $\mathbf{A}$ and $\mathbf{B}$ with an initial shape $\mathrm{P}[0]$ registered to frame $\mathbf{A}$. Let **Similarity**$(\mathbf{A}, \mathbf{B})$ be the similarity that takes $\mathbf{A}$ to $\mathbf{B}$, so $\mathbf{U} =$ **Similarity**(A, B). Then, for a given repetition count $\overline{u}$, a steady pattern may be extrapolated as $\mathrm{P}[i] = \mathbf{U}^i \circ \mathrm{P}[0]$, for $i \in [0, \overline{u})$. However, this control scheme is not the most intuitive interface. When $\overline{u}$ is large, a small change of either $\mathbf{A}$ or $\mathbf{B}$ could result in a drastic change for the last shape $\mathrm{P}[\overline{u}]$ of the pattern.

A more intuitive control scheme using two frames would be to create a pattern that interpolates between $\mathbf{A}$ and $\mathbf{B}$. To do this, let $\mathbf{I} = \mathbf{U}^{1/(\overline{u}-1)}$ be a portion of $\mathbf{U}$ such that $\mathbf{I}^{\overline{u}-1} = \mathbf{U}$. Then, let the steady pattern be $\mathrm{P}[i] = \mathbf{I}^i \circ \mathrm{P}[0]$, for $i \in [0, \overline{u})$. With this interface,

each individual transformation between consecutive instances represents a relatively small change compared to **U**, allowing for manual manipulation without drastic or unintuitive changes.

2.6.2 Steady 2-patterns (slabs)

A steady pattern provides a 1-variable family of shapes with a nice regularity. To model slabs, we introduce here a 2-variable family extension of steady patterns called a **steady 2-pattern**. Let instance (i, j) of a steady 2-pattern P be defined as $\mathrm{P}[i, j] = \mathbf{V}^j \circ \mathbf{U}^i \circ \mathrm{P}[0, 0]$, for $i \in [0, \overline{u})$ and $j \in [0, \overline{v})$, where $\mathrm{P}[0, 0]$ is the initial shape, $\overline{u}$ and $\overline{v}$ are the repetition counts, and **U** and **V** are similarities.

P may be considered a steady 1-pattern of steady 1-patterns. To understand this, consider a steady 1-pattern with an initial shape equal to the union of the shapes $\mathrm{P}[i, 0] = \mathbf{U}^i \circ \mathrm{P}[0, 0]$, for all valid i, that is transformed by powers of **V**.

The layout of a steady 2-pattern can be represented and controlled by 3 frames **A**, **B**, and **C** and by repetition counts $\overline{u}$ and $\overline{v}$. Consider that the shapes in the steady 2-pattern are frames, which may later be replaced by some other shape, such as a ball or a beam. The three frames represent three of the four corners of the pattern such that $\mathbf{A} = \mathrm{P}[0, 0]$, $\mathbf{B} = \mathrm{P}[\overline{u} - 1, 0]$, and $\mathbf{C} = \mathrm{P}[0, \overline{v} - 1]$. Therefore, $\mathbf{U} = \textbf{Similarity}(\mathbf{A}, \mathbf{B})^{1/(\overline{u}-1)}$ and $\mathbf{V} = \textbf{Similarity}(\mathbf{A}, \mathbf{C})^{1/(\overline{v}-1)}$.

The layout of a 2D planar steady 2-pattern has 12 DoF, assuming the repetition counts are fixed. These 12 DoF account for the 3 frames at 4 DoF per frame. Similarly, the layout of a 3D non-planar steady 2-pattern has 21 DoF due to being controlled by 3 frames at 7 DoF per frame.

2.6.3 Steady 2-fields

In a steady 2-pattern, the initial shape $\mathrm{P}[0, 0]$ is transformed into its instances by a 2-variable family of similarities $\mathbf{S}(i, j) = \mathbf{V}^j \circ \mathbf{U}^i$. We call **S** a **steady 2-field** of similarities.

In general, **U** and **V** do not commute, so the i **U** transformations must be performed before the j **V** transformations.

We have defined steady 2-patterns in terms of a discrete number of instances. However, a steady 2-field may be defined in terms of continuous variables u and v such that $\mathbf{S}(u, v) = \mathbf{V}^v \circ \mathbf{U}^u$.

2.6.4 Steady 2-maps and 2-warps

Consider a continuous steady 2-field $\mathbf{S}(u, v)$. We define a **steady 2-map** $\mathbf{M}(u, v) = \mathbf{S}(u, v) \circ P$ to be a parameterized surface generated by carrying an origin point $P = \mathbf{M}(0, 0)$ by the steady 2-field **S**. A steady 2-map may be used to define a **steady 2-warp**, for which the iso-curves are distorted into helices or conchospirals.

2.6.5 Control of a 2D planar steady 2-map by 5 points

A 2D planar steady 2-map **M** in the domain $[0, 1) \times [0, 1)$ can be controlled by 5 points such that $A = P = \mathbf{M}(0, 0)$, $B = \mathbf{M}(1, 0)$, $C = \mathbf{M}(0, 1)$, $D = \mathbf{M}(1, 1)$, and $E = \mathbf{M}(.5, 0)$. To satisfy these constraints, we let $\mathbf{U} = \textbf{Similarity}(\{A, E\}, \{E, B\})^2$ and $\mathbf{V} = \textbf{Similarity}(\{A, B\}, \{C, D\})$. The 2D planar steady 2-map has 10 DoF, because it is controlled by 5 points at 2 DoF per point.

We cannot control a non-planar, 3D steady 2-map by 5 points. This is because 5 3D points only represent a total of 15 DoF, but a non-planar, 3D steady 2-map has 17 DoF (7 DoF per similarity **U** and **V** plus 3 DoF for the origin point P).

2.6.6 Control of a non-planar steady 2-map by by 3 frames

A non-planar, 3D steady 2-map can be controlled by 3 frames (**A**, **B**, and **C**), by considering the 3 frames as part of a steady 2-pattern (where $\overline{u} = 2$ and $\overline{v} = 2$). Then, the steady 2-field **S** of the steady 2-pattern may used to specify the steady 2-map $\mathbf{M}(u, v) = \mathbf{S}(u, v) \circ O$, where $O = \mathbf{M}(0, 0)$ is the origin of frame **A**. In this control scheme, the origins of **B** and

C also represent the corners $\mathbf{M}(1,0)$ and $\mathbf{M}(0,1)$ respectively.

Note that a 3D steady 2-pattern has 4 DoF more than a 3D steady 2-map (21 vs 17), so the steady 2-pattern (defined by 3 frames) encodes more information than needed to represent the steady 2-map. In other words, multiple different configurations of the 3 frames may represent the same steady 2-map. These extra DoF represent the steady 2-pattern's ability to encode a rotation and a dilation (relative to the origin point O) for the initial shape in the pattern

CHAPTER 3

STEADY SLAB LATTICES

In section 2.4, we introduced the steady 2-map, which we use here to produce steady 2-patterns of warped nodes and beams, which we call **steady warped slab lattices**. Then, we suggest a strategy for rectifying the warped nodes and beams to produce a **steady slab lattice**. Finally, we discuss the steadiness properties of the rectified nodes, beams, and hubs of steady slab lattices, and we discuss efficient algorithms for PMC queries and the IPC queries on steady slabs.

3.1 Steady warping of a regular slab lattice

Consider a regular slab lattice with $\overline{u} \times \overline{v}$ node-groups of the form $\mathrm{N}_T[i,j] = \mathrm{N}_T[0,0] + i\vec{U}_T + j\vec{V}_T$, for translation vectors $\vec{U}_T = <1/\overline{u}, 0>$ and $\vec{V}_T = <0, 1/\overline{v}>$. Then consider its warping by a steady 2-map $\mathbf{M}(u,v) = \mathbf{V}^v \circ \mathbf{U}^u \circ O$, where $O = \mathbf{M}(0,0)$ is the origin point and **U** and **V** are similarities. Figure 3.1 shows an example of a steady warped slab lattice with the iso-curves of the map **M** overlaid. Notice that the tiles of the map are not all pairwise similar. However, all tiles in a common row along the **V** direction are pairwise similar.

3.2 Rectifying a steady warped slab lattice

The form of the warp **M** suggests an interesting strategy for rectifying the nodes of a regular lattice warped by **M**. The idea is to simply replace the origin point O with an "origin node-group" (i.e., a rectified template node-group).

Imagine that we have already computed the rectified template node-group $\mathrm{N}_r[0,0]$ using some black box process that produces round balls or that a designer has directly specified

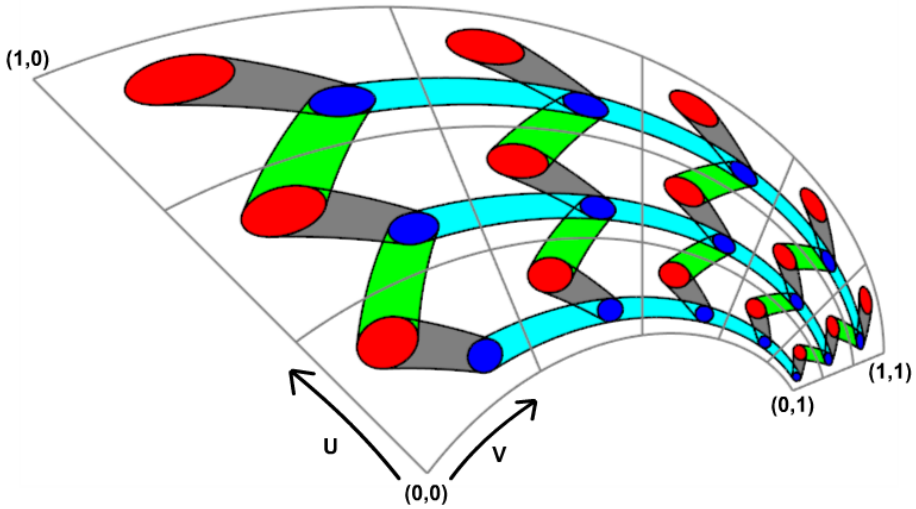


Figure 3.1: A steady warped slab lattice and the iso-curves of the steady slab map used to create it. Due to the warp, the nodes are not circular and the beams are not straight.

the rectified nodes of $\mathrm{N}_r[0,0]$ in the warped space. We may then transform the rectified template node-group via the steady 2-field to compute the rectified version of any other node-group $\mathrm{N}_r[i,j] = \mathbf{V}^j \circ \mathbf{U}^i \circ \mathrm{N}_r[0,0]$, for $i \in [0, \overline{u})$ and $j \in [0, \overline{v})$.

Given the set of rectified node-groups, the lattice's nodes may be connected with straight beams using the beam pattern approach described in subsection 2.1.2. An example of such a rectified lattice is shown in Figure 3.2.

3.3 Steadiness of the rectified nodes, beams, and hubs

The proposed rectification strategy works well because **U** and **V** are similarities. The steady 2-field is a composition of parameterized powers of similarities, the result of which is a parameterized similarity. For any given values for indices i and j, $\mathrm{N}_r[i,j]$ is $\mathrm{N}_r[0,0]$ transformed by a similarity. In other words, all rectified node groups are similar and their nodes are non-warped balls.

Even better, each rectified node-group $\mathrm{N}_r[i,j]$ belongs to a steady 1-pattern in both the **U** and **V** directions. This is easy to see for the **V** direction because Equation 3.1 has the

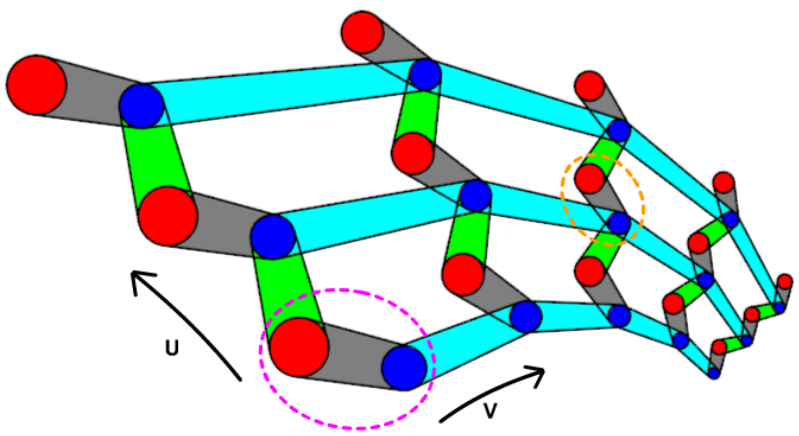


Figure 3.2: A similarity steady slab lattice that has a similar taper and bend as the warped lattice in Figure 3.1 but has been rectified to have circular nodes and straight beams. Node-group $\mathrm{N}[0,0]$ is circled in in dashed magenta and $\mathrm{N}[1,2]$ is circled in dashed orange. Notice that all node-groups are similar.

form of a steady 1-pattern when the i index is constant.

$$\mathrm{N}_r[i,j] = \mathbf{V}^j \circ (\mathbf{U}^i \circ \mathrm{N}_r[0,0]) = \mathbf{V}^j \circ \mathrm{N}_r[i,0] \tag{3.1}$$

To see that $\mathrm{N}_r[i,j]$ belongs to a steady 1-pattern in the **U** direction, consider that Equation 3.2, for a constant j, has the form of a steady 1-pattern that has been transformed by the similarity $\mathbf{S} = \mathbf{V}^j$.

$$\mathrm{N}_r[i,j] = \mathbf{V}^j \circ (\mathbf{U}^i \circ \mathrm{N}_r[0,0]) = \mathbf{S} \circ (\mathbf{U}^i \circ \mathrm{N}_r[0,0]) \tag{3.2}$$

Now, do a steady slab lattice's beams also belong to steady 1-patterns in the **U** and **V** directions? First consider the case of beams that connect nodes within the same node-group. If a node-group is transformed by either similarity **U** or **V** (or any other similarity), then all beams connecting nodes within the group are transformed by the same similarity. Thus, beams with both nodes in the same node-group do belong to a steady 1-pattern in both the **U** and **V** directions, because all node-groups belong to a steady 1-pattern in both the **U** and **V** directions.

Next, consider the case of beams that connect nodes in different node-groups. Such beams do belong to a steady 1-pattern in the **V** direction. Let beam B connect nodes $\mathrm{N}_1 = \mathrm{N}_r[i,j,x] = \mathbf{V}^j \circ (\mathbf{U}^i \circ \mathrm{N}_r[0,0,x]) = \mathbf{V}^j \circ \mathrm{N}_r[i,0,x]$ and $\mathrm{N}_2 = \mathrm{N}_r[m,n,y] = \mathbf{V}^n \circ (\mathbf{U}^m \circ \mathrm{N}_r[0,0,y]) = \mathbf{V}^n \circ \mathrm{N}_r[m,0,y]$, for some indices i, j, m, and n and for node IDs x and y. The beam $\mathbf{V}^v \circ \mathrm{B}$ is a beam that smoothly connects $\mathbf{V}^v \circ \mathrm{N}_r[i,j,x] = \mathrm{N}_r[i,j+v,x]$ and $\mathbf{V}^v \circ \mathrm{N}_r[m,n,y] = \mathrm{N}_r[m,n+v,y]$. Therefore, all beams belong to a steady 1-pattern in the **V** direction, because incrementing both the j and n indices by v results in a common transformation by $\mathbf{V}^v$.

However, beams that connect nodes in different node-groups do not belong to a steady 1-pattern in the **U** direction. Consider the possibility of transforming **B** by $\mathbf{U}^u$. This yields two nodes $\mathbf{U}^u \circ \mathbf{V}^j \circ \mathbf{U}^i \circ \mathrm{N}_r[0,0,x]$ and $\mathbf{U}^u \circ \mathbf{V}^n \circ \mathbf{U}^m \circ \mathrm{N}_r[0,0,y]$. In general, similarities do not commute, so these forms cannot be simplified, and the resulting nodes are not equivalent to $\mathrm{N}_r[i+u,j,x]$ and $\mathrm{N}_r[m+u,n,y]$, respectively. Therefore, beams that connect nodes in different node-groups do not generally belong to a steady 1-pattern in the **U** direction.

Finally, let us consider the steadiness of hubs in a steady slab. A hub is a union of half-beams incident on the same node, so the existing steadiness properties for beams can be trivially generalized to hubs. If a hub is constructed entirely from beams that connect nodes in one node-group, then the hub belongs to a steady 1-pattern in both the **U** and **V** directions. However, if the hub is constructed from one or more beams that connect nodes in different node-groups, then the hub generally only belongs to a steady 1-pattern in the **V** direction.

3.4 Strategy for rectifying the template node-group

Although it is easy to rectify an entire steady slab after the template node-group has been rectified, there is not an obviously correct method for rectifying the template node-group. This is because the image of a ball under the warp **M** may have a bean shape. This bean shape may be long and thin, or it may twist into a spiral shape. The goal of rectification is

to replace the bean with the ball that "best" represents some desired design constraints. We do not believe that there is any automated rectification that is best for all scenarios, so our system uses a simple rectification strategy and then we allow a human designer to tweak the result as desired.

The simple rectification strategy we use is to rectify each node in the template node-group independently of the others, where a single node is rectified by first arbitrarily picking four equally spaced points on the pre-warp node, mapping the four points by the warp map, and letting the rectified node be the sphere through the mapped points. Note that a different choice of four points on the pre-warp node will result in a different rectified node. An alternative strategy might be to sample many points on the pre-warp node and use a sphere-fitting procedure [1] on the warped sample points. Using more sample points may yield a better sphere fit, but we prefer to use the simple approach with four points, because we expect that no automatic rectification will be perfect and that a designer will prefer to manually adjust the result.

If some quantifiable design constraints are known, then an automated strategy could be implemented to optimize for a good choice of rectification for a warp **M**.

3.5 Point Membership Classification (PMC) on steady slab lattices

PMC can be performed on a steady slab lattice in $\mathbf{O}(\overline{u})$ time using our **RangeFinder** algorithm (see chapter 7). For slab lattices with many beams, this is a significant improvement from the naive approach, which tests PMC on every beam of the slab lattice and takes $\mathbf{O}(\overline{u}\,\overline{v})$ time. For example, in a slab lattice with 1000×1000 node-groups, the naive approach tests PMC on $1{,}000{,}000$ node-groups worth of beams. However, if the slab lattice is steady, RangeFinder can be used to reduce the work to testing PMC on an expected 1000 node-groups worth of beams.

Note that we do not include the number of beams $\overline{b}$ in the time-complexity of queries, because $\overline{b}$ tends to be small and does not grow with the size of a lattice.

The steady 2-map $\mathbf{M}$ does not have a known closed-form inverse. A closed-form inverse would enable constant-time PMC on non-rectified steady lattices. However, it is not clear how such an inverse could improve PMC on rectified steady lattices beyond $\mathbf{O}(\overline{u})$ time.

3.6 Integral Property Calculation (IPC) on steady slab lattices

IPC queries can also be performed on a steady slab lattices in $\mathbf{O}(\overline{u})$ time. In particular, we focus on surface area and volume calculations.

The naive approach computes either the surface area or the volume for each hub in the slab lattice and adds it to a running total, which takes $\mathbf{O}(\overline{u}\,\overline{v})$ time.

Our approach relies on the fact that all hubs belong to a steady 1-pattern in the $\mathbf{V}$ direction, so in the $\mathbf{V}$ direction, consecutive hubs differ by the constant scaling factor s_v of $\mathbf{V}$. As a consequence, the surface area of consecutive hubs in the $\mathbf{V}$ direction also differs by a constant factor ${s_v}^2$, and the volume differs by a constant factor ${s_v}^3$.

In fact, consider the parameterized union H_j of hub-groups $\mathrm{H}[i, j]$, for all $i \in [0, \overline{u})$. H_j belongs to a steady 1-pattern such that $\mathrm{H}_j = \mathbf{V}^j \circ \mathrm{H}_0$. So, if we have already computed the surface area $\mathbf{Area}(\mathrm{H}_0)$, then the surface area of H_j is $\mathbf{Area}(\mathrm{H}_j) = {s_v}^{2j}\,\mathbf{Area}(\mathrm{H}_0)$. Similarly, if we have computed the volume $\mathbf{Volume}(\mathrm{H}_0)$, then the volume of H_j is $\mathbf{Volume}(\mathrm{H}_j) = {s_v}^{3j}\,\mathbf{Volume}(\mathrm{H}_0)$. Both $\mathbf{Area}(\mathrm{H}_0)$ and $\mathbf{Volume}(\mathrm{H}_0)$ can be computed in $\mathbf{O}(\overline{u})$ time by computing and summing the respective values for hub-groups $\mathrm{H}[i, 0]$, for all $i \in [0, \overline{u})$.

In the case where $s_v = 1$, the total surface area is $\mathbf{Area}(\mathrm{H}) = \overline{v}\,\mathbf{Area}(\mathrm{H}_0)$ and the total volume is $\mathbf{Volume}(\mathrm{H}) = \overline{v}\,\mathbf{Volume}(\mathrm{H}_0)$.

Otherwise, the formulas for $\mathbf{Area}(\mathrm{H})$ and $\mathbf{Volume}(\mathrm{H})$ are geometric progressions, and sums of geometric progressions can be expressed in closed-form, yielding the following expressions,

$$\begin{aligned}\textbf{Area}(\mathrm{H}) &= \sum_{j=0}^{\overline{v}-1}\sum_{i=0}^{\overline{u}-1}\textbf{Area}(\mathrm{H}[i,j]) \\ &= \frac{1-{s_v}^{2\overline{v}}}{1-{s_v}^2}\sum_{i=0}^{\overline{u}-1}\textbf{Area}(\mathrm{H}[i,0]) \\ &= \frac{1-{s_v}^{2\overline{v}}}{1-{s_v}^2}\,\textbf{Area}(\mathrm{H}_0)\end{aligned}$$

$$\begin{aligned}\textbf{Volume}(\mathrm{H}) &= \sum_{j=0}^{\overline{v}-1}\sum_{i=0}^{\overline{u}-1}\textbf{Volume}(\mathrm{H}[i,j]) \\ &= \frac{1-{s_v}^{3\overline{v}}}{1-{s_v}^3}\sum_{i=0}^{\overline{u}-1}\textbf{Volume}(\mathrm{H}[i,0]) \\ &= \frac{1-{s_v}^{3\overline{v}}}{1-{s_v}^3}\,\textbf{Volume}(\mathrm{H}_0)\end{aligned}$$

3.7 IPC for the special case of steady 2-patterns

For steady 2-patterns and for special steady slab lattices that are a steady 2-pattern (for example if no beams connect nodes in different node-groups), IPC queries can be performed in $\mathbf{O}(1)$ time. Let s_u be the scaling factor of **U** and let s_v be the scaling factor of **V**. Assuming $s_u \neq 1$ and $s_v \neq 1$, the total area and the total volume of a steady 2-pattern P may be expressed as,

$$\textbf{Area}(\mathrm{P}) = \frac{1-{s_v}^{2\overline{v}}}{1-{s_v}^2}\frac{1-{s_u}^{2\overline{u}}}{1-{s_u}^2}\,\textbf{Area}(\mathrm{P}[0,0])$$

$$\textbf{Volume}(\mathrm{P}) = \frac{1-{s_v}^{3\overline{v}}}{1-{s_v}^3}\frac{1-{s_u}^{3\overline{u}}}{1-{s_u}^3}\,\textbf{Volume}(\mathrm{P}[0,0])$$

3.8 Handling edge cases

When engineering a lattice, it may be desirable to remove the beams that "stick out" of the edge if a lattice. I.e., remove the beams for which at least one node belongs to an out-of-range node-group. However, removing these beams means that some hub-groups on the boundary of the lattice will be different than the other hub-groups. Such edge cases must be kept in mind when implementing the above algorithms. For example, care must be taken to not compute the volume of beams that have been removed. Handling these edge cases is not elegant, so we do not explain it here, but the required changes do not change our stated asymptotic computational complexities.

CHAPTER 4

BECOTS SLABS

4.1 Motivation

The **Bent Corner-Operated Tran-Similar** (**BeCOTS**) **field** and **map**, discussed here, may be used to produce bent (non-planar) BeCOTS versions of the previously proposed planar **COTS lattices** [35] while retaining their tran-similarity property and the associated computational benefits.

The BeCOTS lattice presented here offers the same computational complexity (PMC and IPC queries in constant time) as the regular lattices and the planar COTS lattices but without their planarity constraint. It is controlled by four non-coplanar points, and hence has 12 DoF (i.e., one more than the planar COTS that it generalizes).

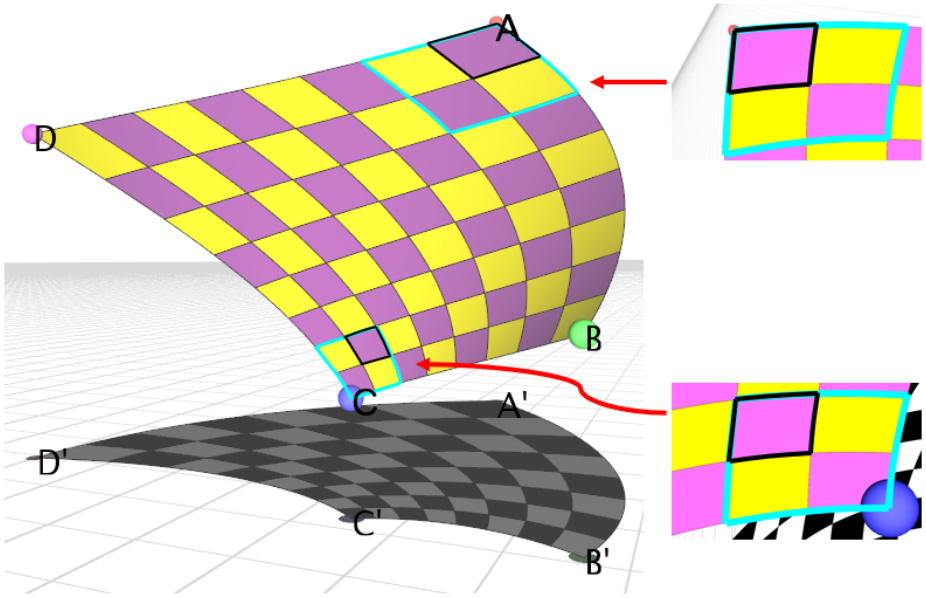


Figure 4.1: (Left) Range R of a BeCOTS map M, where the colored spheres indicate the 4 corners of R: $A = M(0,0)$ in red, $B = M(0,1)$ in green, $C = M(1,1)$ in blue, and $D = M(1,0)$ in magenta. (Right) Any two $\overline{i} \times \overline{j}$ blocks of tiles are similar (we outline 1×1 blocks in black and 2×2 blocks in cyan).

4.2 Corner-Operated Tran-Similar (COTS) maps

Before discussing our non-planar Bent Corner-Operated Tran-Similar (BeCOTS) maps, we first review the planar Corner-Operated Trans-Similar (COTS) maps [35]. Both COTS and BeCOTS are 2-directional, so here we simply use field, map, pattern, and lattice to refer to a 2-field, 2-map, 2-pattern, and slab lattice. The two most important properties of the COTS maps are being Corner-Operated and Tran-Similar.

We say that a field, **S**, (or a map, pattern, or lattice defined in terms of **S**) is **Corner-Operated** (**CO**) when **S** is fully controlled by four control-points, $\{A, B, C, D\}$, such that, $\mathbf{S}(0,0) \circ A = A$, $\mathbf{S}(0,1) \circ A = B$, $\mathbf{S}(1,1) \circ A = C$, $\mathbf{S}(1,0) \circ A = D$. Note that these four control-points are the corners of the range of a CO map (Figure 4.1), hence, we call them **corners**.

We say that a steady field, **S**, (or a map, pattern, or lattice defined in terms of **S**) is **Tran-Similar** (**TS**) if it maps translations into similarities. More formally, **S** is TS if and only if, for any vector $<u, v>$, there exists a similarity, **T**, such that, for any t, $\mathbf{S}(x + tu, y + tv) = \mathbf{T}^t \circ \mathbf{S}(x, y)$. Note that **S** is TS if and only if its similarities commute (i.e., when $\mathbf{V} \circ \mathbf{U} = \mathbf{U} \circ \mathbf{V}$). In that case, $\mathbf{V}^y \circ \mathbf{U}^x = \mathbf{U}^x \circ \mathbf{V}^y$.

The COTS map [35] takes parameter-pair (x, y) to a point $P = \mathbf{M}(x, y)$ in the plane. It maps the unit-square parametric domain D onto a range R that is bounded by four curved (logarithmic spiral) edges. The unique aspect of the COTS map is that $\mathbf{M}(x, y)$ may be expressed as $\mathbf{S}_2(x, y) \circ A$, in terms of a planar similarity steady field $\mathbf{S}_2(x, y)$ and corner $A = \mathbf{M}(0, 0)$. Finally, $\mathbf{S}_2(x, y)$ is the commutative product $\mathbf{U}_2^x \circ \mathbf{V}_2^y$ of non-trivial (i.e., not pure translation) planar similarities $\mathbf{U}_2$ and $\mathbf{V}_2$. The case where $\mathbf{U}_2$ and $\mathbf{V}_2$ are pure translations can be detected and handled as a special case. Similarity $\mathbf{U}_2 =$ **Similarity**$(\{A, B\}, \{D, C\})$ maps A to D and B to C and is represented by the triplet $<F, m_u, w_u>$, where F is the common fixed point of $\mathbf{U}_2$ and $\mathbf{V}_2$, m_u is the dilation factor,

and w_u is the rotation angle. $\mathbf{U}_2^x \circ A$ may be evaluated as

$$\mathbf{U}_2^x \circ A = \textbf{Rotation}_2(w, F) \circ \textbf{Dilation}_2(m, F) \circ A \tag{4.1}$$

where **Rotation**$_2$ is a 2D rotation by angle w around fixed point F and **Dilation**$_2$ is a 2D dilation by factor m about F. Similarity $\mathbf{V}_2 = \textbf{Similarity}(\{A, D\}, \{B, C\})$ maps A to B and D to C. All COTS parameters, $<F, m_u, w_u, m_v, w_v>$, may be computed from the four coplanar corners $\{A, B, C, D\}$ using closed-form expressions [35]. Hence, COTS is Corner-Operated, i.e., fully defined by the four corners $\{A, B, C, D\}$ of R. The pre-image, $(x, y) = \mathbf{M}^{-1}(Q)$, of a point Q may be computed in closed form. The COTS field is Tran-Similar. Therefore, the COTS map has uniform distortion (see [35] for the proof of Tran-Similarity and for a discussion of the benefits of tran-similarity and of distortion uniformity). Hence, regular cells of domain D map onto tiles that are all similar to each other. Several planar maps that are not Tran-Similar are compared in [23], to the Four Point Interpolant (FPI) map which is quasi-conformal but not TS.

4.3 3D generalization of COTS

We propose here a 3D generalization of the planar COTS field [35] and of the maps, patterns, or lattices defined in terms of such a field. The proposed BeCOTS field defines a two-parameter field of 3D similarities. A BeCOTS map (which is defined by a BeCOTS field) is not flat, but bent; its range lies on a curved surface controlled by the designer. We conjecture that this generalization will enable a variety of 3D applications in design, texturing, and animation.

Note that we often refer to a BeCOTS field even when the property of being Corner-Operated (CO) is not required. Perhaps the term BeTS field would be more accurate in such cases, but for simplicity, we do not use it. Similarly, when discussing planar maps, we may write COTS in place of TS.

A BeCOTS field (and hence a map, pattern, or lattice defined in terms of it) is Tran-Similar. Note that steady fields are not generally TS. Tran-Similarity has many benefits, such as accelerating IPC and PMC.

The isocurves of a BeCOTS map are conchospirals, a 3D generalization of logarithmic spiral. All lines in domain D map to conchospirals in the range R. The conchospirals may degenerate into lines or circles.

The COTS field is a commutative product of four transformations,

$$\begin{aligned}\mathbf{S}_2(x,y) = &\mathbf{Rotation}_2(xw_u, F) \circ \mathbf{Dilation}_2(m_u^x, F) \circ \\ &\mathbf{Rotation}_2(yw_v, F) \circ \mathbf{Dilation}_2(m_v^y, F)\end{aligned} \tag{4.2}$$

The BeCOTS field proposed here is a 3D extension of the COTS field. It is a commutative product of 3D versions of these four transformations. The key difference is that, in the formulation of BeCOTS, the 3D rotations have not only the same fixed point, but also the same axis. It is remarkable that their parameters may be computed from only the four non-coplanar corners $\{A, B, C, D\}$ of the range R of a map defined by that field.

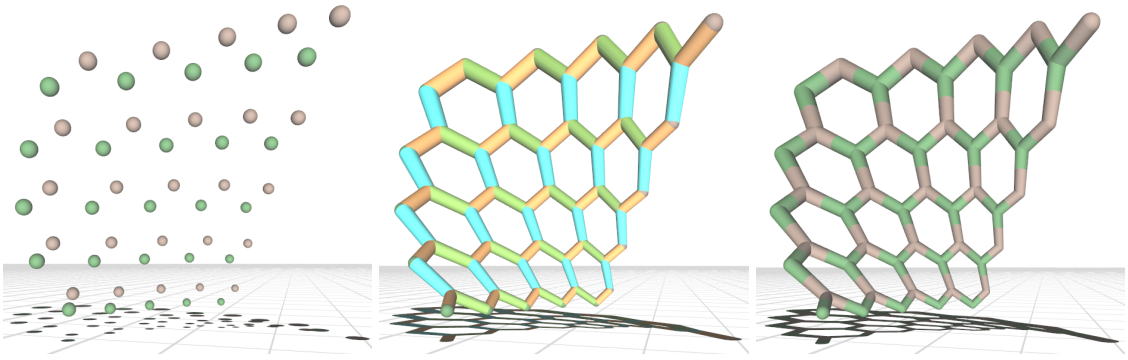


Figure 4.2: (Left) Two patterns (green and brown) of nodes, (Center) and three patterns (lime, cyan, orange) of cone-beams each smoothly connecting two balls. These beams form a lattice having these balls as nodes. (Right) This lattice is the union of two patterns of hubs, which are each the union of a node with its incident half-beams. This lattice is clean (the interiors of the hubs are pairwise-disjoint).

4.3.1 Raising from and projecting to a COTS map

Consider a planar but 3D formulation of the COTS field. It is the commutative product of four 3D transformations:

$$\begin{aligned}\mathbf{S}(x,y) = &\textbf{Rotation}(xw_u, F, \vec{T}) \circ \textbf{Dilation}(m_u^x, F) \circ \\ &\textbf{Rotation}(yw_v, F, \vec{T}) \circ \textbf{Dilation}(m_v^y, F)\end{aligned} \tag{4.3}$$

Here $\vec{T}$ is the (unit vector) direction of the axis of rotation, **Rotation**$(w, F, \vec{T})$ is a 3D rotation by angle w around the axis through F with direction $\vec{T}$, and **Dilation**(m, F) is a 3D dilation (uniform scaling) by factor m about F. The expression for **S** may be simplified to

$$\mathbf{S}(x,y) = \textbf{Rotation}(xw_u + yw_v, F, \vec{T}) \circ \textbf{Dilation}(m_u^x m_v^y, F) \tag{4.4}$$

When point A lies on the plane π that passes through F and has normal $\vec{T}$ (i.e., when $\overrightarrow{FA} \cdot \vec{T} = 0$), this field degenerates to a planar COTS map $\mathbf{M}(x,y) = \mathbf{S}_2(x,y) \circ A$.

When $\overrightarrow{FA} \cdot \vec{T} \neq 0$, the above formulation produces the **bent** (non-planar) range R of a BeCOTS map (see Fig. Figure 4.3).

This BeCOTS map is the **Raised** version, **Raise**$(z) \circ \mathbf{M}(x,y)$, of the COTS map **M**. **Height-parameter** $z = \overrightarrow{FA} \cdot \vec{T}$ may be used to control the amount of raising.

The range of the BeCOTS maps lies on a cone K with apex F and axis direction $\vec{T}$. As justification, simply remember that the dilation operations move a point P toward or away from F, leaving the image somewhere on the ray Y from F through P, and that the rotation operations rotate the ray Y around the axis through F with direction $\vec{T}$, sweeping out the cone K.

The fact that the range of a BeCOTS map lies on a cone makes it a developable surface, which may benefit applications in manufacturing and architecture.

Although a BeCOTS map may be controlled by four corner points, four points are not

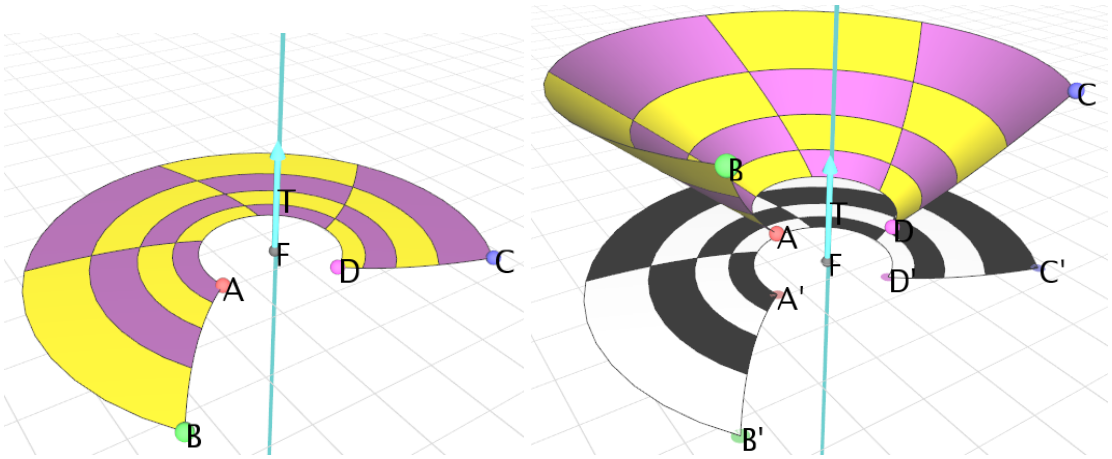


Figure 4.3: (a) A COTS map in a plane π with normal $\vec{T}$ is produced when π contains both A and F. (b) Raising A outside of π produces a BeCOTS map. The grey shadow is the closest projection of the BeCOTS onto π and is equal to the original COTS.

enough to uniquely define an interpolating cone. Even when given six distinct points, there may exist up to 12 distinct cones that interpolate the points [8]. However, for the four corner points of a BeCOTS, we compute a particular, unique cone that interpolates the corners, as will be discussed in subsection 4.3.4.

The one-parameter **Raise-family** of BeCOTS maps may be useful for animation (see Figure 4.6) and for creating three-dimensional maps or structures, such as the BeCOTS Stack lattices described in section 6.3.

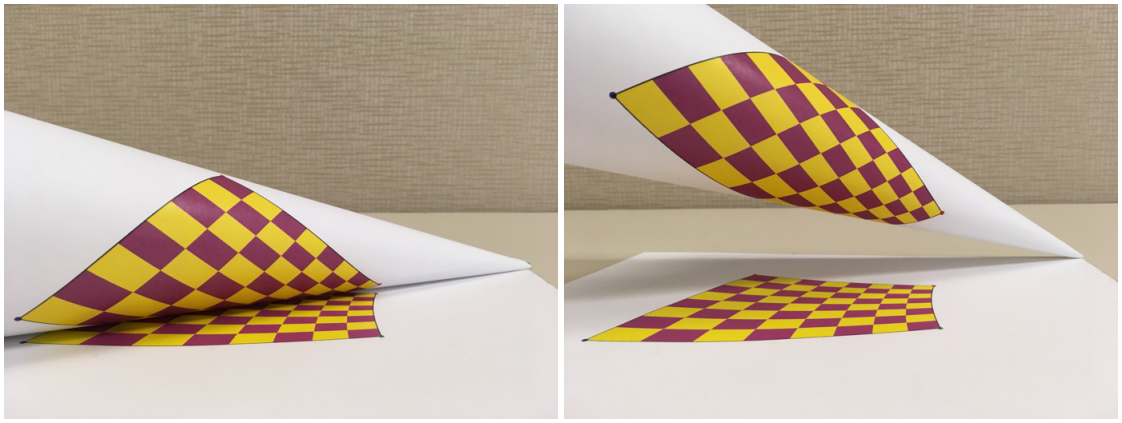
The **Project** operator, which we define as the inverse of Raise, is simply the closest projection onto the plane π through F with normal $\vec{T}$. Hence, Project maps a BeCOTS map to a COTS map (see Figure 4.3).

The geometric formulations of the Raise and Project operators are trivial. Additionally, the simple correspondence between a BeCOTS and its projected COTS, makes the Project operator particularly useful for analyzing BeCOTS maps and lattices.

Note that, in general, a bending of a planar COTS map onto the surface of a cone does not produce a Tran-Similar map.

4.3.2 Bending a COTS map

The Raise operator discussed above does not preserve local geodesic distances. For applications where preserving geodesic distances is important, we define the **Bend** of a COTS map. The intuitive idea is to consider that the COTS map is drawn on a sheet of paper that lies on the table and that its fixed point F is a corner of that sheet. Keeping F and A fixed on the table, we bend (roll) the sheet into a right-circular cone of apex F (see Figure 4.4).



(a) (b)

Figure 4.4: (a) BeCOTS formed by rolling a paper cone, and (b) a different positioning of the cone. To produce this contraption, the images on the flat and on the rolled sheets should be mirror images of each other to show the top and the bottom of the surfaces respectively.

Note that, because of the physical constraint, the cone remains tangent to the table. Hence, the bending preserves the surface normal along the contact line through F and A.

Also note that, instead of A, we could have picked an arbitrary point on the sheet. We picked A because doing so simplifies the formulation of the Bend operation.

However, we could not have picked a different point than F to be the apex of the cone. A dilation transforms a point P so that it is mapped onto the ray from F through P. Imagine a planar COTS with all such rays extending from F. After the bending operation, all of these rays must remain straight if they are to continue representing a dilation. For a bend that maps F onto a point other than the apex, these rays will also become bent around the cone, instead of remaining straight. Hence, in general, a bending of a planar COTS map

onto the surface of a cone does not produce a Tran-Similar map.

Observe that we can control the amount of bending by the angle β between the axis of the cone, which has direction $\vec{T}'$, and the vector $\overrightarrow{FA}$ (see Figure 4.5).

Given a point $P = \mathbf{M}(x, y)$ on the plane of the range R of the COTS map $\mathbf{M}$, the bending of P may be computed using the composition of commutative rotations and dilations, as discussed in subsection 4.3.1, however with modified parameters such that:

$$\begin{aligned} \vec{T}' &= \textbf{Rotation}(\pi/2 - \beta, F, \underline{\vec{T} \times \overrightarrow{FA}}) \circ \vec{T} \\ w'_u &= w_u / \textbf{sin}(\beta) \\ w'_v &= w_v / \textbf{sin}(\beta) \\ \textbf{Bend}(P) &= \textbf{Rotation}(xw'_u + yw'_v, F, \vec{T}') \circ \textbf{Dilation}(m_u^x m_v^y, F) \circ A \end{aligned} \tag{4.5}$$

In our notation, $|\overrightarrow{FA}|$ is a vector length and $\underline{\overrightarrow{FA}}$ is a vector normalization, $\overrightarrow{FA}/|\overrightarrow{FA}|$. $\vec{T}'$ represents the axis direction of the bent cone, which has an angle of β with $\overrightarrow{FA}$. w'_u and w'_v are the rotation angles around $\vec{T}'$ which are scaled versions of w_u and w_v. Observe that when $\beta = \pi/2$, the bending leaves COTS unchanged.

Figure 4.6 shows a planar COTS map and several of its Bent versions. Observe that tightening the Bend (i.e., reducing β) will eventually produce a bent version of R that not only self-overlaps but wraps several times around the cone.

Unbend, the inverse of bend, flattens a BeCOTS by unrolling it onto a plane. Given the parameters describing a BeCOTS map (A, F, m_u, m_v, w'_u, w'_v, and $\vec{T}'$), we compute the parameters (w_u, w_v, and $\vec{T}$) of its unbent COTS version as follows:

$$\begin{aligned} \beta &= \overrightarrow{FA} \hat{} \vec{T}', \text{ the angle from } \overrightarrow{FA} \text{ to } \vec{T}' \\ w_u &= w'_u \, \textbf{sin}(\beta) \\ w_v &= w'_v \, \textbf{sin}(\beta) \\ \vec{T} &= \underline{\vec{H}}, \text{ where } \vec{H} = (\overrightarrow{FA} \times \vec{T}') \times \overrightarrow{FA} \end{aligned} \tag{4.6}$$

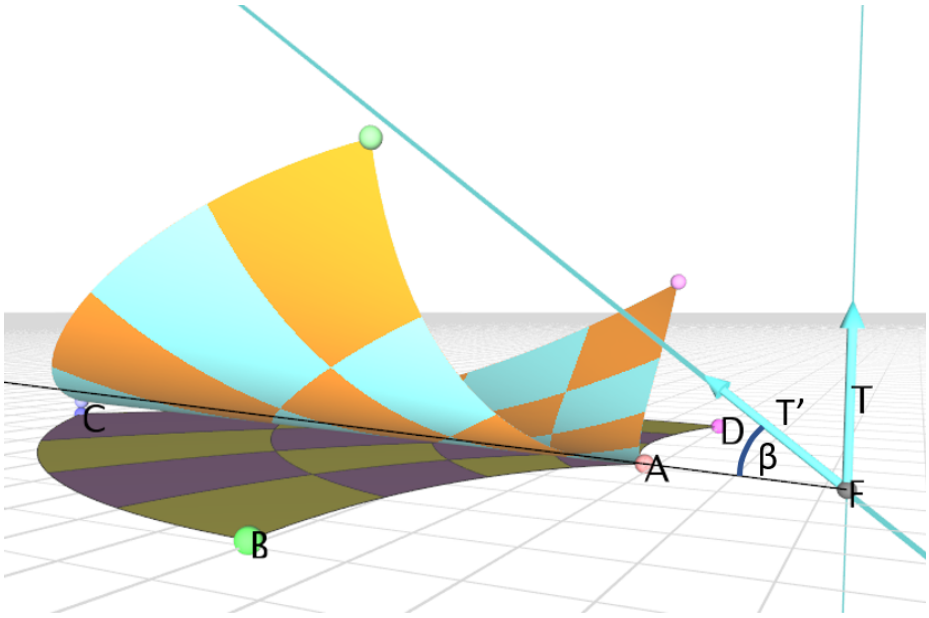


Figure 4.5: Using the parameter β, a user can control the bend of the planar COTS onto the cone with apex F, axis direction $\vec{T}'$, and apex angle β.

Now, given a point Q on a BeCOTS map, we want to compute its unbent version Q' on the corresponding planar COTS. The bend operation may map multiple points to a single one, when the map self-overlaps from wrapping around the cone. So, for self-overlapping BeCOTS, the Unbend of Q may return multiple valid points Q' (see Figure 4.10).

The key observation for computing Q' is that the domain space coordinates (x, y) of both Q and Q' are the same. So, we first compute the (possibly multiple) domain space coordinates (x, y) of Q with respect to the BeCOTS map **M'**, using the inverse BeCOTS map $\mathbf{M'}^{-1}(Q) = (x, y)$, which is discussed later in subsection 4.3.5. Then, we map the (x, y) coordinates into the flattened map with the following expression:

$$\begin{aligned} \textbf{Unbend}(Q) = \textbf{Rotation}(xw_u + yw_v, F, \vec{T}) \circ \textbf{Dilation}(m_u^x m_v^y, F) \circ A \\ \text{where } (x, y) = \mathbf{M'}^{-1}(Q) \end{aligned} \tag{4.7}$$

Sheet-bending is discussed in [41], including configurations controlled by four corners. We conjecture that the BeCOTS bending proposed here may correspond to the physical

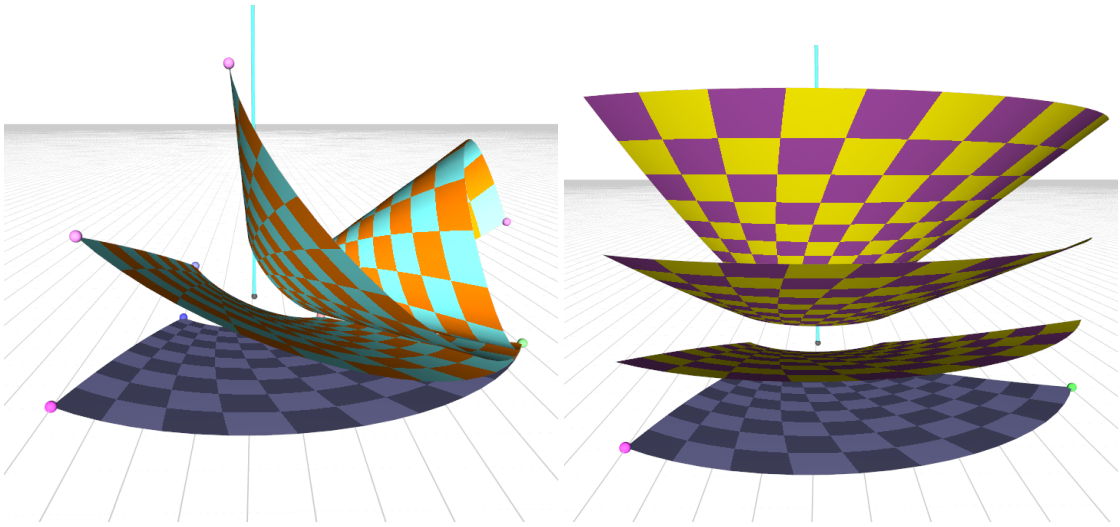


Figure 4.6: Families of BeCOTS maps produced by bending (left) and by raising (right).

bending of homogeneous sheets without gravity or constraints. Hence, the cone construction from four points proposed here may be useful in the following pipeline (1) The user specifies four non-coplanar points which define a cone; (2) He/she obtains the preimages of these points on the plane; (3) He/she can translate, rotate, and scale these preimages with respect to a planar shape that needs to be bent; (4) We provide a closed-form conformal map that preserves geodesic distances that takes the shape onto the cone. This map is the composition of the BeCOTS map with the inverse of the COTS map defined by the preimages of these four points.

4.3.3 Control of the warp, bend, and placement

Here we discuss the **Degrees of Freedom** (**DoF**) of a BeCOTS map by considering a pipeline of operations that defines a BeCOTS field and map: (1) the planar **warp**, which is independent of scale, position, and orientation and controls the distortion of the tiles of the corresponding COTS map; (2) the **bend**, which is the amount of bending applied to transform that COTS map onto a BeCOTS; and (3) the **placement**, which is a similarity that defines the final position, size, and orientation of the BeCOTS field. When the BeCOTS field is used to define a BeCOTS map, which is CO and hence has 12 DoF, the placement

has 7 DoF, the bend has 1 DoF (tip angle β of the cone), and the warp has 4 DoF. In Figure 4.7 we illustrate the effect of the warp and bend operations.

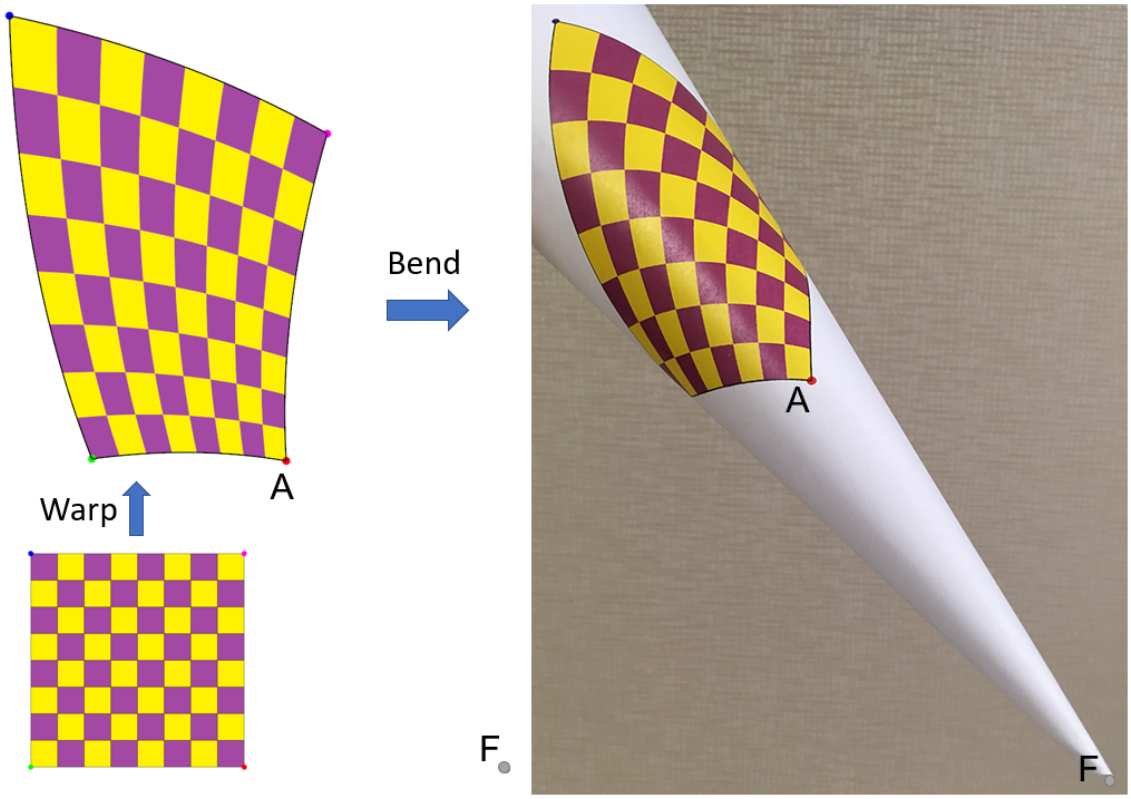


Figure 4.7: The domain D and its warp (left). The result of its bending (right). Notice that the fixed point F is the apex of the cone.

4.3.4 BeCOTS field from four points

Here we explain how the similarities **U** and **V** of a BeCOTS field may be defined using non-coplanar points $\{A, B, C, D\}$ and constraints: $\mathbf{M}(0,0) = A$, $\mathbf{M}(0,1) = B$, $\mathbf{M}(1,1) = C$, and $\mathbf{M}(1,0) = D$.

We define $\mathbf{U} = \mathbf{SIM}(A, B, D, C)$ and $\mathbf{V} = \mathbf{SIM}(A, D, B, C)$, where $\mathbf{SIM}(A, B, C, D)$ is a similarity that maps A to C and B to D. There is not a unique similarity that maps a pair of 3D points to another pair of 3D points (see subsection 2.5.3), so the function **SIM** is not the same as the function **Similarity** that defines the 2D similarity that takes a 2D edge to another 2D edge. However, remember that **SIM** has the additional constraint that **U** and

V be commutative, which makes it well defined.

We propose to compute the parameters $<F, \vec{T}, m, w>$ of **SIM**(A, B, C, D) as follows:

$$\begin{aligned} m &= |\overrightarrow{CD}|/|\overrightarrow{AB}| \\ \vec{T} &= \underline{\vec{N}}, \text{where } \vec{N} = (\underline{\overrightarrow{AB}} - \underline{\overrightarrow{CD}}) \times (\underline{\overrightarrow{AC}} - \underline{\overrightarrow{BD}}) \end{aligned} \tag{4.8}$$

Let B', C', and D' be the closest projections of B, C, and D onto the plane through A with normal $\vec{T}$. Then,

$$\begin{aligned} w &= \overrightarrow{AB'} \hat{} \overrightarrow{C'D'}, \text{ where } \vec{U} \hat{} \vec{V} \text{ denotes the angle from vector } \vec{U} \text{ to } \vec{V} \\ G &= A + \overrightarrow{C'D'} \\ F' &= \textbf{Dilation}_2(\frac{|\overrightarrow{B'D'}|}{|\overrightarrow{B'G}|}, B') \circ \textbf{Rotation}_2(\overrightarrow{B'G} \hat{} \overrightarrow{B'D'}, B') \circ A \\ F &= F' - h\vec{T}, \text{where } h = \overrightarrow{AC} \cdot \vec{T}/(m-1) \end{aligned} \tag{4.9}$$

The construction of **SIM**(A, B, C, D) requires that no two input points coincide. Additionally, $|\overrightarrow{AB}|$ must not equal $|\overrightarrow{CD}|$, or else F will be undefined and **SIM** will not have a dilation component. And, $\overrightarrow{AB}$ must not be parallel with $\overrightarrow{CD}$, or else $\vec{T}$ will be undefined and **SIM** will either not have a rotation component or the rotation will be by $360°$ around an ambiguous axis. These situations may be detected and handled as special cases.

Now, we justify the above construction of the parameters.

We start by justifying the construction of $\vec{T}$. Consider applying $\mathbf{S} = \textbf{SIM}(A, B, C, D)$ to the vector $\overrightarrow{AB}$. The resulting vector will be $\overrightarrow{CD} = \mathbf{S} \circ \overrightarrow{AB}$, which is $\overrightarrow{AB}$ transformed by a composition of a dilation and a rotation around direction $\vec{T}$. If we consider the normalized versions of these vectors (see Figure 4.8), then $\underline{\overrightarrow{CD}}$ (green) is simply $\underline{\overrightarrow{AB}}$ (red) rotated around $\vec{T}$ (cyan). The tip of $\underline{\overrightarrow{AB}}$ rotates in a plane perpendicular to $\vec{T}$, so $\underline{\overrightarrow{AB}} - \underline{\overrightarrow{CD}}$ (yellow) is a vector perpendicular to $\vec{T}$. Similarly, $\underline{\overrightarrow{AC}} - \underline{\overrightarrow{BD}}$ (grey) is a vector perpendicular to $\vec{T}$, so the direction of $\vec{T}$ may be computed as the cross product of the two differences. This

construction is inspired by the construction of the axis of a screw motion in [37].

Now, we justify the construction of F'. The projection of the BeCOTS into the plane, through A with normal $\vec{T}$, is a COTS with corners A, B', C', and D' and with scaling factor m and rotation angle w. The construction of the parameters for a planar spiral transformation were described in subsection 2.4.6, but we also include the justification here for convenience. Consider the diagram in Figure 4.9a, which shows these corners, the planar fixed point F', and the point $G = A + \overrightarrow{C'D'}$. Triangle $B'AG$ is similar to $B'F'D'$, so F' is constructed by rotating and dilating $B'AG$ around B' so that **edge**(B', G) becomes **edge**(B', D'). F' is the image of A after this transformation.

Finally, we justify the construction of F. F is on the line through F' with direction T, so we need to compute h, such that $F' = F + h\vec{T}$. F' is the closest projection of A on this line. Let Y be the closest projection of C on the line. $\overrightarrow{AC} \bullet \vec{T}$ is the distance from F' to Y. Consider the relation $h + \overrightarrow{AC} \bullet \vec{T} = mh$, which we solve for h. See Figure 4.9b.

4.3.5 Inversion

Given a point Q that lies on the range R of a BeCOTS map $\mathbf{M}(x, y)$, we compute the (x, y) parameters for which $\mathbf{M}(x, y) = Q$ by solving the following system of linear equations:

$$\begin{aligned} x \ln m_u + y \ln m_v &= \ln m \\ x w_u + y w_v &= w \end{aligned} \tag{4.10}$$

where $m = \frac{|\overrightarrow{FQ'}|}{|\overrightarrow{FA'}|} = \frac{|\overrightarrow{FQ}|}{|\overrightarrow{FA}|}$, $w = \overrightarrow{FA'} \hat{} \overrightarrow{FQ'} + 2\pi k$, A' and Q' are the closest projections of A and Q onto the plane π through F with normal $\vec{T}$, and integer k defines the **branching option** of the number of full rotations in angle w.

This is the same formula as the one given in [35]. This reuse is justified because an application of the Raise operator (and therefore also Project) (subsection 4.3.1) does not change any parameters of a planar COTS map $\mathbf{M}'$ except for the heights of the corners of R. So, the parameters and formula for computing the inverse of Q with respect to $\mathbf{M}$ are

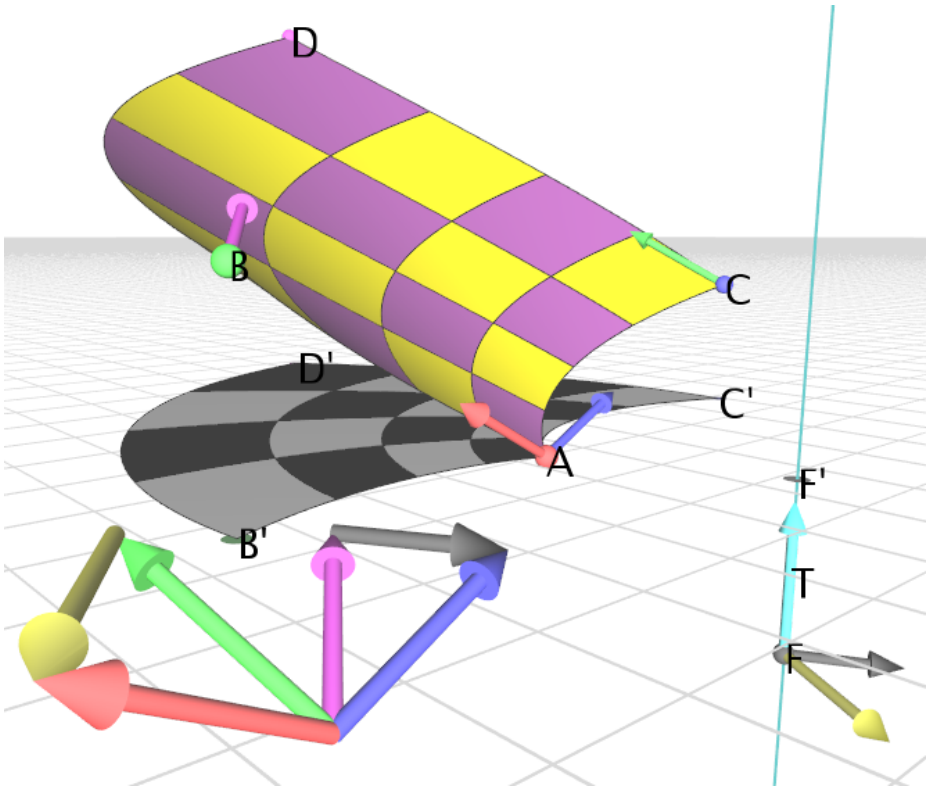
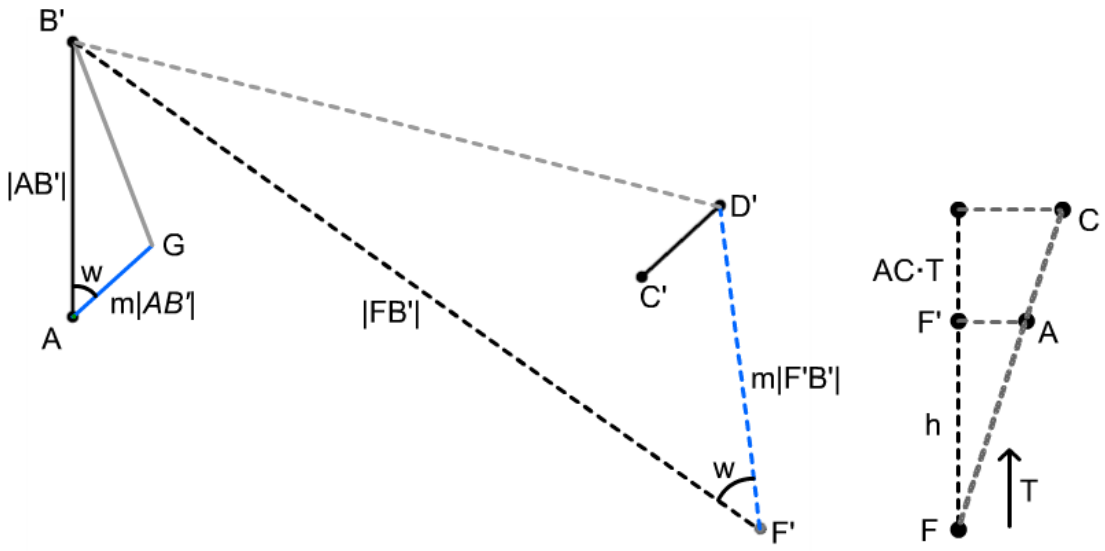


Figure 4.8: Shown here are several arrows that represent important vectors. Red=$\overrightarrow{AB}$, Green=$\overrightarrow{CD}$, Blue=$\overrightarrow{AC}$, Magenta=$\overrightarrow{BD}$, Yellow=$\overrightarrow{AB} - \overrightarrow{CD}$, Grey=$\overrightarrow{AC} - \overrightarrow{BD}$, and Cyan=$\vec{T}$. Note, the corner ordering $(ABDC)$ here is used for computing the parameters of **SIM**(A, B, D, C), and this ordering differs from the ordering $(ABCD)$ around a BeCOTS map.

the same as for computing the inverse of Q' with respect to $\mathbf{M}'$. The corner A is used in Equation 4.10 despite its height changing, but this is not an issue, because under Raise and Project, the ratios of distances from F are preserved and the angles around the axis through F with direction $\vec{T}$ are preserved.

The linear system (Equation 4.10) has an infinite number of solutions, corresponding to each value of branching option k (see Figure 4.10). We are only concerned with maps of the unit square and therefore only concerned with values of k that correspond to x and y values that are both in $[0, 1]$.

So, to compute all valid solutions (there may be more than one if R self-overlaps), we



(a) (b)

Figure 4.9: The triangles $B'AG$ and $B'F'D'$ are similar, which allows F' to be constructed by rotating and dilating point A around B' such that the same transformation would align $B'AG$ to $B'F'D'$. This diagram is a modified version of Figure 2.4, included here for convenience (a). A diagram that, for simplicity, assumes **SIM**(A, B, C, D) is a pure dilation, which demonstrates how F may be computed (b).

compute the range

$$\begin{aligned} \mathrm{W} &= [w_{min}, w_{max}] \\ &\text{where } w_{min} = \mathbf{min}(0, w_u, w_v, w_u + w_v) \\ &\text{and } w_{max} = \mathbf{max}(0, w_u, w_v, w_u + w_v) \end{aligned} \tag{4.11}$$

Then, for each k value for which $w \in \mathrm{W}$, we compute:

$$\begin{aligned} x &= (w_v \ln m - w \ln m_v)/(w_v \ln m_u - w_u \ln m_v) \\ y &= (w - x w_u)/w_v \end{aligned} \tag{4.12}$$

If both x and y are in $[0, 1]$ then the inverse (x, y) is valid.

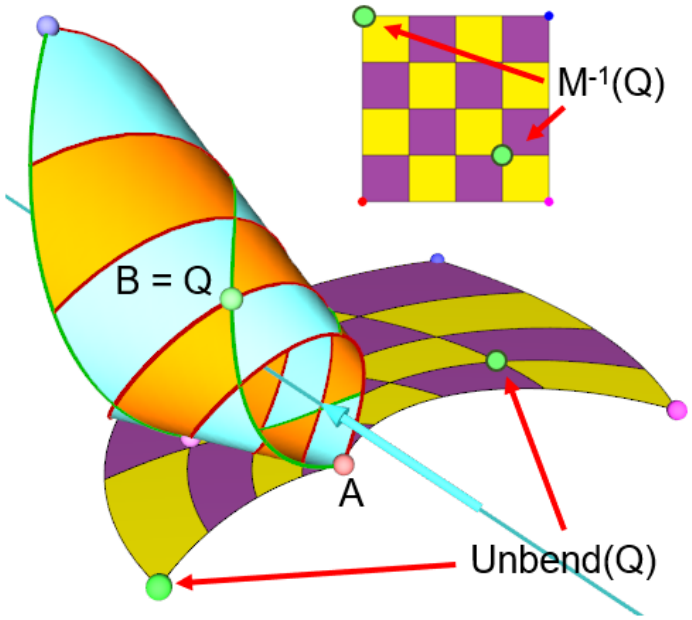


Figure 4.10: A self-overlapping BeCOTS map. Consider the point Q, which is equal to corner B. **Unbend**(Q) maps to two points, shown in green on the flattened COTS. Similarly, $\mathbf{M}^{-1}(Q)$ maps to two points, also shown in green on the image of domain D at the top.

4.4 BeCOTS Patterns

Here we discuss the design of patterns and tilings with BeCOTS, and we discuss the computation of Point Membership Classification (PMC) and Integral Property Computations (IPCs) on patterns designed with BeCOTS.

4.4.1 BeCOTS Pattern and Tiling

Given a template-shape $\mathrm{X}_{0,0}$, a BeCOTS field, $\mathbf{S}(x, y)$, and two repetition counts, $\bar{i}$ and $\bar{j}$, we define the **BeCOTS Pattern** as the set of instances $\{\mathrm{X}_{i,j} = \mathbf{S}(i/\bar{i}, j/\bar{j}) \circ \mathrm{X}_{0,0}\}$, where $0 \leq i < \bar{i}$ and $0 \leq j < \bar{j}$.

Let $\mathrm{T}_{0,0}$ be the image of the rectangle with corners at $(0, 0)$, $(1/\bar{i}, 0)$, $(1/\bar{i}, 1/\bar{j})$, and $(0, 1/\bar{j})$. We call $\mathrm{T}_{0,0}$ the **template-tile**. The BeCOTS pattern of $\mathrm{T}_{0,0}$ is a tiling of the range R.

Given a query point Q on the cone containing R, the (x, y) parameter pair(s) corre-

sponding to Q may be computed as discussed in subsection 4.3.5. The tile containing the pair (x, y) is $\mathrm{T}_{\lfloor x\bar{i} \rfloor, \lfloor y\bar{j} \rfloor}$, where $\lfloor x\bar{i} \rfloor$ denotes the floor of $x\bar{i}$.

4.4.2 Tiles of BeCOTS maps are similar to each other

The tiles of a BeCOTS map (for given $\bar{i}$ and $\bar{j}$ repetition counts) are all similar to each other. This is also true for 2×2 and other blocks of tiles. However, a single tile is not similar to a block of tiles nor to the entire range R. Such tile-similarity was proven for the planar tiles of a COTS map. Tile-similarity in BeCOTS maps is a direct consequence of its field's Tran-Similarity. The cells of the domain D are mapped into the tiles of the range R, and since each pair of cells is related by a translation, each pair of tiles is related by a similarity.

Figure 4.11 demonstrates tile similarity by overlaying two similar copies of a BeCOTS map. Figure 4.1 also demonstrates tile similarity by showing two views of different tiles.

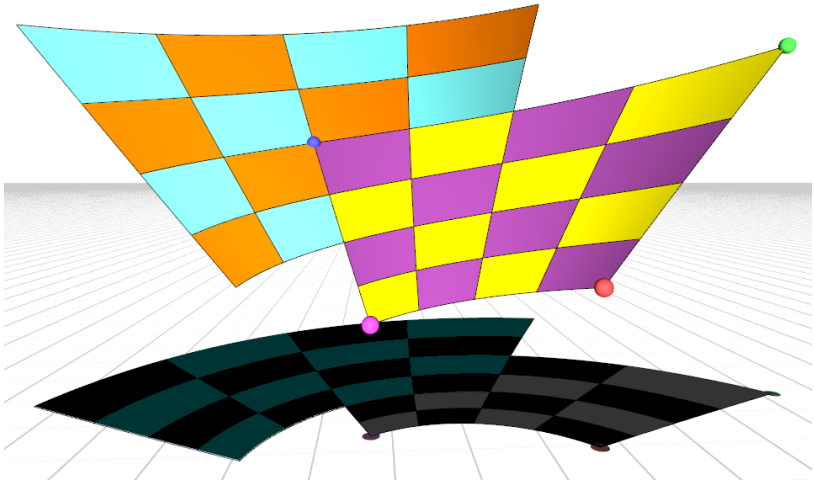


Figure 4.11: A tiled BeCOTS map partly overlaid on a similar version of itself so that the two tilings match seamlessly. The second map is an offset of the first by vector $<1/2, 1/2>$ in the domain D. This corresponds to a transformation of the first map by similarity $\mathbf{U}^{1/2} \circ \mathbf{V}^{1/2}$.

4.4.3 Seamless self-overlap and annulus range

By extending the domain, by reducing the apex (opening) angle β of the cone when bending, or by a non-zero branching, one can create BeCOTS maps that self-overlap. For a given tiling (i.e. repetition counts), the BeCOTS warp can be adjusted so as to align the tile vertices and the border edges perfectly and to produce a seamless tiling that has two border loops (Figure 4.12).

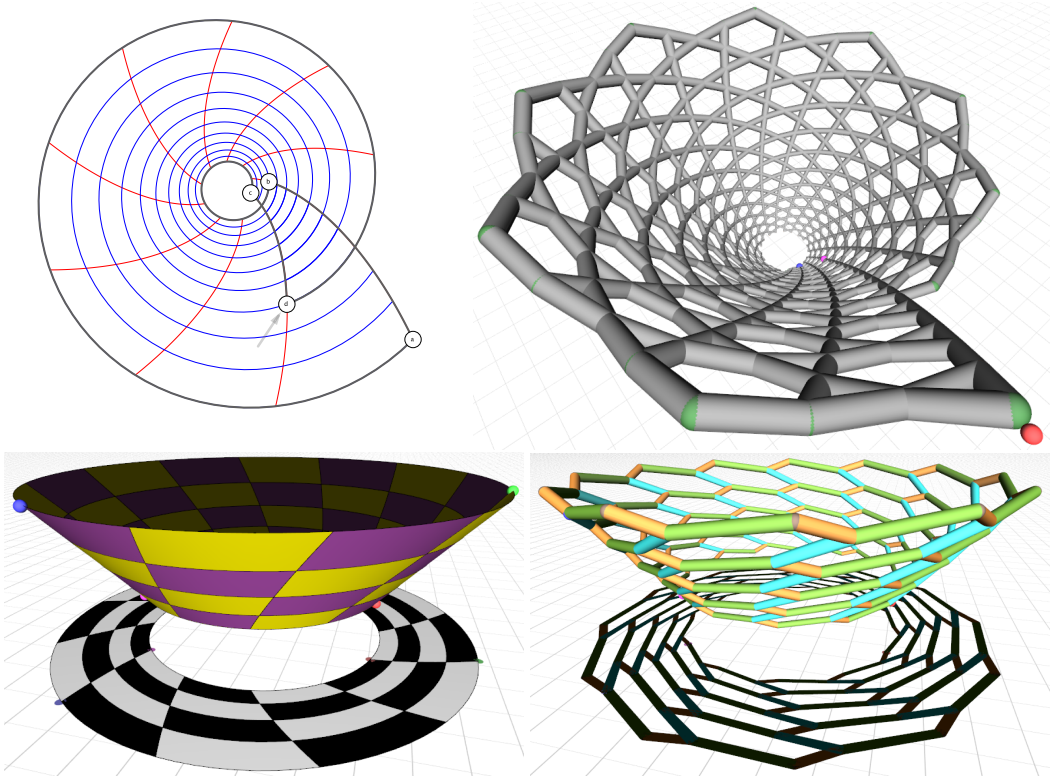


Figure 4.12: A seamless COTS tiling that was created by aligning corner D with grid-vertex $(1, 3)$ (top-left). A BeCOTS Kagome was lattice created by raising it (top-right). A seamless BeCOTS tiling that forms an annulus with two border loops (bottom-left). A BeCOTS honeycomb lattice along its field (bottom-right).

4.4.4 Point-Membership Classification

In chapter 8, we discuss how to perform Ball Interference Queries (BIQs) against BeCOTS patterns and lattices in constant-time. A BIQ takes a query ball Q as input and returns the shapes of a BeCOTS pattern that interfere (have a non-empty intersection) with Q. Point

Membership Classification (PMC) is a special case of BIQ where the query ball Q has zero radius.

4.4.5 Integral Property Computation

We are interested in the efficient computation of various integral properties for BeCOTS patterns. Here, we focus on total volume computation and total surface area computation, both in constant-time. Though, we conjecture that other integral properties, the centroid for example [14], may be computed in a similar manner.

Total Surface Area Computation

Given a BeCOTS pattern with disjoint shapes $\{\mathrm{X}_{i,j}\}$. A naive, $\mathbf{O}(\bar{i}\,\bar{j})$ complexity, approach would be to compute and sum up the surface area of each shape of X. Gupta et al. present an $\mathbf{O}(\bar{i})$ approach to total surface area computation [14] of steady patterns[1]. Here, we present an $\mathbf{O}(1)$ approach.

First, compute the surface area $a_{0,0}$ of the template-shape $\mathrm{X}_{0,0}$.

Then, let $d_u = m_u^{2/(\bar{i}-1)}$ be the surface area ratio between consecutive shapes such that $d_u = a_{i+1,0}/a_{i,0}$. Similarly compute $d_v = m_v^{2/(\bar{j}-1)}$.

The surface area a of the pattern may then be written as a double sum of geometric progressions, which simplifies to a constant time computation.

$$a = \sum_{j=0}^{\bar{j}-1} d_v^j \Big(\sum_{i=0}^{\bar{i}-1} d_u^i * a_{0,0}\Big) = a_{0,0} * \frac{d_u^{\bar{i}} - 1}{d_u - 1} * \frac{d_v^{\bar{j}} - 1}{d_v - 1} \tag{4.13}$$

This closed-form expression may be more accurate than the double sum, because it avoids accumulating error over many repeated operations.

[1] They also present $\mathbf{O}(\bar{i}\,\bar{j})$ algorithms for $\bar{i} \times \bar{j} \times \bar{k}$ steady bricks.

Total Volume Computation

Equation 4.13 can be trivially modified to support the computation of the total volume of a pattern with disjoint shapes. Start by letting the symbol a represent the total volume and $a_{0,0}$ represent the volume of the template-shape. Then, let $d_u = m_u^{3/(\bar{i}-1)}$ be the volume ratio between consecutive shapes such that $d_u = a_{i+1,0}/a_{i,0}$. Similarly compute $d_v = m_v^{3/(\bar{j}-1)}$. These substitutions form the total volume computation, which has the same justification as for the area computation.

4.5 BeCOTS Lattices

The BeCOTS lattices proposed here may be used to address several design and performance issues that limit the usefulness and scalability of previously proposed steady (chapter 3) and non-steady lattices.

Here, we discuss applications of BeCOTS to the design of an interesting subclass of Steady Lattices.

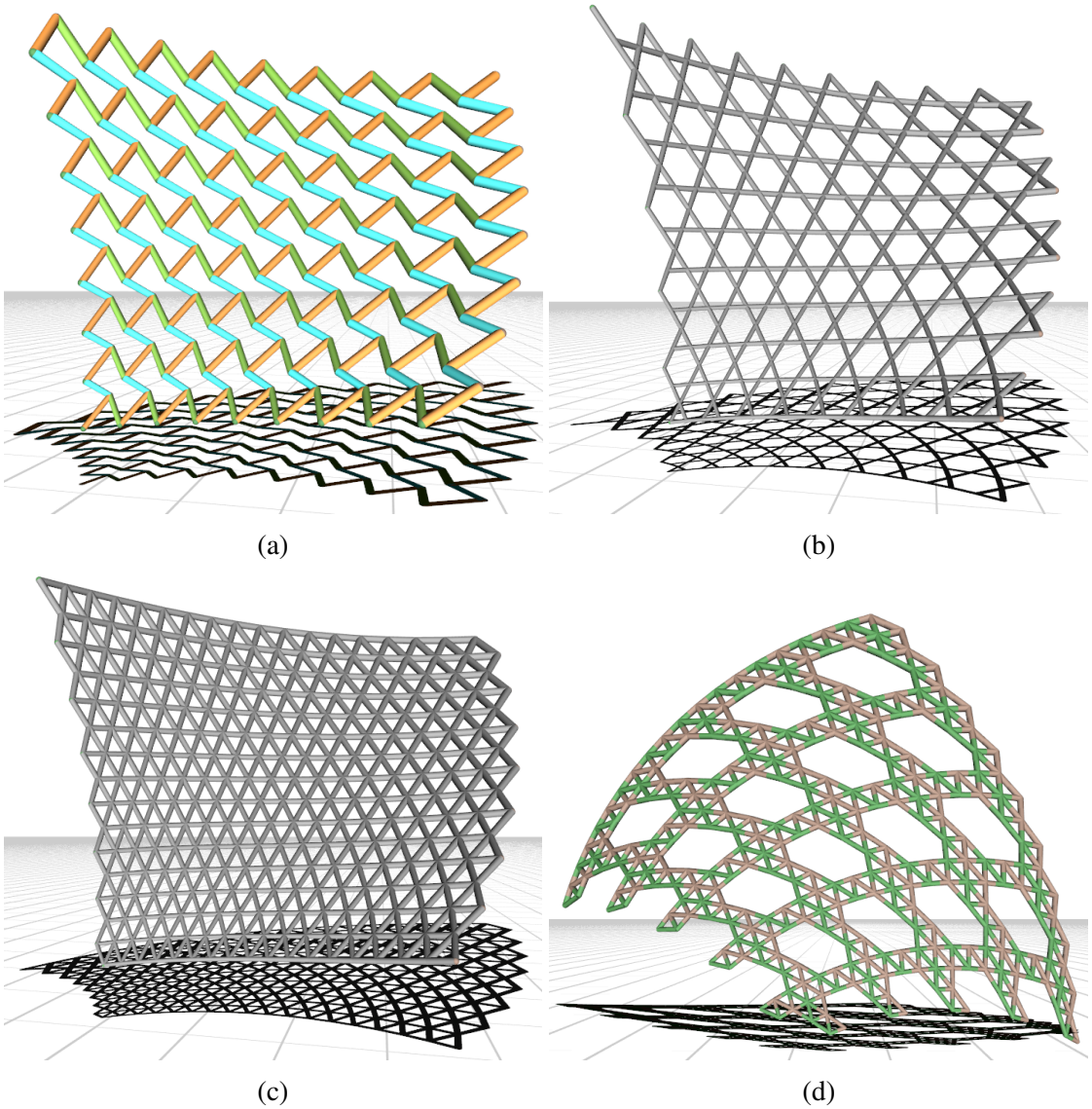
A BeCOTS lattice is a union of BeCOTS patterns where each pattern is a set of shapes $\mathrm{X}_{i,j}$ that are each a hub. We assume the lattice is clean, meaning that all hubs have pairwise disjoint interiors.

Because a BeCOTS lattice is the union of one or more BeCOTS patterns of hubs, the constant time PMC and IPC algorithms discussed in section 4.4 apply here.

4.5.1 Control over the lattice connectivity

The BeCOTS formulation supports a variety of lattice connectivities (Figure 4.13), including a filtered (multi-level) structure (discussed in chapter 9). These can be easily programmed. When combined with the flexibility of selecting the angle β of the supporting cone and the orientation and spacing of swirling beam-patterns, BeCOTS lattices may offer a useful tool for designing and optimizing the support structures or cooling systems for

conical shields in cockpits of airplanes or for nose-cones of rockets.



(a) (b)

(c) (d)

Figure 4.13: BeCOTS lattice with 2 node patterns and 3 beam patterns for which the internal hubs have 3 incident beams each (a) and Kagome lattice for which the internal hubs have 4 incident beams each (b). A triangle mesh for which the internal hubs have 6 incident beams each (c). A programmed multi-level structure where a periodic subset of the triangle mesh beams have been removed algorithmically (d).

4.5.2 Possibility of 3-directional BeCOTS?

We wish to extend the BeCOTS maps, patterns, and lattices with a third direction while maintaining the useful properties of BeCOTS. Unfortunately, such an extension cannot

preserve all of the useful properties of BeCOTS.

For example, consider that a Tran-Similar map (except for the special case of a regular map) must lie on the surface of a cone (or degenerate forms of a cone: a cylinder or a plane), as justified in subsection 4.3.1. A 3-directional map that lies on a cone must have a redundant direction. Therefore, a 3-directional extension of BeCOTS cannot retain Tran-Similarity nor the properties that require Tran-Similarity. Note that a regular, 3-directional map is Tran-Similar, and regular 3-patterns allow constant-time IPC and PMC queries. However, regular 3-maps are neither corner-operated nor bent.

In section 6.3, we describe a possible 3-directional extension of BeCOTS, called a BeCOTS stack, that preserves some useful properties. In particular, the BeCOTS stacks permit constant-time PMC queries, and certain 2-pattern subsets of a BeCOTS stack pattern are Tran-Similar.

4.5.3 Novel application: Constant-radius BeCOTS lattices

A variety of architectural designs include curved lattices produced by assembling bars (beams) using connectors (nodes) [7, 27]. The beams in such structures are not necessarily cones and the nodes are not necessarily balls. Because each connector of a generic lattice is not generally congruent with any other, each connector must be custom made. This considerably increases the manufacturing cost which has encouraged work to design lattices with all or many of its connectors being congruent [7, 27]. We propose to use, when possible, BeCOTS lattices to reduce manufacturing cost.

Our solution is based on the following idea. We start with a BeCOTS lattice. We make the radii of all nodes identical. For each hub, we define the corresponding connector by taking the intersection of the grown version [38] of that hub with an enlarged version of its node and subtracting the original hub. Observe that, because of Tran-Similarity, the connectors corresponding to the same node-pattern are congruent (Figure 4.14), and hence may be mass produced, for example by injection moulding. Also, the beams can all be cut

from equal-radius rods.

When designing a constant-radius BeCOTS lattice, since all connectors are identical, it is important to ensure that each beam is long enough to separate its two connectors.

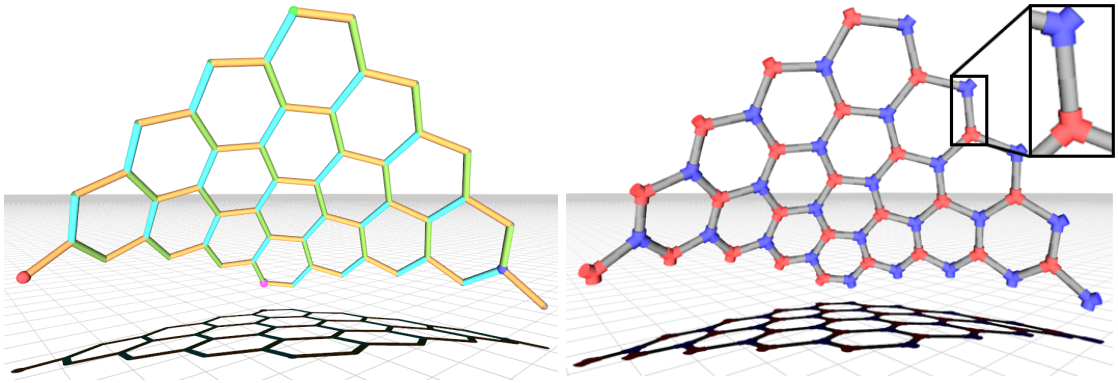


Figure 4.14: BeCOTS lattice (left) and its constant radius version with red and blue connectors (right). All blue connectors are identical and hence can be mass produced. All red connectors are also identical.

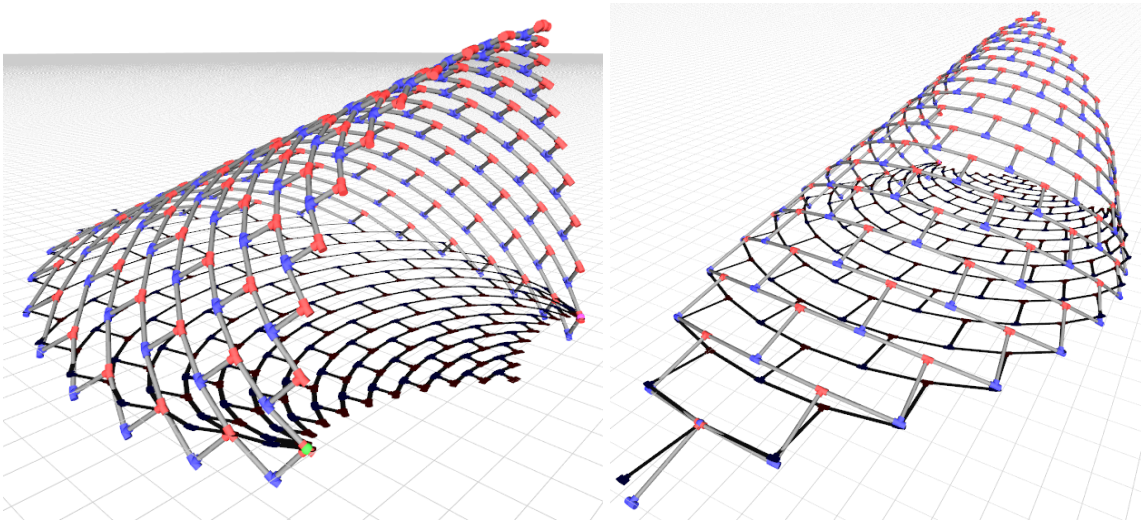


Figure 4.15: Two views of a constant-radius BeCOTS architectural structure and its shadow.

CHAPTER 5

CONSTRUCTIVE LATTICE GEOMETRY (CLG) FOR BECOTS LATTICES

5.1 Motivation and introduction

So far, we have focused on lattices defined as a union of balls and of conical frustums. Here, we generalize the concept of a lattice such that each beam may be a more complex solid that is not a union of balls and of conical frustums.

CSG allows for modeling complex structures as a Boolean combination of many different primitive solids. However, for a complex lattice, manually placing each one of these possibly millions of primitives and specifying the Boolean combinations would be prohibitively time consuming. So, we are combining the power of CSG with the power of periodic lattices in order automate the creation of complexity through programmed repetition.

We propose **Constructive Lattice Geometry (CLG)** as a system for designing complex, periodic CSG structures. CLG may be considered both a generalization of CSG to support periodic structures, and CLG may be considered a generalization of periodic lattices to support CSG-based structures.

CLG must simultaneously satisfy two needs. One is the ability to specify local-scale Boolean combinations to construct primitive level geometries that are not simple balls and cones. Second is the ability to specify global-scale Boolean combinations to union, intersect, and subtract entire lattices.

We have chosen to present CLG as an extension of BeCOTS lattices, rather than as an extension of general steady lattices or of general periodic lattices. We use the Trans-Similarity of BeCOTS to simplify the formulation of CLG. Trans-Similarity also simplifies the design of CLG structures by ensuring the structure is composed of pairwise similar

local-scale geometric objects, so a designer can be guaranteed that a designed CLG structure has no surprises localized in a small piece of the structure. Lastly, Trans-Similarity enables a constant-time BIQ (per Trans-Similar pattern), which we use to accelerate both PMC for voxelization and ray intersection queries for rendering of CLG structures.

Our intentional limiting of CLG as an extension of BeCOTS lattices restricts us to the modeling of slab structures. A generalization of our formulation for CLG can be implemented to support brick structures, and we have in fact implemented a version of CLG that is an extension of general steady brick lattices, which may be useful. This steady brick version of CLG, however, cannot utilize the many benefits of Trans-Similarity.

This chapter is organized as follows. In section 5.2, we present preliminary definitions that we use to define CLG in section 5.3. In section 5.4, we discuss compound primitives, which are user specified solid primitives defined as Boolean combinations of the provided ball and beam primitives. In section 5.5, we discuss PMC queries on CLG structures which we use in section 5.6 for the voxelization of CLG. Finally, in section 5.7, we discuss how to compute the first intersection of a ray with a CLG, which may be used to render a CLG structure.

5.2 Preliminary definitions

A **node-group pattern** N is a BeCOTS pattern of $\overline{u} \times \overline{v}$ node-groups defined as $\mathrm{N}[i,j] = \mathbf{V}^j \circ \mathbf{U}^i \circ \mathrm{N}[0,0]$, for all $i \in [0, \overline{u})$ and $j \in [0, \overline{v})$, where $\mathrm{N}[0,0]$ is the **template node-group** and **U** and **V** are commutative similarities.

A **primitive** is either a ball or a cone-beam defined in terms of one or two nodes. Let **Ball**(i,j,n,s) be a ball with the same center as the node $\mathrm{N}[i,j,n]$ and with a radius of s times the radius of $\mathrm{N}[i,j,n]$. Also, let **Beam**$(i_1,j_1,n_1,s_1,i_2,j_2,n_2,s_2)$ be the cone-beam that smoothly connects the balls **Ball**(i_1,j_1,n_1,s_1) and **Ball**(i_2,j_2,n_2,s_2).

A **primitive-group pattern** B is a BeCOTS pattern of primitive-groups, where each primitive in a group is defined in terms of nodes from N. Each primitive-group $\mathrm{B}[i,j]$,

for $i \in [0, \overline{u})$ and $j \in [0, \overline{v})$, is related to the **template primitive-group** $\mathrm{B}[0,0]$ such that $\mathrm{B}[i,j] = \mathbf{V}^j \circ \mathbf{U}^i \circ \mathrm{B}[0,0]$. Each primitive-group contains $\overline{b}$ primitives, and a primitive in group $\mathrm{B}[i,j]$ with ID b is referred to as $\mathrm{B}[i,j,b]$. Figure 5.1-Left shows an example primitive-group pattern and its underlying node-group pattern.

For example, a template beam primitive $\mathrm{B}[0,0,b] = \textbf{Beam}(-1,0,n_1,s_1,1,0,n_2,s_2)$ is defined between nodes $\mathrm{N}[-1,0,n_1]$ and $\mathrm{N}[1,0,n_2]$. If a copy $\mathrm{B}[i,j,b]$ of this beam is instantiated for the group $\mathrm{B}[i,j]$, it will connect the nodes $\mathrm{N}[i-1,j,n_1]$ and $\mathrm{N}[i+1,j,n_2]$, assuming both nodes exist. For simplicity and to avoid the tedium of handling special cases, we assume both nodes always exist for $i \in [0, \overline{u})$ and $j \in [0, \overline{v})$.

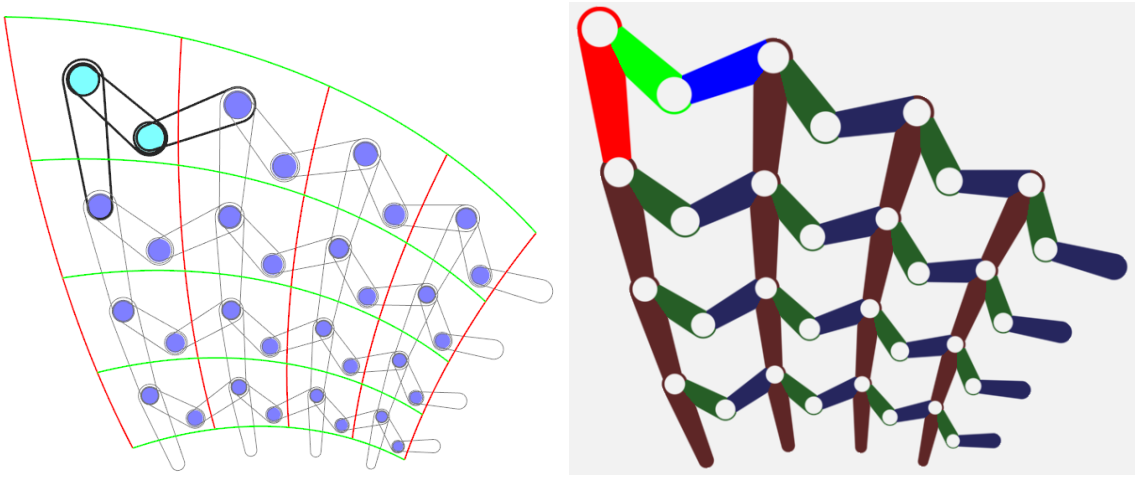


Figure 5.1: (Left) A planar COTS node-group pattern is shown in blue with the template node-group shown in cyan. Outlines of the primitives are shown, and the template primitive-group is shown in bold. (Right) A CLG S constructed from the nodes and primitives on the left. S is a subtraction of two chunk unions, where a union of disks is subtracted from a union of beams.

5.3 Definition

A **CLG** is a CSG solid S defined as a Boolean combination **E** of primitives from a primitive-group pattern B. Here we describe how **E** is constructed from B.

A **chunk pattern** C is a BeCOTS pattern of solids $\mathrm{C}[i,j]$, for $i \in [0, \overline{u})$ and $j \in [0, \overline{v})$, called **chunks**. Each chunk is related to a **template chunk** $\mathrm{C}[0,0]$ such that $\mathrm{C}[i,j] =$

$\mathbf{V}^j \circ \mathbf{U}^i \circ \mathrm{C}[0,0]$. The template chunk $\mathrm{C}[0,0]$ is a solid defined as a Boolean combination of primitives in primitive-group $\mathrm{B}[0,0]$. A CLG has $\bar{c}$ chunk patterns, so we refer to a chunk pattern with ID c as C_c. A **chunk union** H_c is the union of all chunks in the chunk pattern C_c. Finally, the expression $\mathbf{E}$ that defines the CLG solid S is a Boolean combination of chunk unions. Figure 5.1 shows an example CLG defined as the subtraction of two chunk unions.

The definition of a chunk allows for the creation of small complex solids, from simple primitives, that are repeated throughout a chunk union. The definition of a CLG allows for the creation of lattice-scale features in the final solid S, where a lattice-sized solid (a chunk union) may be considered a "primitive" in the Boolean combination that defines S.

5.3.1 Example: Hollow-beam lattice (shelling)

Consider the problem of designing a lattice for which the beams are hollow. Such a hollow lattice may be referred to as **shelled** [2]. The cavity in the lattice is to be fully connected so that a fluid could flow through the entire lattice. This problem demonstrates the utility of defining a CLG as an arbitrary Boolean combination of chunk unions, rather than as a single chunk union.

A first, naive attempt at a solution might be to design a CLG with a single chunk union for which each chunk is a union of hollow beams, where a hollow beam is a beam minus a thinner version of itself. However, this approach has an issue at the junction of multiple beams. The cavity of a beam does not open into the cavities of incident beams, as illustrated in Figure 5.2-Left.

A slight modification of the first approach might be to design each chunk as a subtraction $X - Y$ of two sets of beams X and Y, where the beams of Y are thinner versions of the beams of X. This connects the cavities of incident beams within a single chunk, but the chunks are all still unioned together such that the cavities of adjacent chunks are not connected, as shown in Figure 5.2-Center.

One proper solution is to define the CLG as a subtraction $L_X - L_Y$ of two chunk unions that are each a lattice of solid beams, where L_Y is a thinner version of L_X. L_Y represents the cavity of the hollow lattice, so the cavity is fully connected if L_Y is fully connected. See Figure 5.2-Right

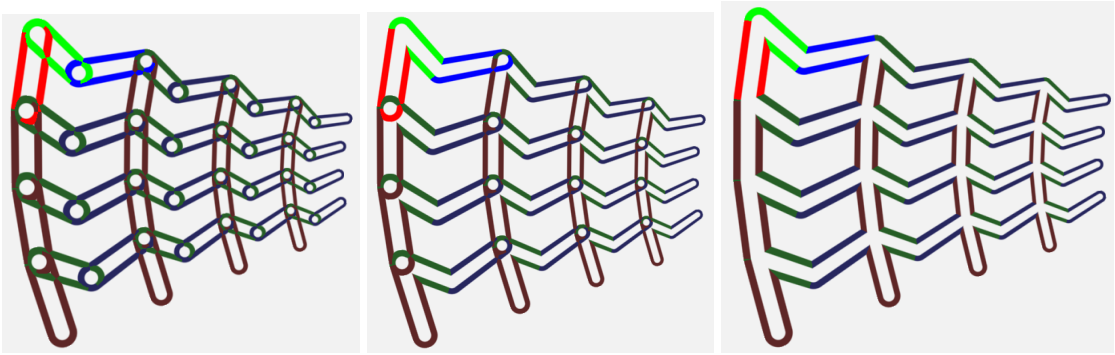


Figure 5.2: (Left) A incorrectly shelled lattice that is a union of hollowed beams. (Center) Another incorrectly shelled lattice that is a union of hollowed chunks of beams. (Right) A properly shelled lattice that is a subtraction of two chunk unions of beams.

5.4 Compound primitives

A **compound primitive** is a solid defined as a Boolean combination of primitives. Compound primitives are useful for simplifying design by offering a high-level abstraction over the simple primitives (balls and beams).

For example, in subsection 5.3.1, we referred to the definition of a chunk as a union of hollow beams, without providing a precise definition for hollow beams. Such a hollow beam may be defined with the following compound primitive expression,

$$\begin{aligned}&\text{HollowBeam}(i_1, j_1, n_1, s_1, i_2, j_2, n_2, s_2, h) = \\ &\qquad \text{Beam}(i_1, j_1, n_1, s_1, i_2, j_2, n_2, s_2) - \text{Beam}(i_1, j_1, n_1, hs_1, i_2, j_2, n_2, hs_2)\end{aligned}$$

for hollowing factor $h \in (0, 1)$.

5.4.1 Example: Extended hub compound primitive

We define an **extended hub** as a union of all beams incident on a node. An extended hub differs from a hub because it is a union of entire beams instead of half-beams. An extended hub incident on node $\mathrm{N}[i, j, n]$ may be expressed as

$$\mathrm{ExtendedHub}(i, j, n, s_1, s_2) = \bigcup \mathrm{Beam}(i, j, n, s_1, i_z, j_z, n_z, s_2)$$

for all beams incident on $N[i, j, n]$, where $N[i_z, j_z, n_z]$ represents the opposite node on the beam.

5.4.2 Example: Beam connectors

A **beam connector** is a hollowed solid that can hold (connect) the beams that are incident on a node, as shown in Figure 5.3 and Figure 5.4. A beam connector for node $\mathrm{N}[i, j, n]$ may be expressed as

$$\mathrm{Connector}(i, j, n, s_1, s_2, r) =$$
$$\mathrm{Ball}(i, j, n, r) \cap \mathrm{ExtendedHub}(i, j, n, s_1, s_1) - \mathrm{ExtendedHub}(i, j, n, s_2, s_2)$$

where $s_1 > s_2$ such that s_1 adjusts the outer radii of the hollow connector beams and s_2 adjusts the inner radii of the hollow connector beams, and r is the radius of a ball that adjusts the lengths of the hollow connector beams.

5.5 PMC queries

Here we discuss PMC to determine if a query point Q is inside or outside of the CLG solid S.

First, consider a naive approach to PMC which uses a CSG tree T. PMC is first performed against each leaf node's primitive. Then, the results are propagated up the tree to

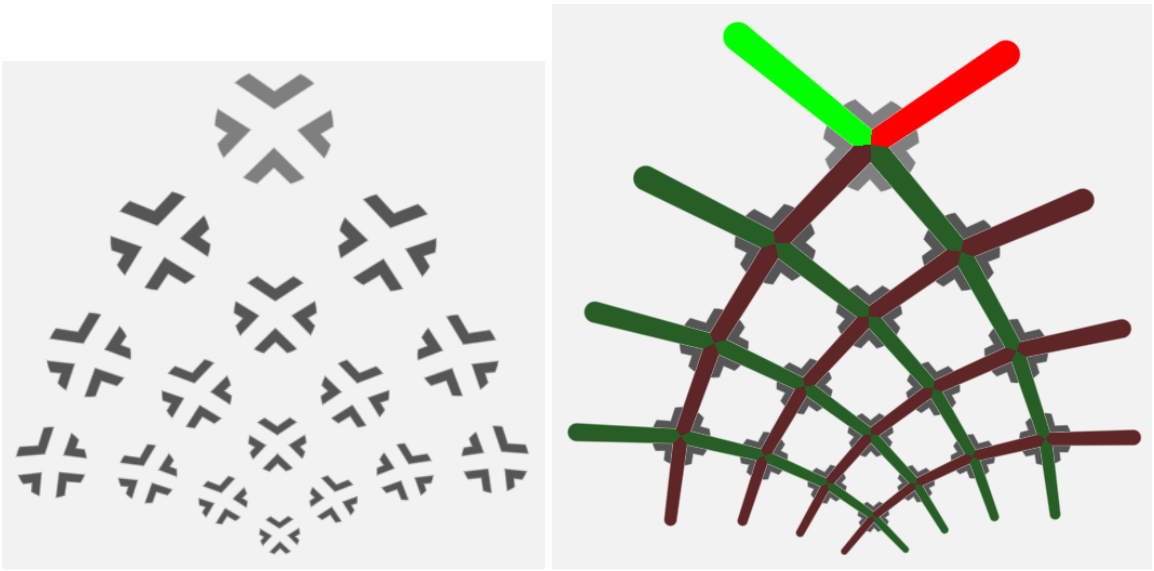


Figure 5.3: (Left) A pattern of connectors and (Right) the pattern of connectors holding the beams of a lattice.

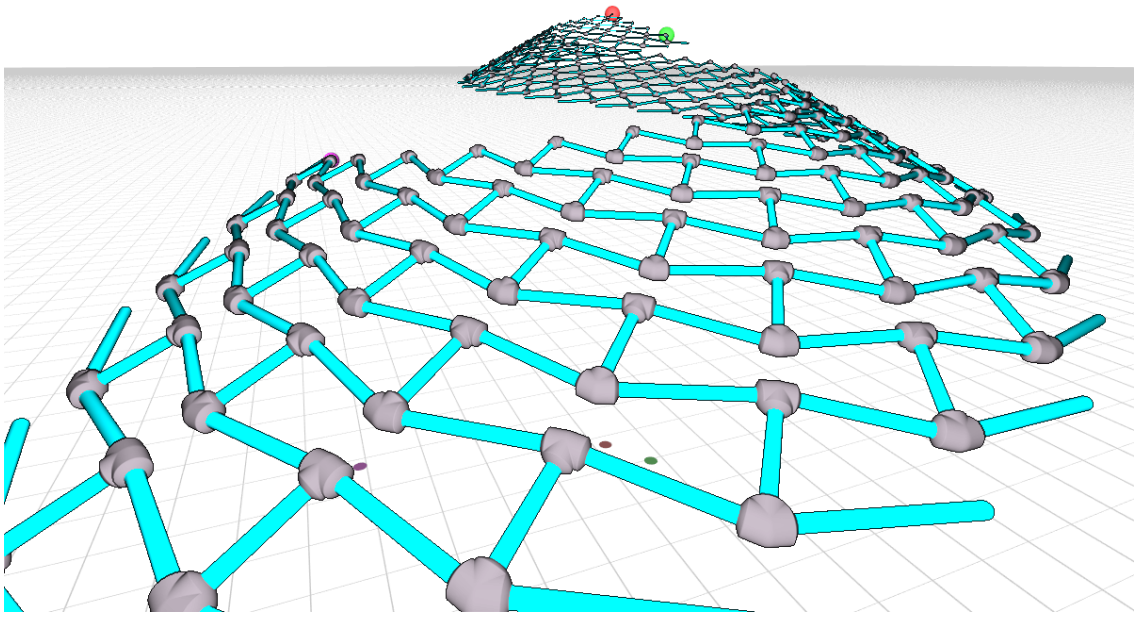


Figure 5.4: A 3D BeCOTS lattice with connectors.

the root by performing the proper Boolean operation for each branch node. T contains $\overline{c}$ subtrees that each represent a chunk union. Each subtree that represents a chunk union has $\overline{u} \times \overline{v}$ subtrees that each represent a chunk. For this naive approach to PMC, we traverse the entire tree T, which takes $\mathbf{O}(\overline{u}\,\overline{v})$ time. Note that for this time complexity, we ignore the number of chunk unions and the number of primitives per chunk because these values

typically do not vary with the size of a lattice. Also, the number of chunk unions and the number of primitives per chunk tend to be small compared to the total repetition count per chunk union.

This naive approach to PMC can be improved by exploiting the regularity of BeCOTS patterns. Consider the classification of Q against a chunk union. A chunk union is a BeCOTS pattern of chunks, so PMC can be performed in constant-time against a chunk union. This significantly improves the total time complexity to $\mathbf{O}(1)$.

Also, note that our CLG representation implicitly represents the CSG tree, so the CSG tree T does not need to be explicitly constructed.

PMC may be further accelerated by initially testing and rejecting query points that do not lie inside a bounding volume of S. S may be tightly bounded by a **conical shell**, the volume between two cones that share the same apex, where the apex is the fixed point of the BeCOTS similarities, so we propose using a conical shell as the bounding volume, such as is pictured in Figure 5.5. Note that in our figure, the bounding conical shell is not as tight as possible. For simplicity, we computed the conical shell to tightly bound the union of the smallest balls that bound each cone beam. If the CLG solid interferes with the axis of rotation of the BeCOTS similarities, then the conical shell degenerates into a cone. Although a bounding volume test does not offer an increase in asymptotic time complexity, it may still offer a significant time saving.

5.6 Voxelization

Voxelizing a CLG may be useful for 3D printing, for physical property analysis, and for visualization. Here we consider the voxelization of a CLG solid S into a grid of $\overline{m} \times \overline{n} \times \overline{o}$ voxels.

The simplest approach to voxelization of S is to do a PMC query against S for the center of each voxel and classifying each voxel (as in or out) based on the result of the query. For naive PMC queries, this approach takes $\mathbf{O}(\overline{m}\,\overline{n}\,\overline{o}\,\overline{u}\,\overline{v})$ time, due to the need to

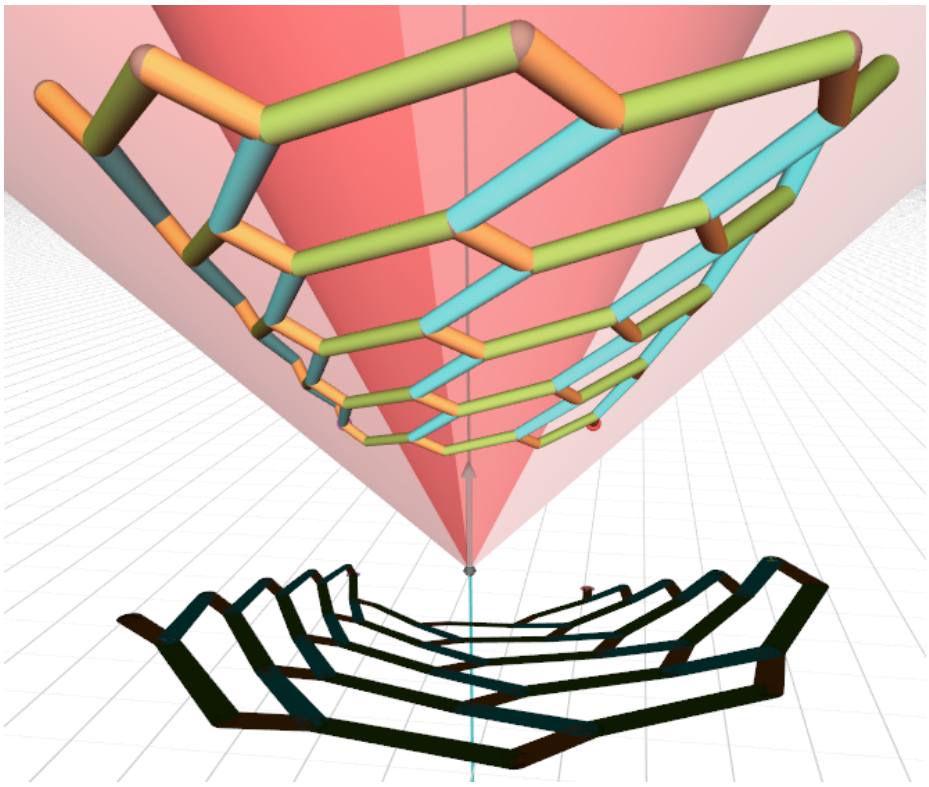


Figure 5.5: A conical shell bounding a BeCOTS lattice, where the outer and inner cone surfaces are shown in translucent red. The apex is the fixed point of the BeCOTS similarities. This conical shell does not bound the lattice as tightly as possible (i.e., the inner and outer surfaces are not tangent to the lattice), because, for simplicity, we computed the conical shell to tightly bound the union of the smallest balls that bound each cone beam.

operate on each chunk for each voxel. However, the constant-time PMC query accelerates voxelization to take $\mathbf{O}(\overline{m}\,\overline{n}\,\overline{o})$ time. Finally, voxelization via $\overline{m} \times \overline{n} \times \overline{o}$ PMC queries is embarrassingly parallel and may be accelerated by a Graphics Processing Unit (GPU).

5.7 Ray intersection queries

Here we discuss how to compute the first intersection X of a ray R, with origin point O and direction $\vec{D}$, against a CLG solid S. Ray intersection queries are useful for visualization and for some property analysis. Figure 5.4 and Figure 5.6 show example 3D CLG structures that have been rendered using the ray intersection process described below. The described ray intersection process is iterative (a modification of sphere tracing [15]) and

supports progressive rendering so that, when the view does not change, the rendered image improves over time. Figure 5.6 shows an example of progressive rendering before and after convergence. We implemented our renderer in GLSL fragment shaders, and, on an NVIDIA GeForce RTX 3080 GPU, the structures shown can be viewed in real time such that the image of the majority of the structure has converged in one rendering frame. Typically, rays converge fastest for intersections near the viewpoint, and convergence takes longer for rays that travel a far distance near a surface without intersecting it.

Before we discuss our approach, consider the task of efficiently culling sets of primitives that R is guaranteed not to intersect, without precomputing and using any spatial acceleration data structures such as Bounding Volume Hierarchies (BVH) or octrees. Consider an inconvenient scenario where R passes near every primitive that composes S without intersecting any of them. This case highlights the difficulty of quickly determining a small but relevant subset of the primitives of S that must be tested for intersection. Additionally, consider a case where R intersects all primitives in S but where the first intersection is the correct query output. It is difficult to quickly cull the primitives that do not need to be tested for intersection.

We propose to keep the intersection query as local as possible, to reduce the amount of work that must be performed at any time, by ignoring distant portions of S. We accomplish this by computing the intersection X via sphere tracing [15], which is a process that starts at ray origin O and incrementally traces along R by iteratively stepping a safe distance that is guaranteed not to step through S. Typically, for sphere tracing, the safe stepping distance s is chosen to be the distance d from the current point Q to the boundary of solid S. Often, s is chosen to be an approximation of the distance d that is guaranteed to be less than d while ensuring that the process does not converge before reaching the boundary of S. However, it may sometimes be safe and advantageous to take a step that is longer than d as long as we can guarantee that the step does not pass through the boundary of S.

The choice to use sphere tracing for computing ray intersection means that we need

a strategy to quickly compute a safe, but ideally long, distance s to step along R from a given point Q on R. Before discussing any acceleration strategies, consider a naive implementation of the sphere tracing approach. For every primitive G in the leaf nodes of the CSG tree T, compute and store the distance between G and Q. Then, propagate the distances up the tree T such that a distance is computed for each branch node as a function of its child nodes' distances. For union nodes, let the distance be the minimum of all its child nodes' distances. For intersection nodes, let the distance be the maximum of all its child nodes' distances. And, for subtraction nodes, let the distance be the maximum of its left child node's distance and of the negative of its right child node's distance. Following these rules results in a distance s at the root node that is a safe distance to step from Q along R.

Unfortunately, this naive approach is prohibitively slow. It requires processing each primitive of S, which takes $\mathbf{O}(\overline{u}\,\overline{v}\,\overline{w})$ time. Worse, this process must be repeated for every iteration of the sphere tracing process. From this naive approach, we consider acceleration strategies, to significantly reduce the number of primitives that must be processed to a constant (with respect to the repetition counts) per sphere tracing step.

First, consider the case where Q is outside of the bounding conical shell L. If R does not intersect L, then R never intersects S. Otherwise, it is safe to step the distance from Q to the first intersection with L.

Now, consider that Q is on or inside L. Before processing any primitives of S, we pick a distance r that is to be the maximum distance we will consider stepping for the current sphere tracing iteration. If a primitive G is farther away from Q than the distance r, then there is no risk of stepping through G during this step, so G can be safely ignored. If r is too large, then the current sphere tracing step may need to process too many primitives of S. However, if r is too small, then the sphere tracing process may require too many iterations before converging. Later, we discuss a strategy for choosing r.

A BIQ can be performed on a BeCOTS pattern in constant-time, for a well chosen

query ball radius. Therefore, a BIQ can be performed on a chunk union in constant-time, because each chunk union is a BeCOTS pattern of chunks. Let a query ball B have center Q and radius r. The primitives of the chunks that interfere with B are the only ones that must be considered for a given sphere tracing step. Therefore, for a well chosen r, each sphere tracing step has a constant-time cost relative to the repetition counts.

We use the following heuristic to choose the query radius r. First, compute a tight bounding sphere Y, with radius b and center B, around the template chunk $\mathrm{C}[0,0]$. We want to know, if we made a BeCOTS pattern with Y as the template shape, what would be the radius of an instance I of Y near the query point Q. We let $r = b|\overrightarrow{FQ}|/|\overrightarrow{FB}|$, where F is the fixed point of the BeCOTS field. This choice of r is the radius of the resulting sphere after dilating Y about F until its center is the same distance from F as Q.

One last acceleration strategy to consider is that, when computing the distance between query point Q and the primitive G, we only care about the distance along the ray R. So, if a closed-form ray intersection function is known for G, then it can be used to help the sphere tracing process converge more quickly. This is especially helpful to accelerate sphere tracing when a ray traces a long distance near a primitive without actually intersecting it. Fortunately, our ball and cone beam primitives both permit fast, closed-form ray intersection computations.

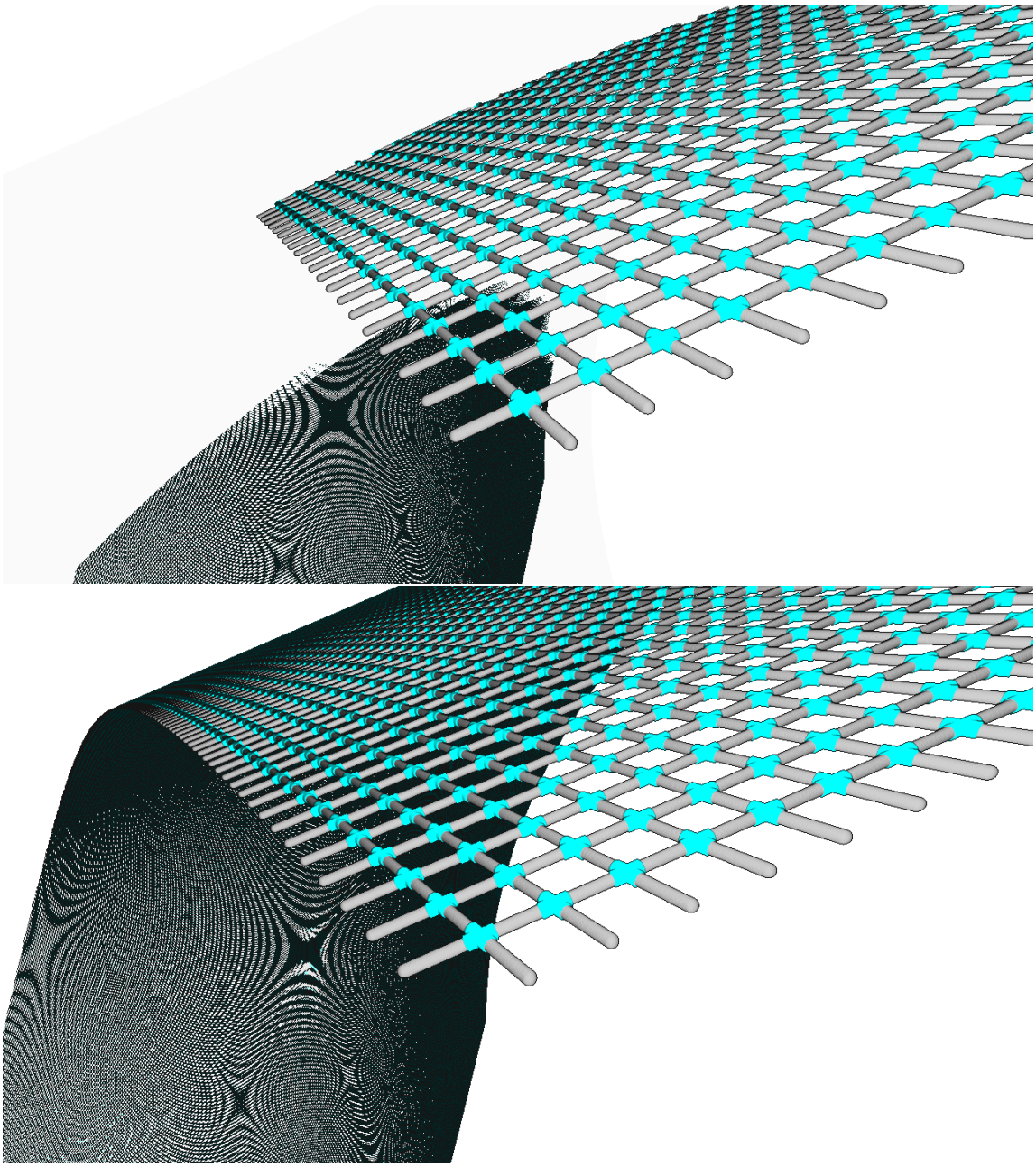


Figure 5.6: (Top) The CLG ray intersection process described in section 5.7 is iterative such that using more iterations yields more accurate results. Therefore, a progressive renderer may be implemented so that, when the view does not change, the rendered image improves over time as more iterations complete. Here, we show a view of a 1000×1000 CLG before the rendering has converged. (Bottom) A view of the same CLG after rendering has converged.

CHAPTER 6

EXTENSIONS TO 3-DIRECTIONAL STEADY LATTICES (BRICKS)

6.1 Steady bricks

6.1.1 Motivation

This thesis has primarily focused on slab lattices (2-directional), for simplicity, up until this point. However, the steady lattice forumlation provided need not be limited to slabs, so we present here a generalization to brick lattices (3-directional). This generalization is useful for modeling volume-like structures, in addition to the surface-like structures, which we have already demonstrated.

Note, however, that some of the useful properties of BeCOTS lattices, discussed in chapter 4, cannot exist in 3-directional structures. In section 6.3, we discuss a special case, BeCOTS Stacks, of brick lattices that preserves some of the useful properties.

6.1.2 Representation

Like for the steady slabs, the main component behind a steady brick maps, patterns, and lattices is the **steady 3-field** $\mathbf{S}(u, v, w) = \mathbf{W}^w \circ \mathbf{V}^v \circ \mathbf{U}^u$, where **U**, **V**, and **W** are similarities and u, v, and w are real numbers.

Then, we define a **steady 3-map** to have the form $\mathbf{M}(u, v, w) = \mathbf{S}(u, v, w) \circ P$, where P is a point.

The shapes of a **steady 3-pattern** P have the form $\mathrm{P}[i, j, k] = \mathbf{S}(i, j, k) \circ \mathrm{P}[0, 0, 0]$, where $\mathrm{P}[0, 0, 0]$ is the template shape and $i \in [0, \overline{u})$, $j \in [0, \overline{v})$, and $k \in [0, \overline{w})$ are integer indices with repetition counts $\overline{u}$, $\overline{v}$, and $\overline{w}$.

The nodes N of a steady brick lattice are organized as a steady 3-pattern such that $\mathrm{N}[i, j, k, b]$ refers to the node with ID n in node-group $\mathrm{N}[i, j, k] = \mathbf{S}(i, j, k) \circ \mathrm{N}[0, 0, 0]$.

The lattice beams are defined in terms of N and organized as $\overline{b}$ beam-patterns where $\mathrm{B}[b]$ refers to the b^{th} beam-pattern, as discussed in subsection 2.1.2.

The properties of steady bricks may be justified with the same strategies used to justify the properties of steady slabs, so here we simply list the results without a complete discussion and justification.

6.1.3 Steadiness properties

Every node in N belongs to a steady 1-pattern in the **U**, **V**, and **W** directions.

A beam that connects two nodes within the same node-group belongs to a steady 1-pattern in the **U**, **V**, and **W** directions.

A beam that connects two nodes in different node-groups belongs to a steady 1-pattern in the **W** direction but generally does not belong to a steady 1-pattern in either the **U** or the **V** directions.

Similarly, a hub belongs to a steady 1-pattern in the **W** direction but generally does not belong to a steady 1-pattern in either the **U** or the **V** directions.

6.1.4 IPC queries

Consider the hubs H of a steady brick lattice and let s_w be the scaling factor of similarity **W**. If $s_w \neq 1$, then

$$\textbf{Area}(\mathrm{H}) = \frac{1 - {s_w}^{2\overline{w}}}{1 - {s_w}^{2}} \sum_{j=0}^{\overline{v}-1} \sum_{i=0}^{\overline{u}-1} \textbf{Area}(\mathrm{H}[i, j, 0])$$

$$\textbf{Volume}(\mathrm{H}) = \frac{1 - {s_w}^{3\overline{w}}}{1 - {s_w}^{3}} \sum_{j=0}^{\overline{v}-1} \sum_{i=0}^{\overline{u}-1} \textbf{Volume}(\mathrm{H}[i, j, 0])$$

Otherwise, if $s_w = 1$,

$$\textbf{Area}(\mathrm{H}) = \overline{w} \sum_{j=0}^{\overline{v}-1} \sum_{i=0}^{\overline{u}-1} \textbf{Area}(\mathrm{H}[i, j, 0])$$

$$\textbf{Volume}(\mathrm{H}) = \overline{w} \sum_{j=0}^{\overline{v}-1} \sum_{i=0}^{\overline{u}-1} \textbf{Volume}(\mathrm{H}[i, j, 0])$$

6.1.5 PMC queries

We discuss, in detail, PMC queries on steady brick patterns and lattices in chapter 7. In fact, we discuss a useful generalization of PMC called Ball Interference Query (BIQ), which tests whether or not a query ball interferes with a steady brick lattice. In typical cases, such a BIQ can be completed in $\mathbf{O}(\overline{u}\,\overline{v})$ time, when using our RangeFinder acceleration algorithm.

6.2 Programmed Lattice Editor (PLE)

To facilitate the design of steady brick (and slab) lattices, we present here the **Programmed Lattice Editor (PLE)**. In Programmed Lattice Editor (PLE), a lattice is represented by a small program that, given a set of input parameters, generates a complete yet unevaluated intermediate representation of a steady lattice. This intermediate representation may used by PLE to either evaluate or to query either the entire or a selected portion of the lattice. A lattice in PLE can be programmed like a traditional text-based program. However, PLE also provides a GUI with which a designer/programmer can visually manipulate a lattice and the editor can convert the visual changes into code that can be pasted into the program representation of the model.

The intermediate representation of a lattice in PLE is the standard representation we used throughout this thesis. PLE stores the similarities **U**, **V**, and **W**, the repetition counts

$\overline{u}$, $\overline{v}$, and $\overline{w}$, the $\overline{n}$ nodes of the template node-group $\mathrm{N}[0,0,0]$, and the $\overline{b}$ beam-patterns B.

6.2.1 PLE code API

A PLE programmer may start with a simple **template program** from which a lattice may be designed. Figure 6.1 shows such a template program along with the lattice that it represents.

The first part of the program represents the nodes in the template node-group. Here, a single node with ID a is created, centered at $(0,0,0)$ with a radius of $.1$.

The second part of the program specifies the beam-patterns, where each line represents a different beam-pattern. For example, the first beam-pattern specified represents **BeamPattern**$(a,1,0,0,a)$ which connects node $\mathrm{N}[i,j,k,a]$ to $\mathrm{N}[i+1,j,k,a]$, for all valid i, j, and k.

The third part specifies the incremental similarities **U**, **V**, and **W** and the repetition counts u, v, and w. In this example, the repetition counts are each 3, and the incremental similarities are orthogonal, unit-length translations.

A PLE programmer can expose one or more values, in the program representation, as input parameters so that a single program can represent a family of lattices. For example, an input parameter may be exposed for controlling beam thickness. A designer may manually tweak the parameter, or the parameter may be automatically tweaked by a program that is optimizing the lattice to meet some design constraints. Input parameters may also be used to trigger conditional branches in the code.

6.2.2 PLE GUI

Through a series of graphical manipulations, a designer/programmer can modify the programmed lattice. PLE allows for adding, removing, and transforming the nodes of the template node-group. It allows for adding and removing beams. And, it allows for changing the repetition counts and for transforming the incremental similarities. Figure 6.2 shows an example of changing an incremental similarity by dragging, rotating, and dilating a local

```
Node a = addNode(Pt(0,0,0), .1);

addBeam(a,a, 1,0,0);
addBeam(a,a, 0,1,0);
addBeam(a,a, 0,0,1);

setU(3, Translation(1,0,0));
setV(3, Translation(0,1,0));
setW(3, Translation(0,0,1));
```

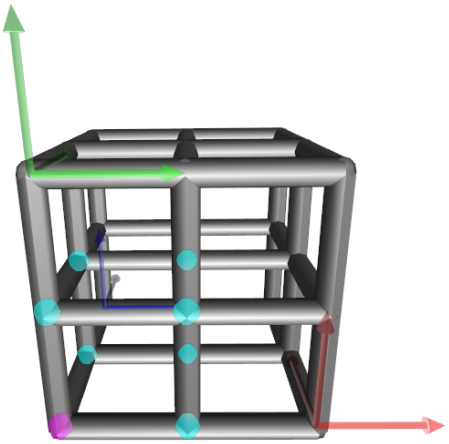


Figure 6.1: A PLE template program and the corresponding lattice, shown in the GUI editing environment. The GUI provides manipulatable frames for specifying either the cumulative or the incremental similarities. The GUI also displays the nodes from a subset of the node-groups, allowing the user to add or remove nodes and beams and to transform nodes. The magenta node represents the template node-group and the cyan nodes represent its neighbors.

coordinate frame.

PLE converts graphical manipulations into code that can be pasted into the original program, to save the changes. For example, Figure 6.3-Right shows a lattice that has been created entirely via graphical manipulations of the template lattice in Figure 6.1. Each graphical manipulation generates code that can be manually pasted into the original program. Changes to the code resulting from graphical manipulations are highlighted in Figure 6.3-Left.

A lattice may contain millions or billions of beams, which is prohibitively expensive for real time rendering. However, we wish to give designers the ability to get both a large-scale and a small-scale view of a designed lattice. To accomplish this, we allow the designer to select a small portion of the lattice in which the nodes and beams are to be fully rendered. Then, the nodes and beams of the lattice outside of the selected portion are not rendered, but instead, the isocurves of the underlying steady 3-map are displayed. We call the isocurve display the **cage view**, because it looks like a cage that shares the global shape of the designed lattice. Due to the periodicity of the small-scale structures, the combined display

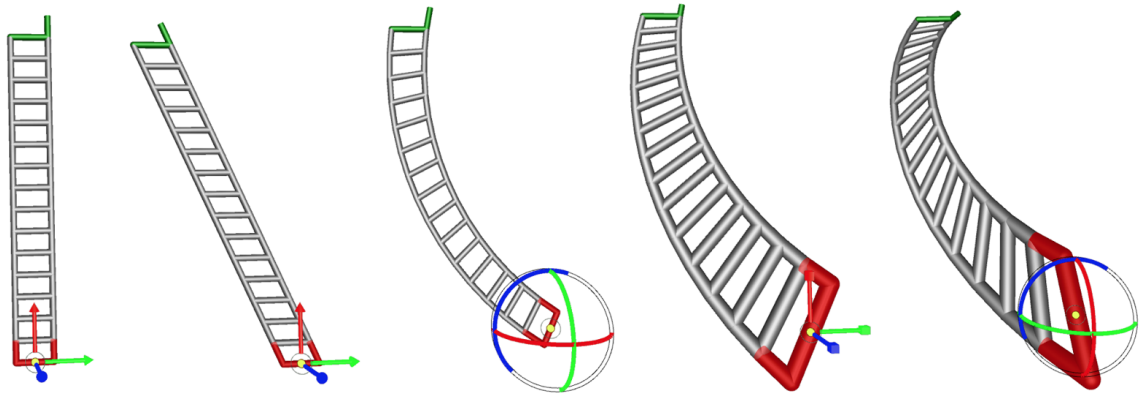


Figure 6.2: A 1-directional lattice is interactively edited by dragging, rotating, and dilating the red handle using mouse motions, which modifies the similarity that takes the red handle to the green handle. The intermediate beams are automatically adjusted so that the lattice remains steady.

```
Node a = addNode(Pt(0,0,0), .0645);

addBeam(a,a, 1,0,0);
addBeam(a,a, 0,1,0);
addBeam(a,a, 0,0,1);
addBeam(a,a, 0,1,1);
addBeam(a,a, 1,1,0);

setU(7, Scaling(1.187, Pt(-3.333,0,0)));
setV(12, Rotation(0.285, Vec(0,0,1), Pt(-1.555,0.003,0)));
setW(15, Swirl(0.224, Vec(0,1,0), Pt(10.872,0,0.039), 0.953));
```

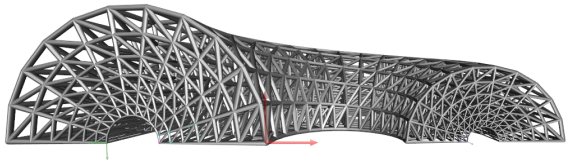


Figure 6.3: A PLE program and lattice, created entirely by quick and easy GUI manipulations on the lattice in Figure 6.1. The changes to the template program are highlighted.

of both the small and large scales allows for some intuitive imagining of how the cage may be filled with the small-scale structure. The designer has the ability to test this intuition by selectively rendering different portions of the lattice.

A strategy for creating watertight tessellations of lattices is discussed in [47]. This strategy creates a separate tessellation for all hubs of a lattice, which is an expensive $\mathbf{O}(\overline{u}\,\overline{v}\,\overline{w})$ time operation. However, in a steady lattice, all hubs belong to a steady 1-pattern in the **W** direction. Therefore, if we tessellate only the hubs $\mathrm{H}[i, j, 0, n]$, for all valid i, j, and n, then the remaining hubs can be obtained as similar copies of the already tessellated hubs. Note that care must be taken to ensure the triangle vertices and edges line up at the disk interface between neighboring hubs.

In special cases, such as a BeCOTS lattice, all hub-groups are pairwise similar such

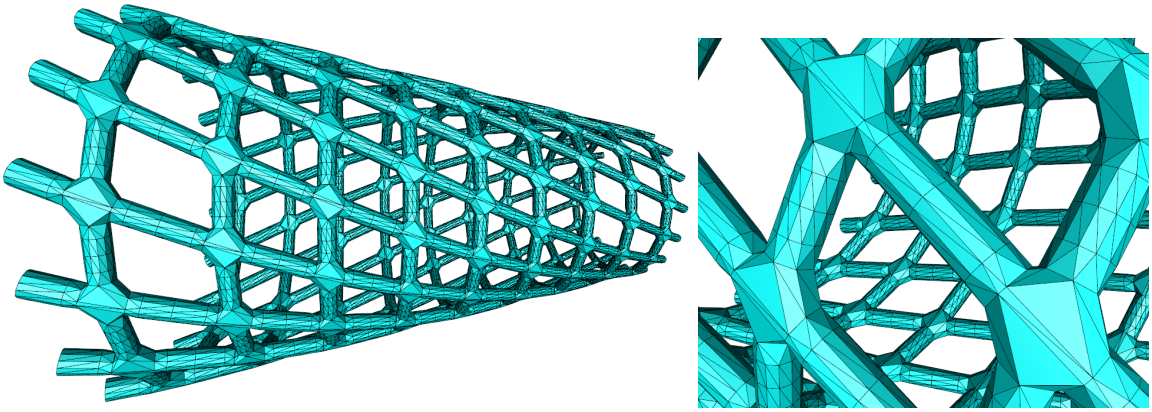


Figure 6.4: A tessellation of a BeCOTS lattice. This was produced by generating a single tessellation of the template hub-group and then making several similar copies of it.

that a full-lattice tessellation can be achieved after tessellating and making similar copies of only the template hub-group. An example tessellation of a BeCOTS lattice is shown in Figure 6.4.

6.3 BeCOTS stacks

Here, we explore extending $\overline{u} \times \overline{v}$ BeCOTS lattices into $\overline{u} \times \overline{v} \times \overline{w}$ lattices that preserve some of their useful properties. We present **BeCOTS Lattice Stacks**, which consist of $\overline{w}$ BeCOTS lattice **layers** that each have the same BeCOTS field **S** and $\overline{u} \times \overline{v}$ repetition counts. Each layer is joined by at least one $\overline{u} \times \overline{v}$ beam-pattern. The warp and placement of the first layer may be controlled as usual, by the four corners of a BeCOTS map. Then, the warp and placement of each subsequent layer is defined simply by changing the location of the starting corner A of its map. The other corners are defined by corner A and the common field **S**. The choice of how corner A changes between layers may affect the properties of the Stack (see Figure 6.5 for examples of two different choices). For example, if corner A has an arbitrary position between layers, then PMC (subsection 4.4.4) against query point Q will require $\mathbf{O}(\overline{k})$ separate PMC queries of Q against each layer. However, the cost of PMC against the Stack may be reduced to $\mathbf{O}(1)$ for certain strategies of changes in corner A. For example, either by raising (subsection 4.3.1) each consecutive layer's A by a constant-

distance d or by normal offsetting the i^{th} layer (from the first layer's defining cone) by distance di for some initial distance d (see Figure 6.5). This is done by first computing the two nearest layers to Q, then only performing PMC queries against the patterns of the two layers and against the beam-patterns connecting them (assuming Q is sandwiched between two layers, otherwise just compute and test the nearest layer). In these two examples, we compute the two nearest layers by mapping Q onto the line L through the regular sequence of A corners by first rotating Q around the axis through fixed point F with direction $\vec{T}$ (until it lies on the plane through F and L) then projecting it toward F and onto L, yielding a point Q' on the line L through the regular sequence of A corners. Then, the two nearest A corners to Q' correspond to the two nearest layers to Q.

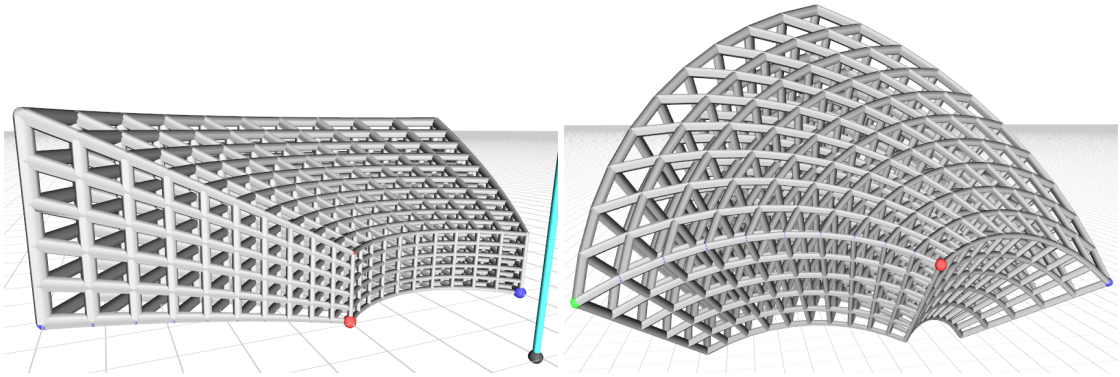


Figure 6.5: A BeCOTS Stack lattice where the bottom layer is a planar COTS and each subsequent layer is a constant-distance raise (subsection 4.3.1) from the BeCOTS of the previous layer (left). Another stack where no layer is planar and the i^{th} layer is a normal offset by distance di, for some d, from the cone of the first layer (right).

CHAPTER 7

RANGEFINDER AND BALL INTERFERENCE QUERIES (BIQS) FOR STEADY BRICK PATTERNS AND LATTICES

The geometric representation of a lattice must support certain geometric queries needed for the lattice's physical analysis. We focus on the **Ball-Interference Query (BIQ)**, which establishes whether or not a query ball B intersects the lattice, and if so, returns the index-sets of all the nodes and beams that it intersects.

BIQs are fundamental queries, because several other useful queries may be created from them. Point-Membership Classification (PMC) returns the nodes and beams that contain a query point and can be formulated as a BIQ with a query ball with zero radius. Local shortest-distance and local nearest-neighbor queries, with a query point C and search distance r, can be performed by retrieving the nearby lattice nodes and beams via a BIQ with a query ball of center C and radius r, then computing the shortest distance from C to each returned element. Hoffmann et al. suggest (with different but compatible query definitions) that the PMC and shortest-distance queries are the only ones needed to support all other geometric queries [16]. For example, ray-intersection queries may be implemented with sphere-tracing [15], built from the local shortest-distance query, and may be used to render a lattice. A lattice may be voxelized, for physical analysis or additive manufacturing, by organizing many balls or points into a grid and performing a BIQ for each.

A naïve BIQ implementation tests the query ball against each node and beam of the lattice. We write the time complexity of a naïve BIQ simply as $\mathbf{O}(\overline{u}\,\overline{v}\,\overline{w})$, hence omitting the number of nodes per group and the beam-count to node-count ratio, since the number of nodes and beams per node-group is often small compared to $\overline{u}$, $\overline{v}$, and $\overline{w}$. A micro- or nano-structure may contain billions of beams, making naïve BIQs very costly.

We present our **RangeFinder** solution for accelerating BIQs on steady lattices. It re-

duces the time complexity of a BIQ from $\mathbf{O}(\overline{u}\,\overline{v}\,\overline{w})$ to $\mathbf{O}(\overline{u}\,\overline{v})$, without using a spatial occupancy data structure. This roughly corresponds to a 100-to-1 time-cost reduction for a lattice with 100^3 node-groups, which is a modest complexity when dealing with micro- and nano-structures.

For a restricted set of steady lattices that meet conditions listed in section subsection 7.3.1, RangeFinder reduces the time-complexity of BIQ to $\mathbf{O}(\overline{u})$ and possibly even to $\mathbf{O}(1)$.

In chapter 9, we discuss the use of RangeFinder accelerated BIQs to create a class of hierarchical lattices that we call **Lattice-in-Lattice (LiL)**. However, at the end of this chapter, we present test results for the RangeFinder acceleration of BIQs when used to generate LiL structures.

This chapter is organized as follows:

- In section 2, we present our RangeFinder algorithm for accelerating a BIQ on a subset of a steady lattice, called a steady row of nodes.
- In section 3, we extend the algorithm to handle steady rows of beams.
- In section 4, we describe how we user RangeFinder on steady rows of beams to accelerate BIQs on steady lattices.
- In section 5, we present experimental results demonstrating the effectiveness of RangeFinder.

7.1 RangeFinder Overview

The key feature, proven in section 3.3, of steady lattices is that their nodes and beams can be organized into **steady rows** (also called steady 1-patterns) which have the form $\mathrm{R}[k] = \mathbf{S}^k \circ \mathrm{R}[0]$, where $\mathrm{R}[0]$ is the **template shape** (node or beam) and **S** is the **incremental similarity**, which transforms $\mathrm{R}[k]$ to $\mathrm{R}[k+1]$.

In this section, we present and justify our **RangeFinder** solution to the problem of Ball-Interference Queries (BIQs) on a steady row R of balls, where a row is formed as the set of nodes of a steady brick lattice with a fixed i, j, and b for all valid values of k, i.e. $\mathrm{R}[k] = \mathrm{N}[i, j, k, b] = \mathbf{W}^k \circ \mathrm{N}[i, j, 0, b]$.

In section section 7.2, we extend this solution to handle BIQs on a steady row of beams.

7.1.1 Overall strategy for RangeFinder

We start here by stating the problem, outlining the general nature of our strategy, and revealing the mathematical formulation that it is based on.

PROBLEM: Given a query ball Q and a steady row R with incremental similarity **S**, repetition count $\overline{n}$, and template shape $\mathrm{R}[0]$, find a conservative, but hopefully tight, **candidate range** (integer interval) ***C*** that contains the indices of all shapes $\mathrm{R}[k] = \mathbf{S}^k \circ \mathrm{R}[0]$ that interfere with Q.

OVERALL STRATEGY: We divide the problem into two simpler RangeFinder subproblems and return the intersection of the candidate ranges produced as solutions to these.

This approach uses a particular **canonical decomposition** of **S**, which is justified by the following property:

CANONICAL DECOMPOSITION: Similarity **S** may always be decomposed into a commutative product of two **primitive similarities**, one of which is a rotation and the other is either a translation or a dilation.

The dilation factor d is the cube-root of the determinant of the 3x3 matrix that represents the linear part of **S** [43]. Depending on the **dilation factor** d of **S**, one of the following two decompositions always exists:

1. If $d = 1$, $\mathbf{S} = \mathbf{R} \circ \mathbf{T}$, the product of a rotation **R** around an axis A with a translation **T** parallel to A.

2. If $d \neq 1$, $\mathbf{S} = \mathbf{R} \circ \mathbf{D}$, the product of a rotation **R** around an axis A with a dilation **D** about a fixed point F on A.

For simplicity, we represent incremental similarities each by an orthonormal frame plus the dilation factor.

We call the sub-type of the similarity associated with the $d \neq 1$ case a **swirl**. The sub-type for the case when $d = 1$ is a **screw**, and although a screw may be considered as a special case of the general family of swirls, we use a different derivation to compute the decomposition and the candidate range.

The canonical decomposition of a screw is discussed in numerous papers and books (see for example [25]). The canonical decomposition for the swirl is presented in [14]. We discuss both screws and swirls in detail in section 2.5. Note that these decompositions include special cases in which one or both primitive similarities degenerate to identities. These special cases may be easily detected and may require special treatments that are simple and hence are not discussed in this paper.

In the next section, we assume that **S** has been decomposed into its two primitive similarities as either $\mathbf{R} \circ \mathbf{T}$ or $\mathbf{R} \circ \mathbf{D}$, and we explain how RangeFinder identifies the candidate range for each one of the three possible primitive similarity types: **T**, **D**, **R**.

7.1.2 RangeFinder for a primitive similarity

First, we present the essence of the RangeFinder solution, regardless of the type (**T**, **D**, or **R**) of the particular primitive similarity **S**.

We want to compute the **range** (interval of consecutive integer values) that identifies the indices k of candidate instances $\mathrm{R}[k]$ that may interfere with Q. We need the range to be conservative, i.e., to guarantee that instances for indices outside of that range do not interfere with Q. We strive to produce a tight range, i.e., to reduce the number of false-positives, at least in common configurations.

To solve this problem, first, we define a **normalized map**, **M**, from space to the real numbers, which satisfies a key property defined below. Then, we compute the **template-extent**, $[s_0, e_0]$, which bounds the image of $\mathrm{R}[0]$ by **M**, and the **query-extent** $[s_q, e_q]$, which

bounds the image of Q by **M**. Both extents are intervals of real numbers. From these, we compute the range, of which the complement identifies the indices k of the instances $\mathrm{R}[k]$ that are guaranteed not to intersect Q.

DEFINITION: M is a normalized map if and only if, for every point X in the valid domain, $\mathbf{M}(\mathbf{S}^k \circ X) = \mathbf{M}(X) + k$.

THEOREM:

Q may interfere with $\mathrm{R}[k]$ only if $k \in \boldsymbol{C} = [\mathbf{max}(0, \lceil s_q - e_0 \rceil), \mathbf{min}(n, \lfloor e_q - s_0 \rfloor)]$.

PROOF: Shape $\mathrm{R}[k]$ is mapped to the extent $[s_0 + k, e_0 + k]$. So, Q may intersect with $\mathrm{R}[k]$ only if $[s_0 + k, e_0 + k]$ and $[s_q, e_q]$ overlap, yielding the conditions $s_q \leq k + e_0$ and $k + s_0 \leq e_q$, i.e., if $k \in [s_q - e_0, e_q - s_0]$. The integer indices that lie in this interval are the members of integer interval $[\lceil s_q - e_0 \rceil, \lfloor e_q - s_0 \rfloor]$. Finally, the indices are clamped to $[0, n]$.

If $\boldsymbol{C}$ is empty, then Q interferes with no instances.

Thus, for each one of the three versions of RangeFinder for our three primitive similarities, we need to define the appropriate map **M** and provide a closed-form expression for computing the extent of the image of a ball under **M**. To determine the bounds of the image of a ball on the real number line, the extent computation is applied to both the template-shape and the query ball.

We focus on a single-ball template-shape because it is simple and because it may be used as a bounding container for steady rows of more complex shapes.

In the subsequent subsections, we consider primitive similarities **T**, **D**, and **R**, one at a time. For each such similarity, **X**, we explain how to define **M** (see definition marked by **X**-MAP) and how to compute the extent $[s, e]$ of a ball B, with center C and radius r, under **M** (see definition marked by **X**-EXTENT). Note that we use the same process to compute the extent when B stands for the template-ball and when it stands for the query-ball.

7.1.3 RangeFinder for translation T

Consider that **T** is a translation by vector $\vec{V}$. Let O be an arbitrarily chosen origin.

T-MAP: The **normalized translation map**, **M**, for a translation by vector $\vec{V}$ with chosen reference point O is $\mathbf{M}(X) = \overrightarrow{OX} \cdot \vec{V}/\vec{V}^2$.

JUSTIFICATION: $\mathbf{M}(X)$ is composed of a projection and a normalization. The expression $\overrightarrow{OX} \cdot \vec{V}/|\vec{V}|$ computes the projection (measured from O) of X onto a line passing through O and parallel to $\vec{V}$. The $1/|\vec{V}|$ normalization ensures that the measure reported by the map is expressed in the proper unit, so that, $\mathbf{M}(O + \vec{V}) = 1$ (see Figure 7.1).

T-EXTENT: The extent $[s, e]$ of B, as a ball of center C and radius r, for a translation by vector $\vec{V}$ is $\mathbf{E}(\mathrm{B}) = [\mathbf{M}(C) - r/|\vec{V}|, \mathbf{M}(C) + r/|\vec{V}|]$.

JUSTIFICATION: Extent $[s, e]$ defines the smallest slice of space orthogonal to $\vec{V}$ that contains B. Let $c = \mathbf{M}(C)$ be the image of the center C. We extend the extent in both directions around c to ensure that it covers the projection of B onto a line parallel to $\vec{V}$. However, we normalize this extension to be expressed in the proper unit (see Figure 7.1).

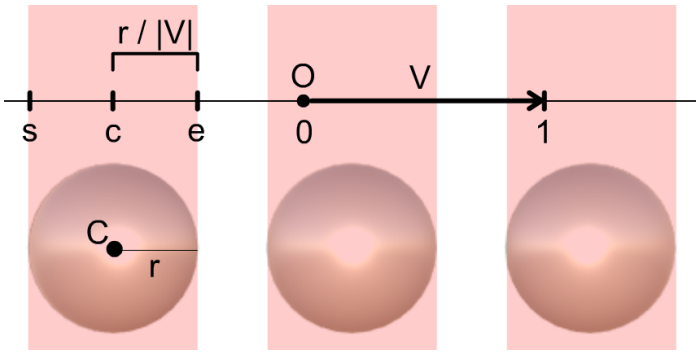


Figure 7.1: A row of balls where **S** is a translation by $\vec{V}$. The real number line is shown through O and parallel to $\vec{V}$ such that 0 on the real line corresponds to O and 1 corresponds to $O + \vec{V}$. A point in space is mapped to the real line based on its distance along the line, such that a translation in space by $\vec{V}$ results in an increase by 1 in its mapped value. The light-red regions indicate the extent of each ball on the real line, and $[s, e]$ is labeled for the left-most ball.

7.1.4 RangeFinder for dilation D

Consider that **D** is a dilation by factor $d > 0$ about fixed point F.

D-MAP: The **normalized dilation map**, **M**, for a dilation about fixed point F by dilation factor $d > 0$ is $\mathbf{M}(X) = \mathbf{log}_d(|\overrightarrow{FX}|)$.

JUSTIFICATION: Consider an arbitrary point P with distance $|\overrightarrow{FP}|$ from F. Applying a composition of k dilations, each by factor d, to P about F results in a new point X with distance $|\overrightarrow{FX}| = |\overrightarrow{FP}| * d^k$. So, letting $\mathbf{M}(X) = \mathbf{log}_d(|\overrightarrow{FX}|/|\overrightarrow{FP}|)$ with an arbitrarily chosen reference point P gives a function that returns the real number dilation power, k, needed to transform P to X. This has the desired property that $\mathbf{M}(d^k X) = \mathbf{M}(X) + k$. The normalized map may be written simply as $\mathbf{M}(X) = \mathbf{log}_d(|\overrightarrow{FX}|)$ by choosing P to be a unit distance away from F.

At point F, the image $\mathbf{M}(F)$ is undefined, because the logarithm of 0 is undefined. However, **M** can be defined for the entire domain and the extent can be computed when B contains F by letting $\mathbf{M}(F) = -\infty$ when $d > 1$ and $\mathbf{M}(F) = \infty$ when $0 < d < 1$.

D-EXTENT: The extent $[s, e]$ of B, as a ball of center C and radius r, for a dilation about fixed point F by dilation factor d is $\mathbf{E}(\mathrm{B}) = [\mathbf{min}(\mathbf{log}_d(|\overrightarrow{FC}| \pm r)), \mathbf{max}(\mathbf{log}_d(|\overrightarrow{FC}| \pm r))]$.

JUSTIFICATION: Let $\mathbf{sphere}(C, r)$ denote a sphere of center C and radius r. Extent $[s, e]$ defines a spherical shell, i.e. the solid bounded by the union of $\mathbf{sphere}(F, |\overrightarrow{FC}| - r)$ and $\mathbf{sphere}(F, |\overrightarrow{FC}| + r)$, such that the larger sphere contains B and the smaller sphere does not. When $d > 1$, $\mathbf{M}(X)$ increases as $|\overrightarrow{FX}|$ increases, so points on the smaller sphere map to s and points on the larger sphere map to e, because s must be smaller than e. However, when $0 < d < 1$, $\mathbf{M}(X)$ decreases as $|\overrightarrow{FX}|$ increases, so points on the larger sphere map to s and points on the smaller sphere map to e. Consequently, $s = \mathbf{min}(\mathbf{log}_d(|\overrightarrow{FC}| \pm r))$ and $e = \mathbf{max}(\mathbf{log}_d(|\overrightarrow{FC}| \pm r))$. (See Figure 7.2).

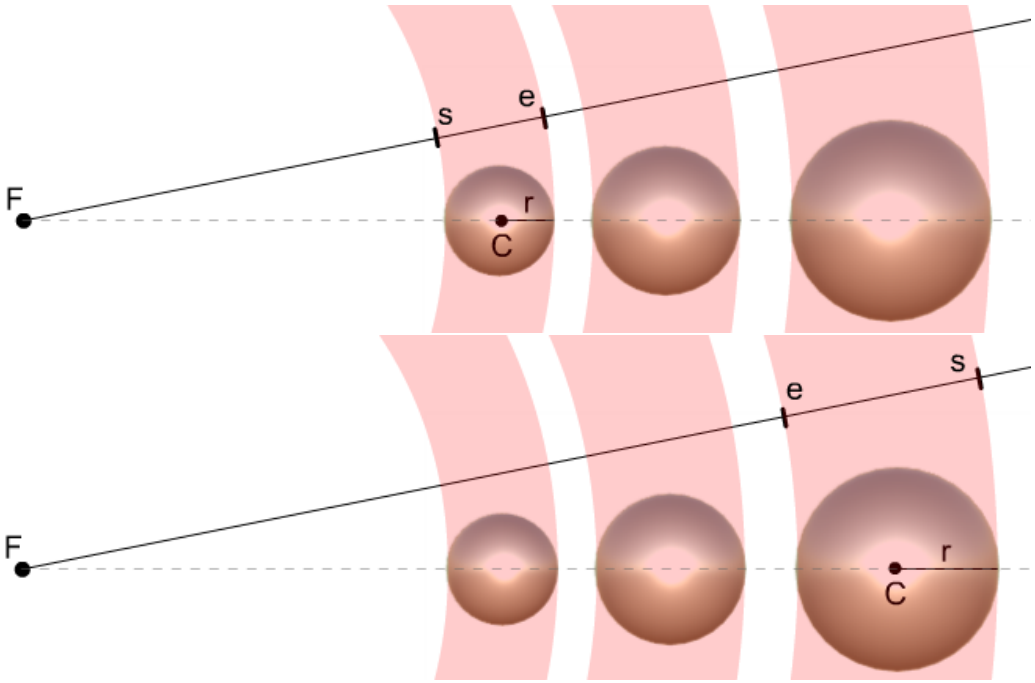


Figure 7.2: A row of balls where **S** is a dilation, by $d > 1$ on the top and by $0 < d < 1$ on the bottom. The "real number line" is visualized here by considering F to be a point at infinity. A point X in space maps to the real line as $\mathbf{log}_d(|\overrightarrow{FX}|)$. All points on a circle centered at F map to the same number, and the light-red annuli indicate the extent of each ball on the real line.

7.1.5 RangeFinder for rotation R

For clarity and elegance, we present here the RangeFinder solution in which we assume that the total angle sustained by the entire row is less than 2π. In other words, the row does not even complete a single **wrap** around its axis.

Such "tame" steady rows may be adequate for some applications, but not for others. Our extension to more general steady rows that wrap around the axis at least once applies RangeFinder to each wrap, where each application may yield a different candidate range. A previously computed candidate range for the other primitive similarity (**T** or **D**) can be used to reduce the number of wraps tested.

Consider that **R** is a rotation by angle θ around axis A. We assume below that A does not intersect ball B. When it does, RangeFinder returns the full range $[0, n]$. In practice, returning the full range is not an issue, because we assume the lattice's nodes are disjoint

and the query ball is relatively small, so the candidate range for translation or dilation will be small if the rotation range is not.

Let **angle**$(\vec{N}, \vec{V}, \vec{W})$ be the angle that rotates vector $\vec{N}$ to vector $\vec{V}$ around vector $\vec{W}$, assuming that $\vec{N}$ and $\vec{V}$ are perpendicular to $\vec{W}$ and that $|\vec{N}| = |\vec{W}| = 1$. It may be computed as **atan2**$((\vec{W} \times \vec{N}) \cdot \vec{V}, \vec{N} \cdot \vec{V})$.

R-MAP: The **normalized rotation map**, **M**, by rotation angle θ about axis A of direction $\vec{W}$ and given an arbitrary reference vector $\vec{N}$ orthogonal to A returns $\mathbf{M}(X, b) =$ **angle**$(\vec{N}, \overrightarrow{X_{[\mathrm{A}]}X}, \vec{W})/\theta + 2\pi b/|\theta|$, where $\overrightarrow{X_{[\mathrm{A}]}X}$ is the normalized vector from $X_{[\mathrm{A}]}$ (the closest projection of X onto A) to X, and where b, the branch ID, represents the number of full rotations around A to be added to the angle measurement. If no b is given, as in $\mathbf{M}(X)$, then **M** represents a map to the infinite values as the union of $\mathbf{M}(X, b)$ for all integers b.

JUSTIFICATION: **M** is the normalized angle measurement **angle**$(\vec{N}, \overrightarrow{X_{[\mathrm{A}]}X}, \vec{W})/\theta$ plus the normalized angle $2\pi b/|\theta|$ of b full rotations around A. The normalization ensures that the measure reported by the map is expressed in the proper unit, so that, there exists a value b such that $\mathbf{M}(X_{[\mathrm{A}]} +$ **Rotation**$(\theta, \vec{W}) \circ \vec{N}, b) = 1$, where $X_{[\mathrm{A}]}$ may be any point on A. (See Figure 7.3). The b rotations around A are normalized by the absolute value of θ, because increasing b should increase the result, regardless of the direction of rotation.

We define a **branch** of ID b to be the interval $[(2\pi b - \pi)/|\theta|, (2\pi b + \pi)/|\theta|)$ on the real line. A single value of image $\mathbf{M}(X)$, of any point X, lies in each branch b, for any integer b.

R-EXTENT: The extent $[s, e]$ of B, as a ball of center C and radius r, for a rotation by angle θ around axis A is $\mathbf{E}(\mathrm{B}, b) = [\mathbf{M}(C, b) - h/|\theta|, \mathbf{M}(C, b) + h/|\theta|]$ where $h =$ **sin**$^{-1}(r/|\overrightarrow{CC_{[\mathrm{A}]}}|)$ and b is the desired branch for C to map to. If no b is given, then the extent $\mathbf{E}(\mathrm{B})$ is the union of the infinite intervals corresponding to every possible b.

JUSTIFICATION: Extent $[s, e]$ defines the smallest wedge of space extending infinitely radially from A that contains B. This wedge is the intersection of two linear half-spaces,

each containing A in its bounding plane. Hence, we represent that wedge by the angles of their oriented bounding planes around A, with respect to reference vector $\vec{N}$. Let $c = \mathbf{M}(C, b)$ be the image of center C in branch b. We want to extend the extent in both directions around c to ensure that it covers the image of all points in B. Doing so requires extending in both directions by h, half of the angle (from $C_{[\mathrm{A}]}$) subtending B, where the extension is normalized to the proper unit.

Although c is in branch b, either s might be in branch $b-1$ or e might be in branch $b+1$, because we extended the extent around c, and c may be anywhere in branch b, including its boundary. If s is in branch $b-1$, then we add $2\pi/|\theta|$ to both s and e, to ensure each lies in either branch b or $b+1$.

The image $\mathbf{E}(\mathrm{Q})$ of query ball Q must map to infinite branches, because it may overlap with the image R in any branch. However, the extent of the entire row R can be restricted to lie in only branches 0 and 1, because the total angle sustained by R is less than 2π, so the only portions of $\mathbf{E}(\mathrm{Q})$ that may overlap the image of R are $\boldsymbol{Q}_0 = \mathbf{E}(\mathrm{Q}, 0)$ and $\boldsymbol{Q}_1 = \mathbf{E}(\mathrm{Q}, 1)$. Evaluating RangeFinder with both query-extents, $\boldsymbol{Q}_0$ and $\boldsymbol{Q}_1$, each along with template-extent $[s_0, e_0] = \mathbf{E}(\mathbf{R}_0, 0)$ yields two ranges, only one of which may be non-empty, because a continuous map of the steady row covers less than $2\pi/|\theta|$ on the real line. If one range is non-empty, then it is returned by RangeFinder.

7.1.6 Combining ranges from the two primitive similarities

Assume the candidate ranges, $[a, b]$ returned by RangeFinder for translation or dilation and $[c, d]$ returned by RangeFinder for rotation, have already been computed. The final candidate range, $[\mathbf{max}(a, c), \mathbf{min}(b, d)]$, is the intersection of the two. We have proven that an element of the steady row cannot intersect the query ball if its index is not contained in any one of the candidate ranges. Therefore, we must only consider the indices contained by both ranges as possibly intersecting the query ball, justifying the intersection.

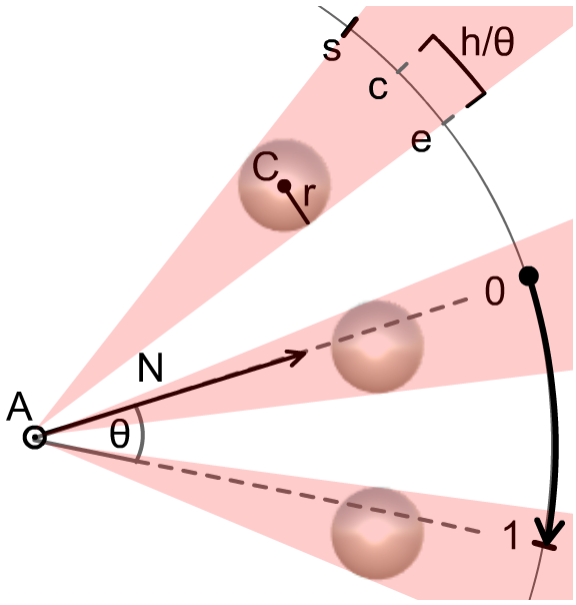


Figure 7.3: A row of balls where **S** is a rotation around A by θ radians. The "real number line" is visualized here as a circle around A. A point in space is mapped onto the real line based on angle around A, such that rotating the point around A by θ radians results in an increase by 1 in its mapped value. The light-red wedges indicate the extent of each ball on the real line.

7.2 Computing the extent of a beam

Here we describe how to compute the extent of a beam defined as the convex hull of two balls, X and Y, with respect to each primitive similarity (translation, dilation, rotation), assuming we have already computed the extents of the balls, $[s_X, e_X]$ and $[s_Y, e_Y]$. The approach proposed here yields a tighter extent than returning the extent of the minimal ball that contains the beam.

T-EXTENT:

$$[s, e] = [\mathbf{min}(s_X, s_Y), \mathbf{max}(e_X, e_Y)].$$

Justification: The most extreme points of the beam along the direction of translation must be in X or Y, so the extent is constructed from only the min and max values of the ball extent. See Figure 7.4.

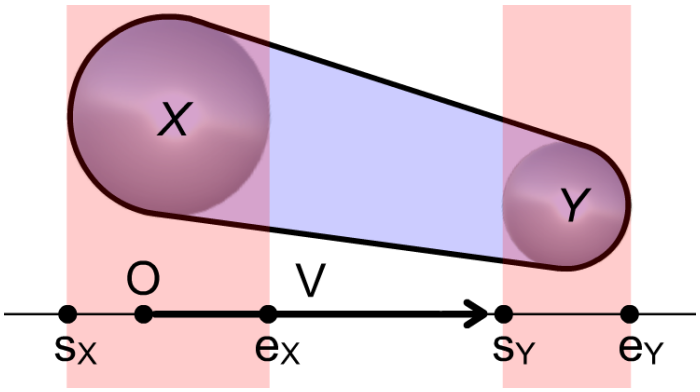


Figure 7.4: Beam extent for a translation.

D-EXTENT:

$$[s, e] = [\mathbf{min}(s_X, s_Y, \mathbf{log}_d(f)), \mathbf{max}(e_X, e_Y, \mathbf{log}_d(f))],$$

where f is the distance from the fixed point F to the interior of the beam.

Justification: The point of the beam farthest from F must be in X or Y. However, the point of the beam closest to F may or may not be in X or Y. Including $\mathbf{log}_d(f)$ accounts for this possibility, and it is used in both min and max to account for dilation factors less than or greater than 1. See Figure 7.5.

Barbier et al. describe an efficient computation of the distance f from a point to a beam [5].

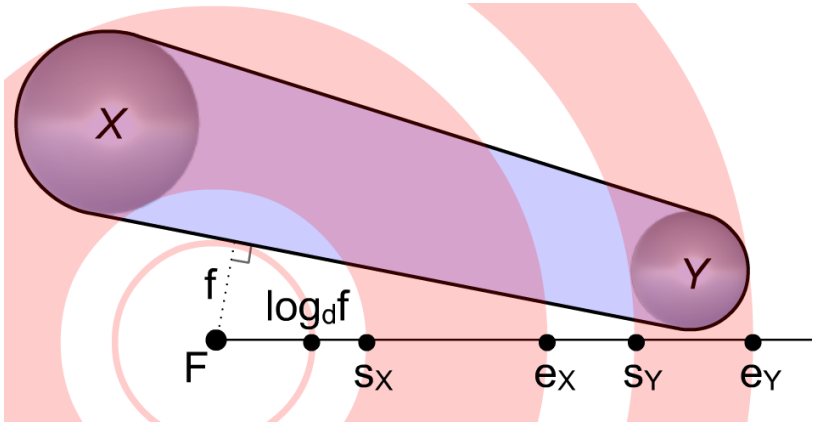


Figure 7.5: Computation of the beam extent for a dilation primitive with dilation factor $d > 1$. F is closer to the conical part of the beam than to the balls.

R-EXTENT: We assume $[s_X, e_X] = \mathbf{E}(\mathrm{X}, 0)$, so the center of X's extent lies in branch 0, though any other branch may be chosen. Let θ be the rotation angle around axis A, $\vec{W}$ be the direction of A, and $h_Y = \mathbf{sin}^{-1}(r_Y/|\overrightarrow{C_Y C_{Y[\mathrm{A}]}}|)$ be half of the angle subtending Y around A. Finally, let $y = (s_X + e_X)/2 + \mathbf{angle}(\overrightarrow{C_{X[\mathrm{A}]}C_X}, \overrightarrow{C_{Y[\mathrm{A}]}C_Y}, \vec{W})/\theta$ be the proper map of C_Y, defined so that its position on the real number line is correct relative to the extent $[s_X, e_X]$. Then,

$$[s, e] = [\mathbf{min}(s_X, y - h_Y/|\theta|), \mathbf{max}(e_X, y + h_Y/|\theta|)].$$

As when computing a ball extent, if A intersects the beam, then we return the full range $[0, n]$.

Justification: Consider starting with the extent $[s_X, e_X]$ of X. The extent $[s_Y, e_Y]$ of Y must be computed relative to s_X and e_X, since computing s_Y and e_Y from Y, in isolation, will not guarantee a map to the correct branch relative to the map of X. The offset from a map of C_X to the proper map of C_Y is $\mathbf{angle}(\overrightarrow{C_{X[\mathrm{A}]}C_X}, \overrightarrow{C_{Y[\mathrm{A}]}C_Y}, \vec{W})/\theta$, which is the angle from C_X to C_Y around A normalized to the proper unit. From the proper map of C_Y, the extent of Y is extended by half of the angle subtending Y from $C_{Y[\mathrm{A}]}$, normalized to the proper unit. Given the extent of X and proper extent of Y, the extent of the beam is computed by combining the two with min and max, as done in the translation case. See Figure 7.6.

7.3 RangeFinder in steady brick lattices

Here we discuss how to use RangeFinder to accelerate a BIQ of query ball Q against a steady lattice, with $\overline{u} \times \overline{v} \times \overline{w}$ node-groups and $\overline{b}$ beam-patterns, to efficiently identify which beams of the lattice intersect Q. We discuss an $\mathbf{O}(\overline{u}\,\overline{v})$ time algorithm for performing BIQs against general steady lattices, then in a subsection, we discuss modifications to the algorithm which may reduce BIQ time-complexity to $\mathbf{O}(\overline{u})$ or $\mathbf{O}(1)$ for special config-

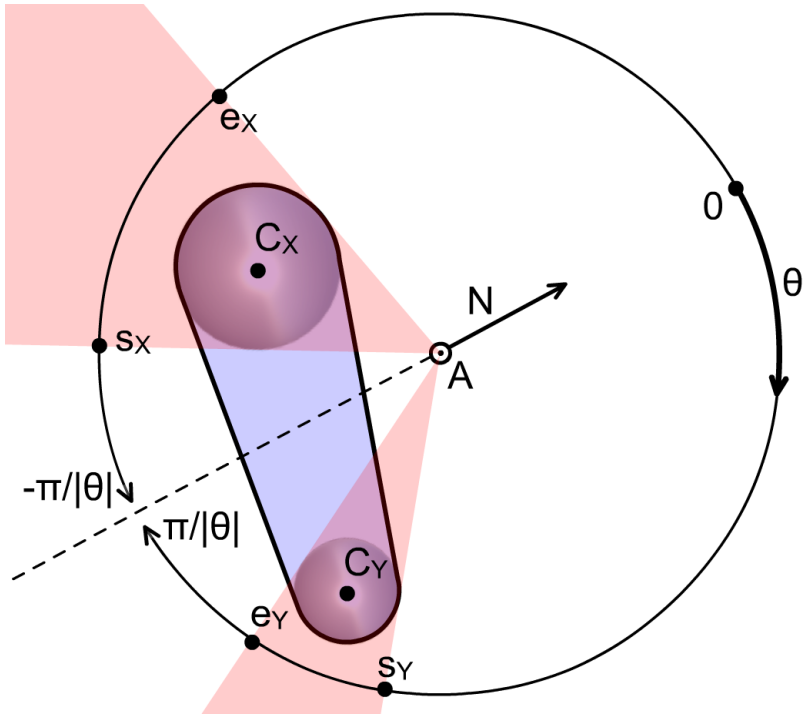


Figure 7.6: Computation of the beam extent for a rotation primitive. Given a map of C_X into branch 0, the correct map of C_Y here is into branch -1.

urations of steady lattices. For simplicity, we process the beam patterns of the lattice one at a time.

First, consider the naïve solution, which performs a ball-beam intersection test between Q and each beam of the lattice. This requires $\mathbf{O}(\overline{u}\,\overline{v}\,\overline{w})$ ball-beam intersection tests.

Using RangeFinder, we reduce the number of ball-beam intersection tests to $\mathbf{O}(uv)$. We do this by first organizing the lattice into a matrix of $\overline{u} \times \overline{v}$ steady rows of beams (see section 3.3), where each row with index (i, j) into the matrix has $\overline{w}$ elements, a template beam originating from node-group $\mathrm{N}[i, j, 0]$, and an incremental similarity **W**. Then, RangeFinder is applied to each of the $\overline{u} \times \overline{v}$ steady rows to identify the (hopefully) small number of beams for which a ball-beam intersection test must be performed.

7.3.1 Special case improvements

For special configurations of a steady lattice, the expected time complexity of a BIQ can be further reduced, from $\mathbf{O}(\overline{u}\,\overline{v})$, to either $\mathbf{O}(\overline{u})$ or $\mathbf{O}(1)$, using simple modifications to the

algorithm above.

2D example

For clarity, we first explain the idea behind our BIQ algorithm modifications, via a 2D example on a $\overline{u} \times \overline{v}$ steady lattice. On such a lattice, we can perform a BIQ in $\mathbf{O}(\overline{u})$ time with the general RangeFinder algorithm. So, here, we want to describe a special case in which $\mathbf{O}(1)$ BIQs are possible.

Consider the special 2D lattice in Figure 7.8. Its **U** is a clockwise rotation around F and its **V** is a dilation towards F. One special property of this lattice is that every beam belongs to two steady rows: one with incremental similarity **U** and one with incremental similarity **V**. For an example when this is not the case, consider the cyan beams along the **U** direction in Figure 7.7. See section 3.3 for a discussion on the steadiness of beams.

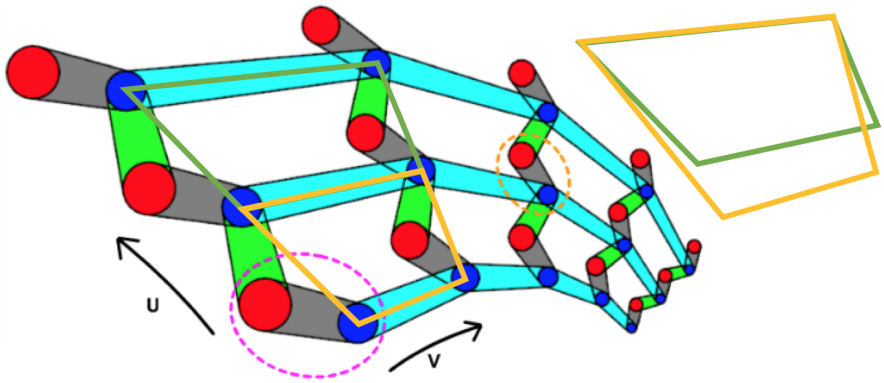


Figure 7.7: We show an orange quad defined by the first and second cyan beams and a green quad defined by the second and third cyan beams. On the right, we use a similarity transform to scale and align the two quads so that their top edges match. The result demonstrates that the pattern of these three cyan beams is not steady.

Consider the four rotation-based rows, with incremental similarity **U**, of which the template shape is either a magenta beam or the red beam. Notice that the rotation extents for each of these template shapes are the same. Each of these rows has the same incremental similarity, the same number of elements, and the same template-extents, so performing a RangeFinder, with the same query ball, on any of the four rows, will always give the same

result. Therefore, we perform one RangeFinder, on any of these rows, to compute a range of i-indices that identify a set of candidate beams in all four rows. In this example, the range of i-indices is $[1, 1]$.

Similarly, consider the five dilation-based rows, with incremental similarity **V**, of which the template shape is either a blue or the red beam. Notice that the dilation extents for each of these template shapes are the same. As before, performing a RangeFinder, with the same query ball, on any of the five rows will give the same result. We perform RangeFinder on the first row to compute a range of j-indices that identify a set of candidate beams in all five rows. In this example, the range of j-indices is $[1, 2]$.

Finally, we combine the i-indices and the j-indices to determine which beams we must test for ball-beam interference. In this example, we must test the beams originating from ball-groups $(1, 1)$ and $(1, 2)$.

O(u) BIQ case

We use the idea from the 2D example, in subsubsection 7.3.1, to create our algorithm for performing $\mathbf{O}(\overline{u})$ BIQs on 3D lattices. Then, we provide conditions specifying for which lattice configurations the algorithm is valid.

To perform an $\mathbf{O}(\overline{u})$ BIQ, we iterate over all $i \in [0, \overline{u})$, and for each i, we consider a $\overline{v} \times \overline{w}$ sub-lattice, with incremental similarities **V** and **W**. Node-group $\mathrm{N}[0, 0]$ of the sub-lattice is the node-group $\mathrm{N}[i, 0, 0]$ of the original lattice. We apply the idea from the 2D example, to perform an $\mathbf{O}(1)$ BIQ on the sub-lattice, by doing the following:

1. Compute a range ***A*** of j-indices using one RangeFinder on a steady row with $\overline{v}$ elements, incremental similarity **V**, and template shape as the beam originating from node-group $\mathrm{N}[0, 0]$.

2. Compute a range ***B*** of k-indices using one RangeFinder on a steady row with $\overline{w}$ elements, incremental similarity **W**, and template shape as the beam originating from node-group $\mathrm{N}[0, 0]$.

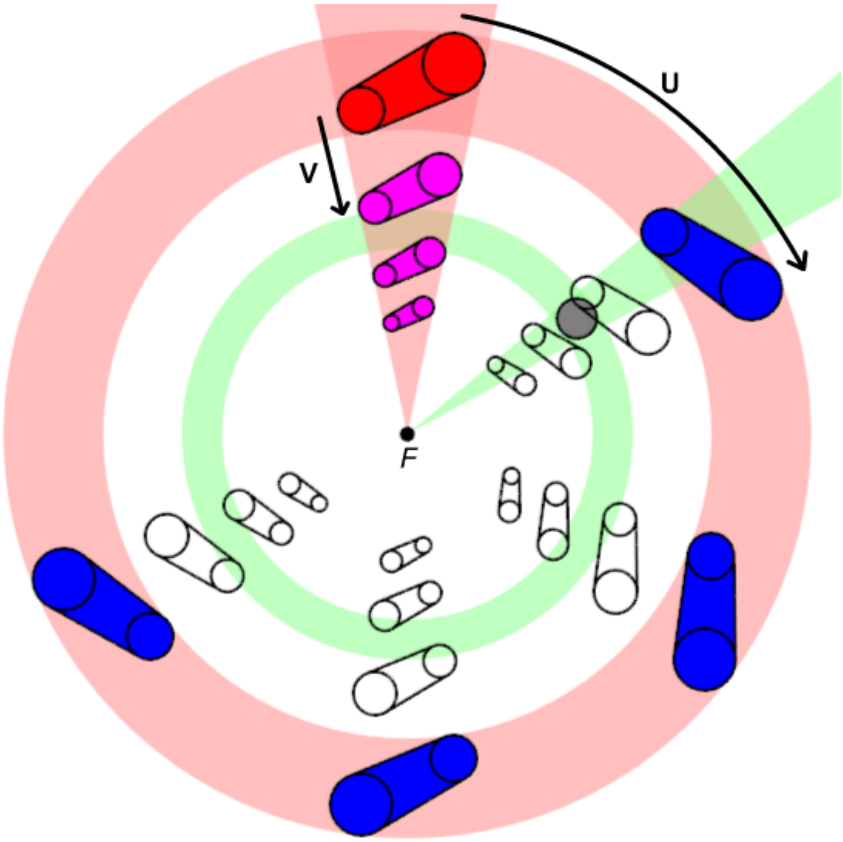


Figure 7.8: A 2D example of a 5×4 steady lattice on which we can perform an $\mathbf{O}(1)$ BIQ. The beam originating from ball-group $(0, 0)$ is shown in red. **U** is a clockwise rotation around F. **V** is a dilation towards F. The rotation extents for each magenta beam are the same, as indicated by the light-red wedge. The dilation extents for each blue beam are the same, as indicated by the light-red annulus. A query disk is shown in grey, and its extents are indicated by a light-green wedge and a light-green annulus.

3. Perform a ball-beam intersection test for all beams originating from node-groups $\mathrm{N}[j, k]$, for all $j \in \boldsymbol{A}$ and for all $k \in \boldsymbol{B}$.

The above algorithm is valid if each sub-lattice meets the following conditions:

1. Each beam belongs to a steady row with incremental similarity **V**. Tested by verifying that node-group $\mathrm{N}[1, k] = \mathbf{V} \circ \mathrm{N}[0, k]$, for any $k \neq 0$.

2. The extents, with respect to **V**, of the beams originating from all node-groups $\mathrm{N}[0, k]$, for all k, are the same. I.e., transforming a beam by **W** does not change its extents with respect to **V**.

3. The extents, with respect to **W**, of the beams originating from all node-groups $\mathrm{N}[j, 0]$, for all j, are the same. I.e., transforming a beam by **V** does not change its extents with respect to **W**.

O(1) BIQ case

To perform an $\mathbf{O}(1)$ BIQ on a valid lattice, we extend the idea from the 2D example in section subsubsection 7.3.1, by doing the following:

1. Compute a range ***A*** of i-indices using one RangeFinder on a steady row with $\overline{u}$ elements, incremental similarity **U**, and template shape as the beam originating from node-group $\mathrm{N}[0, 0, 0]$.

2. Compute a range ***B*** of j-indices using one RangeFinder on a steady row with $\overline{v}$ elements, incremental similarity **V**, and template shape as the beam originating from node-group $\mathrm{N}[0, 0, 0]$.

3. Compute a range ***C*** of k-indices using one RangeFinder on a steady row with $\overline{w}$ elements, incremental similarity **W**, and template shape as the beam originating from node-group $\mathrm{N}[0, 0, 0]$.

4. Perform a ball-beam intersection test for all beams originating from node-groups $\mathrm{N}[i, j, k]$, for all $i \in \boldsymbol{A}$, for all $j \in \boldsymbol{B}$, and for all $k \in \boldsymbol{C}$.

The above algorithm is valid if the lattice meets the following conditions:

1. Each beam belongs to a steady row with incremental similarity **U**. Tested by verifying that node-group $\mathrm{N}[1, j, k] = \mathbf{U} \circ \mathrm{N}[0, j, k]$, for any $j \neq 0$ and for any $k \neq 0$.

2. Each beam belongs to a steady row with incremental similarity **V**. Tested by verifying that node-group $\mathrm{N}[i, 1, k] = \mathbf{V} \circ \mathrm{N}[i, 0, k]$, for any i and for any $k \neq 0$.

3. Transforming a beam by **V** or **W** does not change its extents, with respect to **U**.

4. Transforming a beam by **U** or **W** does not change its extents, with respect to **V**.

5. Transforming a beam by **U** or **V** does not change its extents, with respect to **W**.

7.4 Results

In this section, we demonstrate how RangeFinder improves the performance of BIQs against steady rows, compared to naïve BIQs. We do this first by demonstrating time improvements in identifying the beams of fine, steady lattices that belong to two-level LiLs (presented in chapter 9), and then we show results for tests of 100 random BIQs against steady lattices. Although the LiL concept is presented in chapter 9 instead of in this chapter, we have chosen to present the LiL acceleration tests here, because these tests are more relavant to RangeFinder than they are to LiL.

Table 7.1 shows the time taken to compute the sets of beams, to be displayed, for the LiLs shown in Figure 7.9. The numbers include only the time to identify the beams to be displayed and do not include actual rendering time. Each LiL was generated on 8 threads, where the parameters that implicitly describe the lattices were available to each thread. Each thread was assigned an approximately equal number of node-groups from the fine lattice, divided along the **U** direction. For each valid beam B originating from the assigned groups, its assigned thread analyzed B for membership in the final LiL by performing two BIQs, one for both balls of B, against the coarse lattice. Each beam identified to be in the LiL is added to a list of beams to be rendered, each as a triangle mesh, and each beam in the list is represented by the two (i, j, k, b) four-tuples representing its incident nodes. Storing the identified beams like this does not prohibit rendering lattices with more beams than can be stored in memory, because smaller chunks of the lattice can be rendered individually and later composed, without ever needing to store a fully evaluated list of beams.

The results clearly show a benefit of RangeFinder accelerated BIQs over naïve BIQs. However, the benefit of $\mathbf{O}(\overline{u}\,\overline{v})$ RangeFinder BIQs over naïve $\mathbf{O}(\overline{u}\,\overline{v}\,\overline{w})$ BIQs may not be as dramatic as expected. The reason for this is that the BIQs were performed against coarse

lattices with no more than 9 groups in the **W** direction, which is very small. Therefore, we also performed BIQ tests, without the LiL concept, on steady lattices with more groups.

For three different fine lattices, on a single thread, we performed 100 BIQs with query balls of which the centers were randomly placed in the axis-aligned bounding box of the lattice and of which the radii were randomly chosen to be between 90%-110% of the average radius of the lattice's balls. Table 7.2 shows the times it took to perform the 100 BIQs. The lattices labeled "Regular" and "Cylindrical" are the same as the fine lattices used to produce the LiLs in Figure 7.9. The other lattice is shown in Figure 7.10. These tests demonstrate an *improvement between 45.4× and 9420×* over naïve BIQs, though improvement varies heavily based on the complexity of a lattice.

These tests were performed on a machine with an i7-7700K@4.20GHz and 32GB DDR4 RAM and were developed in Java within the Processing framework.

	Regular	**Cylindrical**	**Warped**
Naïve BIQ	427 s	1480 s	869 s
$\mathbf{O}(\overline{u}\,\overline{v})$ BIQ	95.5 s	239 s	266 s
$\mathbf{O}(\overline{u})$ BIQ	13.7 s	22.3 s	-
$\mathbf{O}(1)$ BIQ	3.44 s	-	-

Table 7.1: Time taken to identify each beam to be displayed in the LiLs shown in Figure 7.9, for naïve, $\mathbf{O}(\overline{u}\,\overline{v})$, $\mathbf{O}(\overline{u})$, and $\mathbf{O}(1)$ BIQs. A dash result means the BIQ complexity cannot be achieved on that lattice, because the lattice does not meet the conditions described in section subsection 7.3.1

	Regular	**Cylindrical**	**Figure 7.10**
Naïve BIQ	55.2 s	259 s	167 s
$\mathbf{O}(\overline{u}\,\overline{v})$ BIQ	1.60 s	2.72 s	3.68 s
$\mathbf{O}(\overline{u})$ BIQ	0.0558 s	0.0554 s	-
$\mathbf{O}(1)$ BIQ	0.00586 s	-	-

Table 7.2: Time taken perform 100 BIQs on the fine lattices used to generate the LiLs in Figure 7.9 plus the fine lattice in Figure 7.10. A dash means the BIQ complexity cannot be achieved on that lattice.

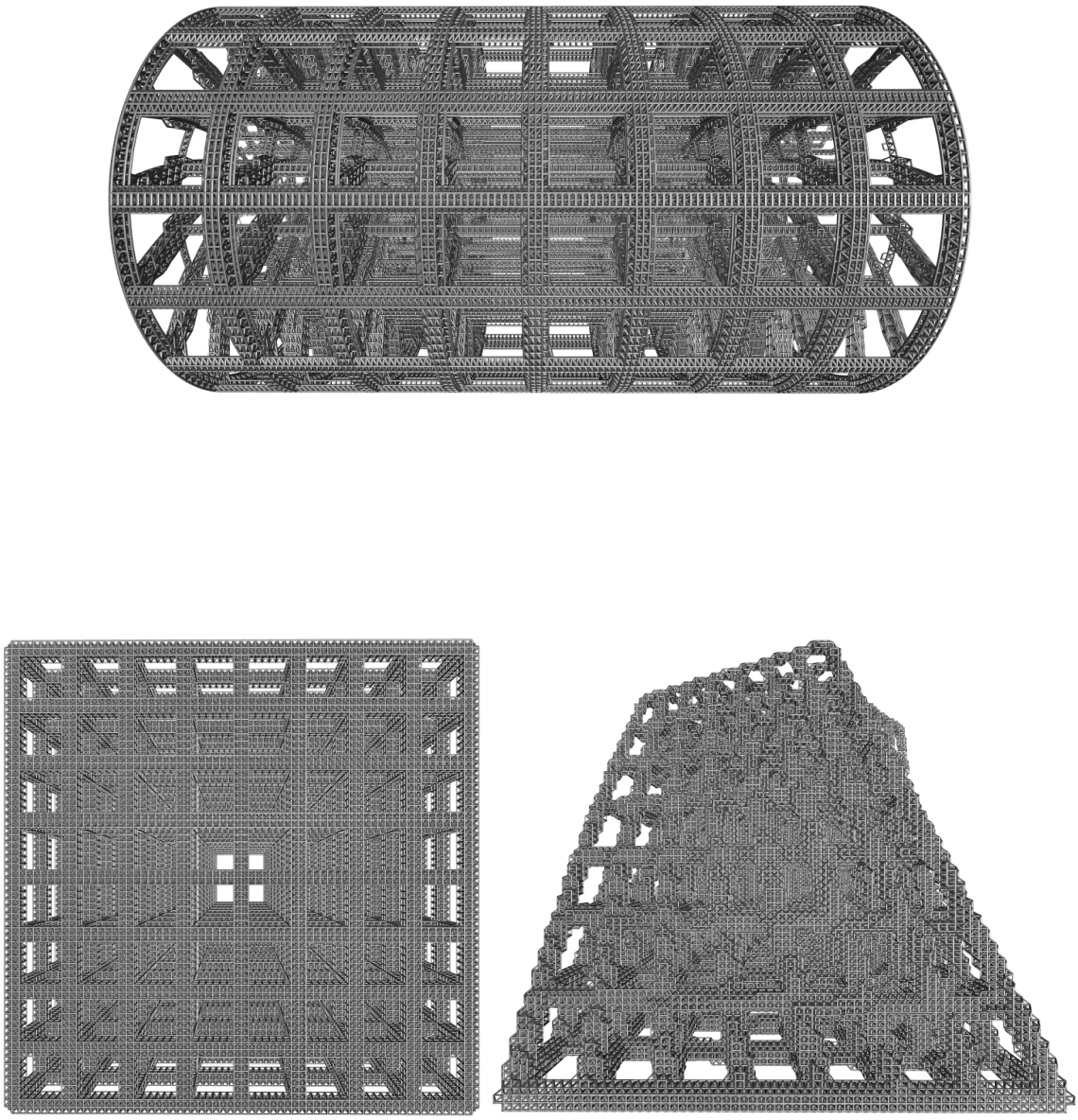


Figure 7.9: (Top) LiL created from a cylindrical, steady fine lattice of $32 \times 128 \times 128$ groups and a cylindrical, steady coarse lattice lattice of $5 \times 17 \times 9$ groups. (Bottom-left) LiL created from a regular, fine lattice of 75^3 groups and a regular, coarse lattice of 9^3 groups. (Bottom-right) LiL created from a regular, fine lattice of 75^3 groups and a semi-regular coarse lattice of 8^3 groups. All three examples have a fine lattice of 2 balls per group and 14 edge-patterns and have a coarse lattice of 1 ball per group and 3 edge patterns.

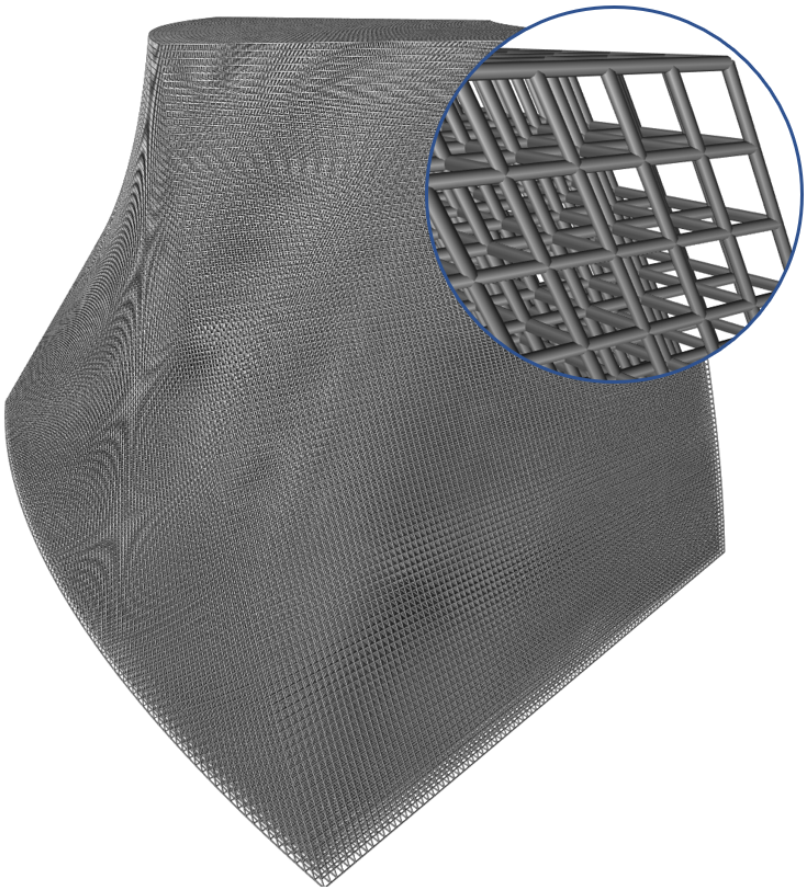


Figure 7.10: Steadily bent, twisted, and tapered semi-regular lattice with 100^3 groups of a single ball and 3 edge-patterns. The top-right is magnified to show detail. This image is the same as Figure 1.1, but we also include it here for convenience.

CHAPTER 8

BECOTS RANGEFINDER

In this chapter, we address the problem of performing Ball Interference Queries (BIQs) on BeCOTS patterns. That is, given a BeCOTS pattern P of shapes and a query ball Q, we want to quickly determine the subset of shapes of P that interferes (has a non-empty intersection) with Q. The shapes of the BeCOTS pattern are defined as $\mathrm{P}[i,j] = \mathbf{V}^j \circ \mathbf{U}^i \circ \mathrm{P}[0,0]$, where $\mathbf{U}$ and $\mathbf{V}$ are commutative similarities, $\mathrm{P}[0,0]$ is the template shape, and $i \in [0, \overline{u})$ and $j \in [0, \overline{v})$ are integer indices. We call this version of BIQ the **BeCOTS BIQ**. A BeCOTS BIQ is a generalization of the steady 1-pattern BIQ discussed in chapter 7. Our RangeFinder $\mathbf{O}(1)$ time solution for a steady 1-pattern BIQ enables us to perform a steady 2-pattern BIQ in $\mathbf{O}(\overline{u})$ time, assuming that the pattern's shapes are well separated and that the query ball is not too large. A BeCOTS pattern is a special case of a steady 2-pattern, so the $\mathbf{O}(\overline{u})$ solution may be used for BeCOTS BIQs.

Here, we introduce a new solution that enables us to solve the BeCOTS BIQ problem in constant-time. Our new solution is in the same spirit as the RangeFinder solution for steady 1-pattern BIQs, but this solution is entirely new. The essence is to (in constant-time) identify a small but conservative set R of (i,j) index pairs that identify all shapes $\mathrm{P}[i,j]$ that could possibly interfere with the query ball Q. R is created from two integer intervals $\boldsymbol{I}$ and $\boldsymbol{J}$ that each identify a range of relevant i and j indices such that R is composed of all possible combinations of indices from $\boldsymbol{I}$ and $\boldsymbol{J}$. The set of shapes of P that interfere with Q are a subset of the shapes identified by the index pairs in R, so we test each shape identified by R for interference with Q, and we return the positive results as the output of a BeCOTS BIQ. We call this new solution the **BeCOTS RangeFinder**, because the essence of the solution is finding ranges of indices, and the BeCOTS RangeFinder solves a more general problem than the original RangeFinder described in chapter 7.

This chapter has two parts. First, we describe how to compute a small but conservative set R of index pairs that identify all shapes of a planar COTS pattern that could possibly interfere with a query disk. Then, we describe how a 3D BeCOTS pattern can be transformed into a planar COTS pattern and how a query ball can be transformed into a planar query shape (not quite a disk), such that the output set R of a planar COTS RangeFinder is the output of the BeCOTS RangeFinder.

8.1 Planar COTS RangeFinder

A key feature of COTS maps is that they map translations to similarities [35]. The inverse $\mathbf{M}^{-1}$ of a COTS map $\mathbf{M}$ maps some similarities into translations, and, in particular, the $\mathbf{U}$ and $\mathbf{V}$ similarities, between consecutive shapes along their respective directions in a COTS pattern, are mapped into translations by the vectors $<1,0> = \mathbf{M}^{-1} \circ \mathbf{U}$ and $<0,1> = \mathbf{M}^{-1} \circ \mathbf{V}$.

The key idea behind the planar COTS RangeFinder is that the pre-image of a COTS pattern P is a regular 2-pattern P', where the pre-image shapes are defined as $\mathrm{P}'[i,j] = i<1,0> + j<0,1> + \mathrm{P}'[0,0]$. And, it is possible to compute in constant-time the ranges $\boldsymbol{I}$ and $\boldsymbol{J}$ of indices i and j that identify all shapes of P' that could possibly interfere with the pre-image $\mathrm{Q}' = \mathbf{M}^{-1} \circ \mathrm{Q}$ of the query disk Q.

The exact pre-images $\mathrm{P}'[0,0]$ and Q' are not easy to compute. However, we do not need to compute the exact pre-images. We only need to compute a decent axis aligned bounding box (AABB) for both $\mathrm{P}'[0,0]$ and Q'. To compute these AABBs, we take advantage of the fact that a COTS map is a composition of an affine transformation and log-polar transformation [34], so the inverse of a COTS map is the composition of the inverse of a log-polar transformation with the inverse of an affine transformation. The inverse of a log-polar transformation maps a set of lines through fixed point F (of the similarities $\mathbf{U}$ and $\mathbf{V}$) into parallel lines, and it maps a set of circles centered at F into parallel lines. And, the inverse of an affine transformation preserves both lines and parallelism. We use the term

log-polar bound to refer to the area bounded by two lines through F and by two circular arcs centered at F. Therefore, if we tightly bound each shape $\mathrm{P}[0,0]$ and Q by a log-polar bound, then the pre-images of the log-polar bounds are parallelogram bounds of $\mathrm{P}'[0,0]$ and Q'. The four corners of the parallelogram bounds are the pre-images of the four corners of the log-polar bounds, so the parallelogram bounds may be computed by inverse mapping each corner of the log-polar bound. Let $\mathrm{B}'[0,0]$ be the parallelogram bound of $\mathrm{P}'[0,0]$. The parallelogram bound of each other pre-image shape $\mathrm{P}'[i,j]$ is $\mathrm{B}'[i,j] = <i,j> + \mathrm{B}'[0,0]$. The AABB of a parallelogram is trivial to compute. Figure 8.1 shows a COTS pattern and a query disk along with their pre-images. The log-polar and parallelogram bounds are also shown.

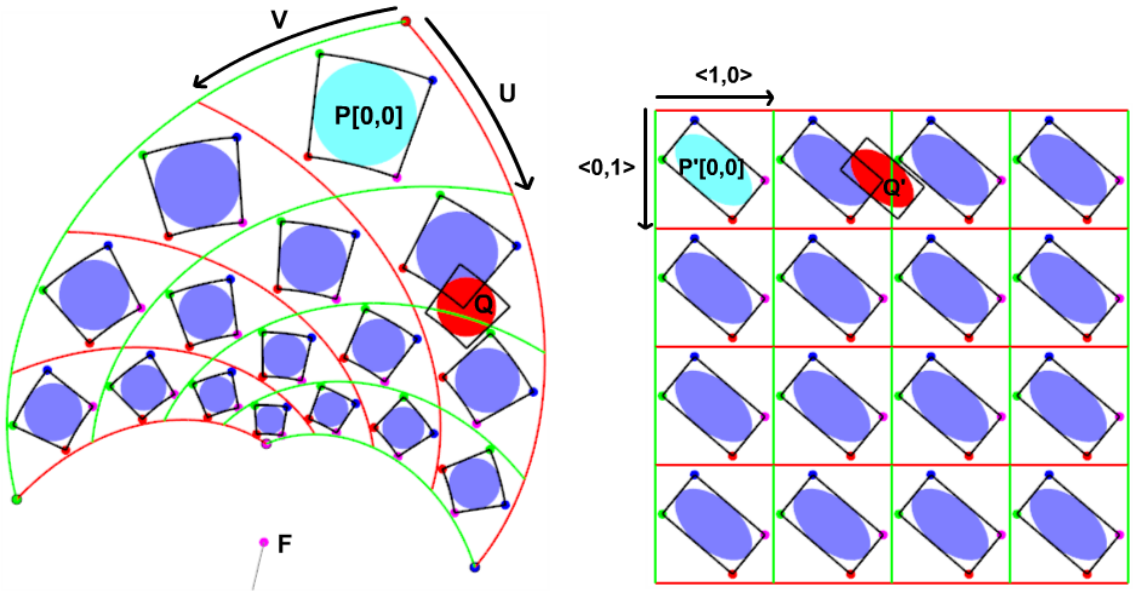


Figure 8.1: (Left) A COTS pattern of disks shown in blue. The template disk $\mathrm{P}[0,0]$ is shown in cyan, and the query disk Q is shown in red. The log-polar bounds are shown for each disk as a black outline. (Right) The pre-images P' and Q'. P' is a regular 2-pattern, and the log-polar bounds from the left have become parallelogram bounds.

The range $\boldsymbol{I}$ of i indices can be obtained using the RangeFinder for translation (described in subsection 7.1.3) on the translational pattern of parallelogram bounds. This yields $\boldsymbol{I} = [\mathbf{max}(0, \lceil c - b \rceil), \mathbf{min}(\overline{u}, \lfloor d - a \rfloor)]$, where $[a,b]$ represents the min and max x-coordinates of parallelogram that bounds $\mathrm{P}'[0,0]$ and $[c,d]$ represents the min and max

x-coordinates of the parallelogram the bounds Q'. The same process may be used to compute $\boldsymbol{J} = [\mathbf{max}(0, \lceil y - x \rceil), \mathbf{min}(\overline{v}, \lfloor z - w \rfloor)]$, where $[w, x]$ represents the min and max y-coordinates of parallelogram that bounds $\mathrm{P}'[0, 0]$ and $[y, z]$ represents the min and max y-coordinates of the parallelogram the bounds Q'.

Note that care must be taken with regards to choosing an appropriate branching option k when computing the inverse mappings of the log-polar bound corners. In particular, if we were to trace a path along the boundary of the log-polar bound, then we may cross into a different branch of the COTS map while tracing from one corner to another, and the branching option k would need to be incremented or decremented to appropriately represent the change of branch. Also, if the COTS pattern P has shapes in multiple different branches of the COTS map, then the log-polar bound on the query disk Q must be inverse mapped into each different branch, and a different set R of index pairs must be generated for each branch to ensure all interferences between the query disk and the COTS pattern shapes are detected.

8.2 Implementing BeCOTS RangeFinder with planar COTS RangeFinder

Here we discuss how a BeCOTS pattern may be transformed into planar COTS pattern so that the process described in section 8.1 can be used to compute the set R of index pairs. The key idea here is what we call the **umbrella projection** that takes a 3D BeCOTS pattern to a planar COTS pattern. An umbrella projection maps a 3D point P onto the plane Π through point F with normal $\vec{T}$. The umbrella projection moves P onto Π by rotating around F, in the plane that contains F, $F + \vec{T}$, and P. The smallest angle that takes P onto Π is used in the rotation. We call this mapping the umbrella projection because its action on a cone with apex F is like the opening of an umbrella. For an umbrella projection of a BeCOTS pattern, F is the fixed point (of the similarities **U** and **V**) and $\vec{T}$ is the direction of the axis of rotation (of the similarities). An umbrella projection preserves rotations between shapes around the axis through F with direction $\vec{T}$ and preserves dilations between shapes

about point F, so a COTS pattern is obtained from an umbrella projection of a BeCOTS pattern with fixed point F and with an axis of rotation with direction $\vec{T}$. An example umbrella projection of a BeCOTS map and lattice is shown in Figure 8.2. We compute a COTS pattern as the umbrella projection of a BeCOTS pattern, and we compute a query shape as the umbrella projection of the query ball Q.

The exact umbrella projection of a shape may not be easy to compute. For example, the umbrella projection of a ball is not a disk. However, our planar COTS RangeFinder was formulated in terms of log-polar bounds, so, to reuse the planar COTS RangeFinder here, we only need to compute log-polar bounds for the umbrella projections of $\mathrm{P}[0,0]$ and Q. The two lines in each log-polar bound are the intersection of Π with the planes through the BeCOTS rotation axis that are tangent to either $\mathrm{P}[0,0]$ or Q. And, the two circular arcs in each log-polar bound are the intersection of Π with the spheres centered at F that are tangent to either $\mathrm{P}[0,0]$ and Q.

Note, we assume the query ball Q does not interfere with the axis of rotation of the BeCOTS pattern. Also, if the query ball Q does not interfere with the conical shell that bounds the BeCOTS pattern (see section 5.5) then we can safely let R be an empty set.

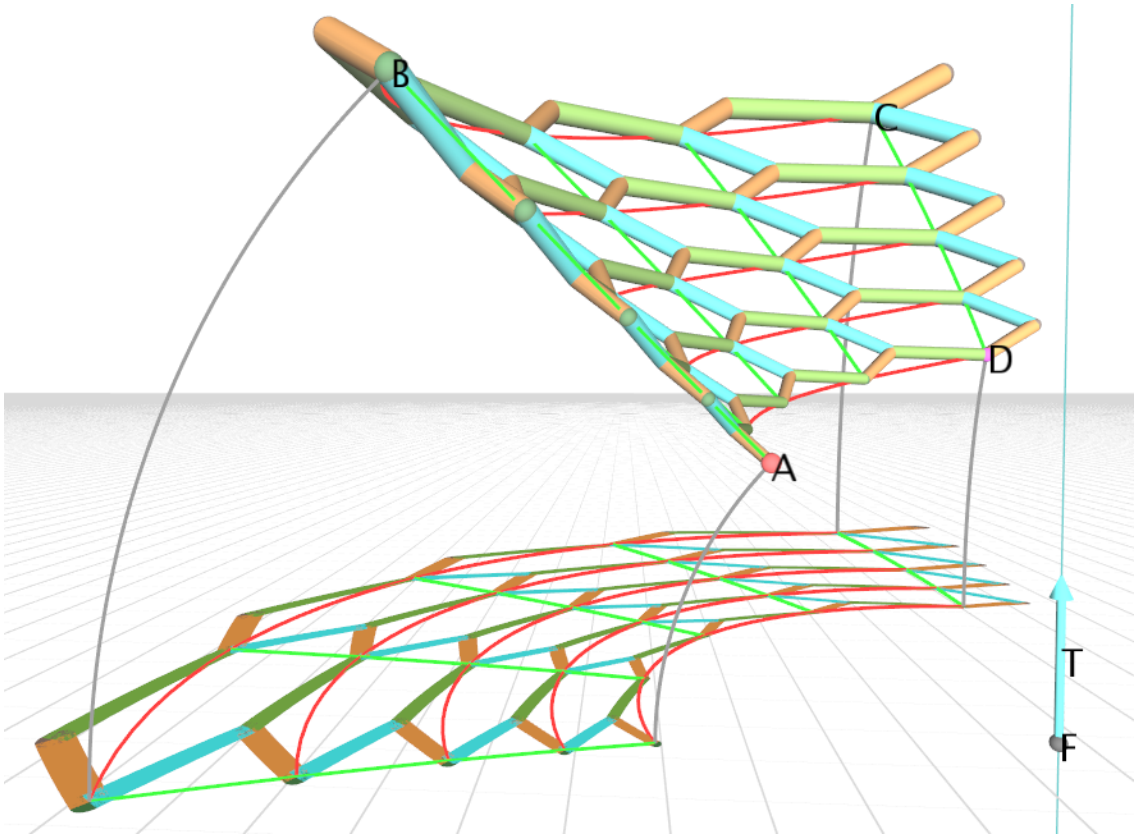


Figure 8.2: A BeCOTS map and lattice and their umbrella projections. The umbrella projection moves the four corners of the BeCOTS map along the grey circular arcs and onto the plane.

CHAPTER 9

GEOMETRY FILTERS FOR HIERARCHICAL LATTICES

A multi-level lattice (also called hierarchical lattice) is a lattice for which each beam is itself a lattice composed of smaller beams. It has been shown that multi-level lattices may be used to increase recoverability and resistance to failure from compression [28]. Multi-level lattices may exhibit higher strength and stiffness, for a given weight, when compared to existing single-level lattices [28]. Multi-level lattices have also been used to design scaffolds on which tissue such as bone can be grown, where the multi-level structure increases porosity to enhance nutrient transport [9]. See [24] for a more general discussion of modeling and querying multi-level structures. The complexity of such hierarchical structures far exceeds the modeling capabilities of legacy commercial CAD systems.

A **filtered lattice**, is a lattice for which some of the beams have been filtered (procedurally removed). The beams are filtered by iterating through each beam of a lattice and testing both nodes N_1 and N_2 of each beam with a **filtering function F**(N) that takes a node N as input and returns either true or false. If the filtering function returns false for one or both nodes of a beam, then the beam is removed. If a lattice's beams are filtered carefully, the result may be a hierarchical lattice for which the remaining beams appear to belong to a larger beam. We present two types of filtering functions, index filters and Lattice-in-Lattice (LiL) filters, for modeling hierarchical lattices.

An **index filtering** function **F**(N) filters a node based on the node-group indices (i, j, k) and the node ID n of the node N. Many different types of index filtering strategies may exist for different use cases. One possible type of index filter is an **index stencil filter**. The idea of the index stencil filter is to define a $\overline{x} \times \overline{y} \times \overline{z}$ array S of Boolean values called the stencil where the value with indices (x, y, z) is accessed as $S[x, y, z]$. To determine if a node N with node-group indices (i, j, k) is to be removed, we map the indices into the

stencil such that $\mathbf{F}(\mathrm{N})$ returns the value of $S[i \bmod \overline{x}, j \bmod \overline{y}, k \bmod \overline{z}]$. Figure 9.1 shows an example hierarchical lattice constructed with an index stencil filter.

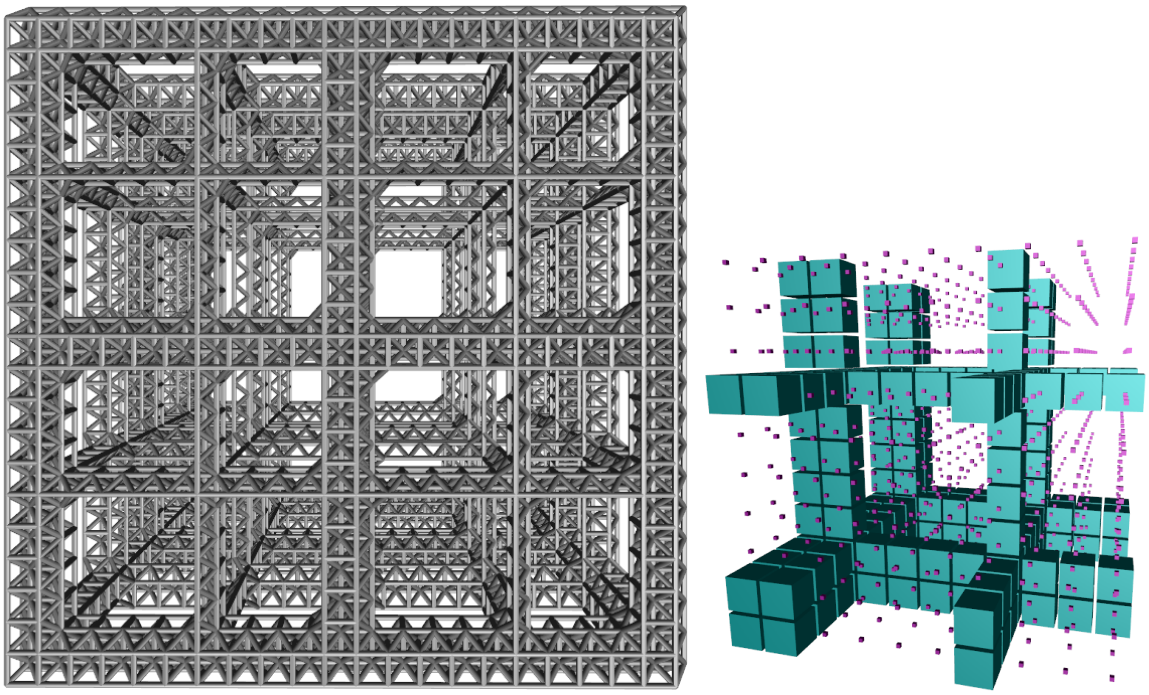


Figure 9.1: (Left) A lattice where some beams have been removed using an index stencil filter. (Right) A voxel representation of the stencil used to filter the lattice on the left. The cyan cubes represent node-groups that remain and the small magenta cubes represent node-groups that are removed.

For the **Lattice-in-Lattice (LiL)** filtering function, a node N of the fine lattice L is filtered out if N does not interfere with a given coarse lattice C. In other words, N is removed if a BIQ against C returns empty for query ball N. The RangeFinder algorithm, presented in chapter 7, can be used to accelerate BIQs against steady lattices, which can improve the time complexity of LiL filters from $\mathbf{O}(\overline{u}\,\overline{v}\,\overline{w})$ to $\mathbf{O}(\overline{u}\,\overline{v})$, or to $\mathbf{O}(\overline{u})$ or $\mathbf{O}(1)$ in special cases. Figure 9.2 demonstrates the trimming of a fine lattice against a coarse lattice. Figure 7.9 shows three different example LiLs.

Index filtering tends to be an efficient and easy to implement method for generating some types of hierarchical lattices. However, an index filtered lattice tends to look like a LiL for which the coarse lattice has the same global shape as the fine lattice, so it would not be a simple task to design the lattice in Figure 7.9-Bottom-Right using an index filter.

LiL filters tend to be a better option than index filters for designing hierarchical lattices that look like a coarse lattice that does not have the same global shape as a fine lattice.

It is possible to create compound filters that combine multiple filters into a more complex filter. For example, a compound filter may use two or more LiL filters to model a lattice with beams made of beams made of beams, as demonstrated in Figure 9.3. It is also possible to use both an index filter and a LiL filter together in a compound filter.

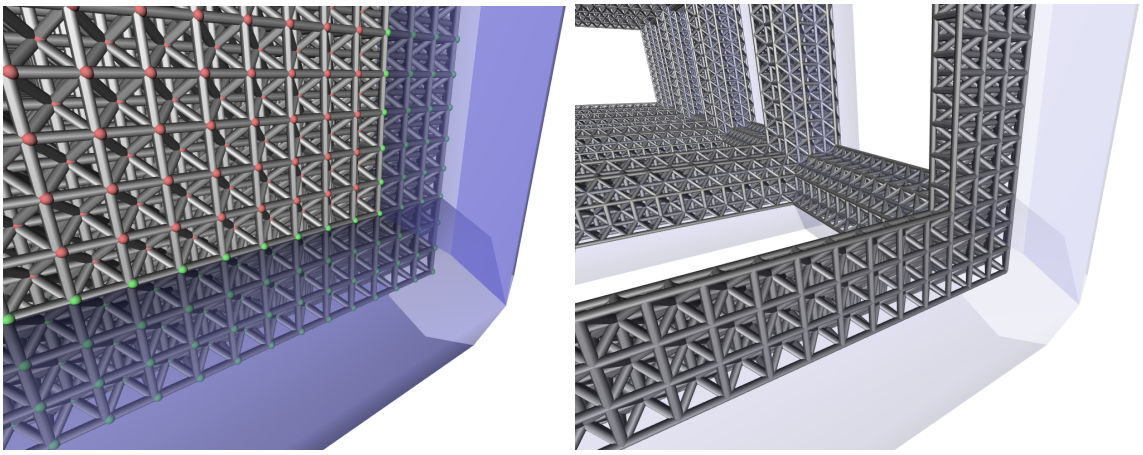


Figure 9.2: (Left) Coarse lattice C shown in transparent blue and a fine lattice F of which its balls are colored green if they interfere with C and red if they do not. (Right) The LiL resulting from removing from F all beams with at least one red ball.

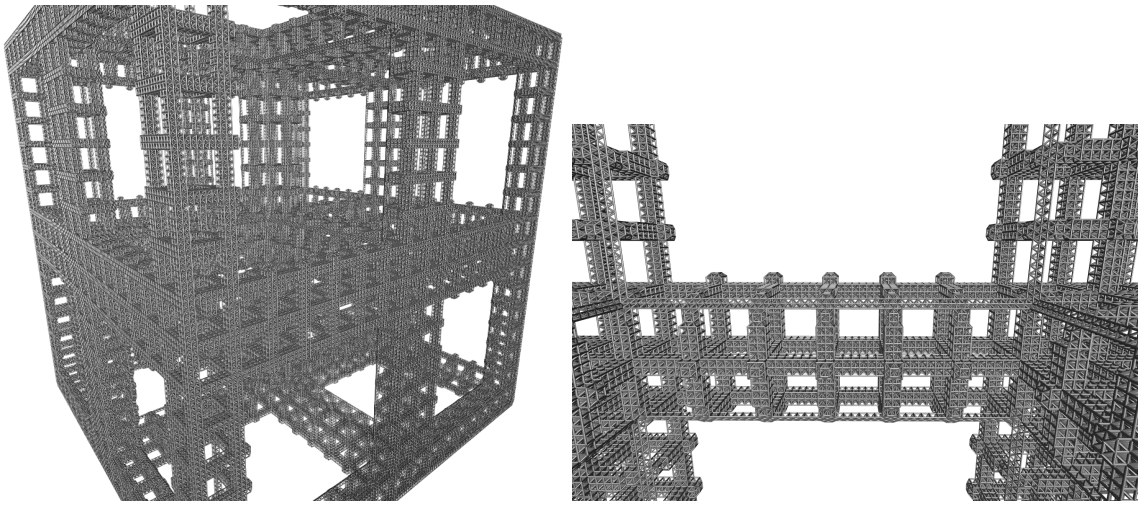


Figure 9.3: (Left) Compound LiL produced from two coarse lattices and a fine lattice with 138^3 groups. The resulting structure has 1,494,074 beams. (Right) Magnification on a multi-level beam.

The filters presented in this chapter have been for the goal of producing beam-shaped hierarchical structures. However, there is no need to restrict the filters such that they produce beam-shaped coarse structures. For example, a filter may be designed to produce coarse gyroid structures (see Figure 9.4). Gyroids have been used in the design of periodic structures for additive manufacturing [21].

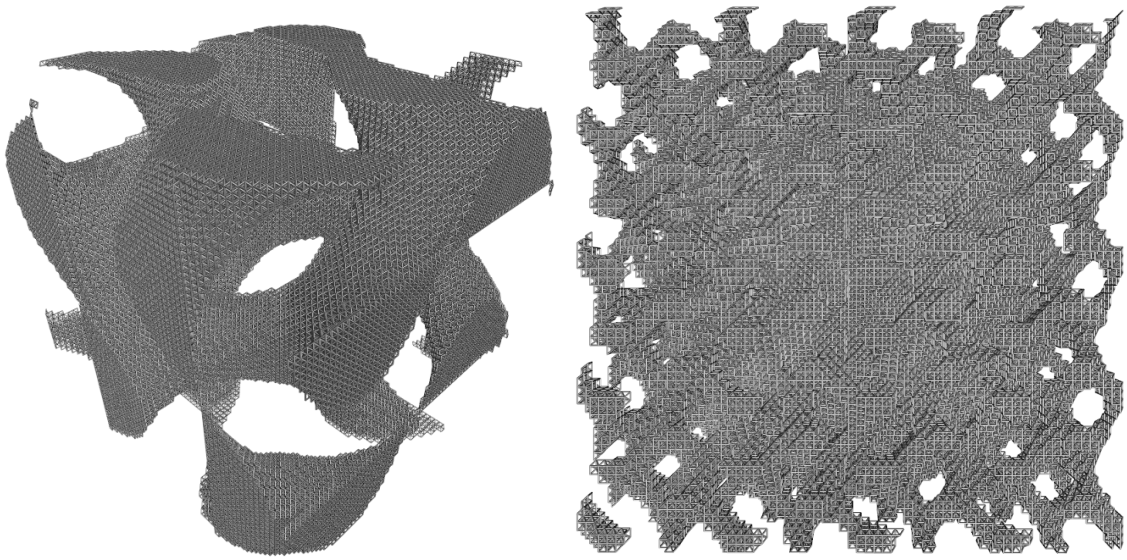


Figure 9.4: Two lattices filtered by a gyroid filter.

CHAPTER 10
RECURSIVE CLG STRUCTURES

In chapter 9, we discussed hierarchical lattices for which each beam may itself be made up of many smaller beams. In this chapter, we are interested in more general hierarchical structures that are not a union of balls and beams. Here, we discuss an extension of Constructive Lattice Geometry (CLG), which we introduced in chapter 5, to support recursively defined structures, in which a CLG may be made up of many small CLG structures. The interactive design of recursive and periodic solid models have previously been explored in [44], and recursion of regular, periodic function-representation (FRep) models has been used for modeling hierarchical microstructures [12].

10.1 Definition

A recursive CLG is an extension to the non-recursive CLG structures defined in chapter 5. As such, the definition of recursive CLG reuses all pieces of the original non-recursive definition.

However, here we generalize the definition of a primitive to make it more powerful. This modification is to assign a material ID to each primitive in the template primitive-group $\mathrm{B}[0,0]$, where a primitive is either a ball or a cone beam. All repeated instances of a primitive have the same material ID.

A **material ID** may simply be an integer that is used to refer to an **abstract material**. An abstract material may be used by a CLG designer for any purpose. For example, we use abstract materials to assign colors to primitives. For recursive CLG, we also allow the abstract material to define the (possibly) recursive nature of the primitive. This is accomplished by associating a function $\mathbf{R}(\mathrm{X})$ with each abstract material that takes as input the ball or beam primitive X being processed and that outputs a new (possibly recursive) CLG

structure L. If the function **R** returns a null L, then we consider X to be non-recursive. Otherwise, we let L replace X as representation the primitive's solid geometry. For example, in Figure 10.1, when a cyan beam X is passed into the function **R**, a new CLG replacement of the beam is generated by **R** and returned.

The function **R** must be carefully designed so that it outputs a CLG L that can act as a reasonable replacement of X. For processing elegance, we require that X be a bounding volume of L (i.e., $\mathrm{L} \subset \mathrm{X}$). Care must also be taken when designing a recursive CLG so that infinite recursions are not defined. A simple, although inconvenient, way to prevent infinite recursion is to disallow reusing the same abstract material in different levels of the CLG hierarchy. An example recursive CLG is shown in Figure 10.1.

We have kept the function **R** simple here by only giving it one parameter X. However, variations of **R** may be used. For example, **R** may also be given a parameter for the current recursion depth, which may be used in an alternative strategy for preventing infinite recursions.

10.2 PMC queries and voxelization

Here we consider the implementation of PMC queries on recursive CLG structures. In section 5.5, we discussed a CSG tree T that represents the construction of a CLG solid S as a Boolean combination of the primitive balls and beams. Now, we must consider that the primitive balls and beams may be replaced by entire CLG solids. Let X be a primitive represented by the leaf node Y in T. If X is a recursive primitive, as defined by its abstract material's function **R**, then X is to be replaced by the CLG solid L which has its own CSG tree T_L. So, for T to represent the recursion of X, the leaf node Y is to be replaced by T_L as a sub-tree.

PMC can be implemented for recursive CLG by simply using the same strategy from section 5.5 after replacing the leaf nodes in T, that represent a recursive solid, by their respective CSG subtrees. The performance of this approach can be improved by taking

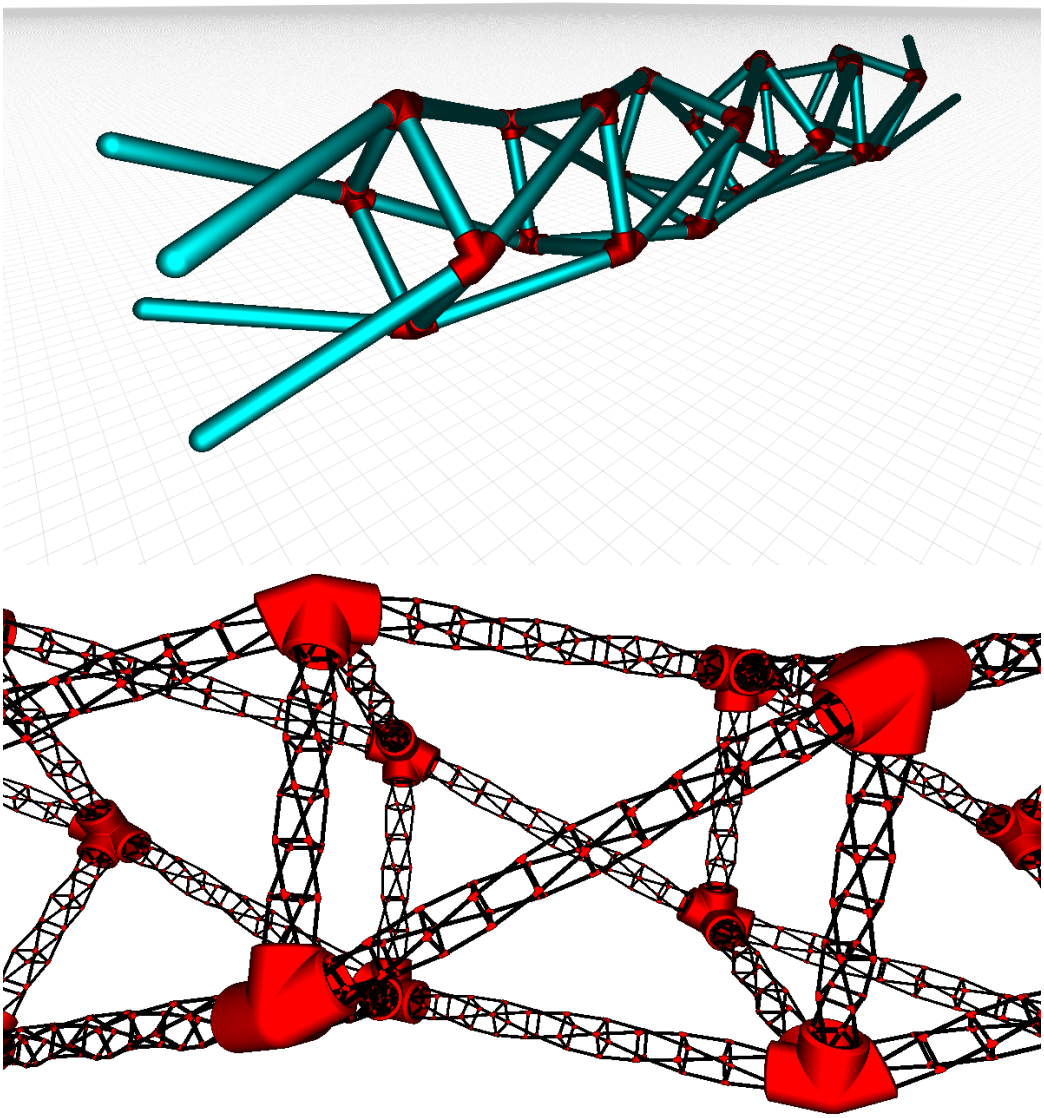


Figure 10.1: Two levels of a recursive CLG beam. The cyan beams of the top structure are replaced to create the structure on the bottom. Note that, where multiple cyan beams intersect (inside the red connectors), the replacement CLG structures of the intersecting beams may intersect or be tangled. The intersecting or tangled portions of the replacement structures may be trimmed (removed) via a Boolean intersection with the Boolean intersection of several planar half-spaces. We do not discuss the details of this trimming operation, however, the details of such a trimming operation are discussed in [13].

advantage of the constraint that X be a bounding volume for L. That is, if X does not contain a query point Q then L does not contain Q.

Like for non-recursive CLG structures, voxelization can be completed by classifying each voxel as in or out based on the result of evaluating a PMC query on S for the center of each voxel.

10.3 Ray intersection queries

In section 5.7, we discussed how to compute the first intersection of a ray R, with origin O and direction $\vec{D}$, against a non-recursively defined CLG solid S. Here we discuss how to modify this query to compute the first ray intersection against S if it is recursively defined.

Consider the original algorithm for sphere tracing a non-recursive CLG. When processing a leaf node in the CSG tree T, the distance (possibly approximate) between the query point Q and the primitive G is computed.

Now, for the recursive version, the primitive G may be replaced with a new CLG solid L, so the original leaf node is replaced by a sub-tree T_s representing L. Therefore, the computed distance between Q and G is replaced by a computed distance between Q and L.

Remember that if Q is not inside the bounding conical shell S of L, then the approximate distance for the sub-tree that represents L can be computed quickly in closed-form as the first intersection between $\mathbf{Ray}(Q, \vec{D})$ and the conical shell S.

CHAPTER 11

IDEAS FOR FUTURE WORK ON MÖBIUS PATTERNS

In this chapter we discuss several different generalizations of the similarity steady patterns, maps, and warps that may be useful for designing lattices and other periodic structures. In particular, we discuss Möbius steady maps along with their associated patterns and lattices.

We will first focus primarily on planar maps, patterns, and lattices. Then we will briefly discuss possible 3D generalizations of the planar results.

Also, the purpose of this chapter is not to provide a significant amount of detail on these possible generalizations. This thesis is primarily focused similarity steady patterns and lattices, but we briefly discuss these generalizations because we believe they are interesting and worth exploring. We have implemented the ideas discussed here, unless stated otherwise, and our implementations show positive results. Despite the positive results, not everything has been proven rigorously, and we intend to further explore these ideas in future work.

In section 2.4, we defined a similarity steady 1-pattern P with the formula $\mathrm{P}[i] = \mathbf{U}^i \circ \mathrm{P}[0]$, for similarity $\mathbf{U}$ and integer i. However, $\mathbf{U}$ does not need to be a similarity. The formula could just as well represent a steady pattern based on any transformation $\mathbf{U}$. For example, we may let $\mathbf{U}$ be a Möbius transformation or an affinity [36]. The transformations in the formulas for 2-patterns and 3-patterns may also be non-similarities.

Additionally, if $\mathbf{U}$ is a transformation for which it is possible to compute real number powers, then it is possible to define a steady 1-field $\mathbf{S}(u) = \mathbf{U}^u$, for real number u. We may likewise use the formulas for steady 2-fields and 3-fields with non-similarity transformations given that it is possible to compute real number powers of the transformations.

Our contributions in this chapter include,

- The introduction of the Möbius frame, which is a curved coordinate frame (general-

ization of a similarity frame) that may be useful for controlling and reasoning about Möbius transformations.

- A construction of a steady Möbius 1-pattern defined by two Möbius frames.
- A method for computing the real number power of a Möbius transformation as the composition of a circle inversion and the real number power of a similarity.
- A generalization of similarity steady patterns, maps, and lattices to Möbius steady patterns, maps, and lattices.
- The introduction of the planar Trans-Möbius Interpolant (TMI) map that is a generalization of COTS maps (as well as of several maps). A TMI map **M** is controlled by five points (4 corners and 1 mid-edge point) of the image of the unit-square, and the map is Trans-Möbius, which means that translations in the parameter space of **M** correspond to Möbius transformations in the image of **M**.

11.1 Preliminary: Planar Möbius transformations

A Möbius transformation is a generalization of the orientation preserving similarity transformations. Like similarities, Möbius transformations are conformal, which means they preserve both angle magnitude and orientation locally.

Unlike similarities, the Möbius transformations do not generally preserve shape. In other words, Möbius transformations result in space deformation or warping. Straight lines are generally transformed into circles by a Möbius transformation, and circles are generally transformed into circles. To be more precise, clines are transformed into clines by a Möbius transformation, where a **cline** is either a circle or a line. Here, lines may be considered as a special case of circles that have infinite radius, so we consider lines and circles to be the same type of geometric object called a cline. It is also useful to think of lines as circles that pass through the point at infinity, denoted as ∞.

A Möbius transformation $\mathbf{G}$ may be thought of as a map $\mathbf{G}(x, y)$ of point coordinates x and y such that $\mathbf{G} \circ (x, y) = \mathbf{G}(x, y)$. A Möbius transformation can transform the point at infinity, which we denote as $\mathbf{G} \circ \infty = \mathbf{G}(\infty)$. For example, if $\mathbf{G}$ transforms a line into a circle, then $\infty \neq \mathbf{G} \circ \infty$, because Möbius transformations preserve clines, but only the preimage cline passes through ∞, so the point at infinity must have been transformed by $\mathbf{G}$.

Möbius transformations have a closed-form inverse $\mathbf{G}^{-1}$, and it is possible to compute real number powers of Möbius transformations, which we describe in section 11.2.

A planar Möbius transformation has 6 DoF and may be controlled by 3 points A, B, and C. We define a **Möbius frame** $\{A, B, C\}$ to be three points that define a Möbius transformation $\mathbf{G}$ such that $A = \mathbf{G}(0, 0)$, $B = \mathbf{G}(1, 0)$, and $C = \mathbf{G}(0, 1)$. A Möbius frame is a curved local coordinate frame where the x-axis is represented by a cline through A and B and the y-axis is represented by a cline through A and C. The x- and y-axis clines meet in right angles at two points A and $\mathbf{G}(\infty)$, the images of $(0, 0)$ and ∞. An example Möbius frame is shown in Figure 11.1 with the deformation of space that it defines.

A Möbius transformation may be composed of two (or any even number of) **cline inversions**, more commonly called circle inversions. Cline inversions are a generalization of reflections. Each cline inversion reverses orientation, so a composition of an even number of cline inversions is orientation preserving. Cline inversions are discussed more in subsection 11.1.1.

A Möbius transformation may also be composed of an even number of cline inversions and orientation reversing similarities. Figure 11.2 demonstrates how cline inversions and orientation reversing similarities are related to Möbius transformations.

11.1.1 Cline inversions

Consider a cline inversion $\mathbf{R}$ over a cline I. A cline inversion is a generalization of reflection such that, if I is a line, then $\mathbf{R}$ is a reflection. However, if I is a circle, then $\mathbf{R}$ is a circle

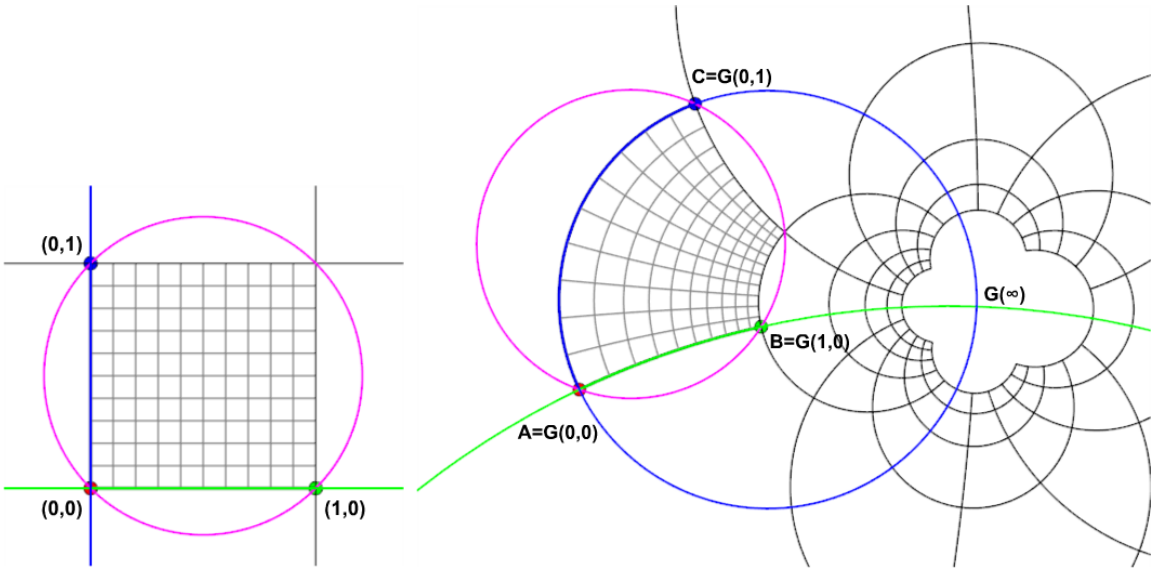


Figure 11.1: (Left) The identity frame and iso-curves of undeformed space. (Right) A Möbius frame $\{A, B, C\}$ and iso-curves of the deformation of space by the transformation **G** defined by the frame. The x-axis cline is shown in green and the y-axis cline is shown in blue. The x- and y-axes meet at A and $\mathbf{G}(\infty)$. A magenta circle is shown circumscribing the unit-square in the undeformed space, and its image is also a circle that circumscribes the image of the unit-square. Notice that all angles are locally preserved, for example, the magenta circle passes through the x- and y-axes at a $45°$ angle.

inversion. Circle inversions are less well known than reflections, so we briefly present circle inversions here.

Let I be a circle with center C and radius r, and let **R** be the circle inversion about I. A point P is transformed by **R** such that $\mathbf{R} \circ P = C + \frac{r^2 \overrightarrow{CP}}{\overrightarrow{CP}^2}$ [46]. We let $\mathbf{R} \circ C = \infty$ and $\mathbf{R} \circ \infty = C$. We list some useful properties of circle inversions:

- A circle inversion is its own inverse, so $\mathbf{R} = \mathbf{R}^{-1}$.
- Points on I are unchanged by the inversion, so $\mathrm{I} = \mathbf{R} \circ \mathrm{I}$.
- Inversion reflects points over I such that points on the inside move outside and points on the outside move inside.
- Clines are mapped to clines.
- A line through C is unchanged by **R**.

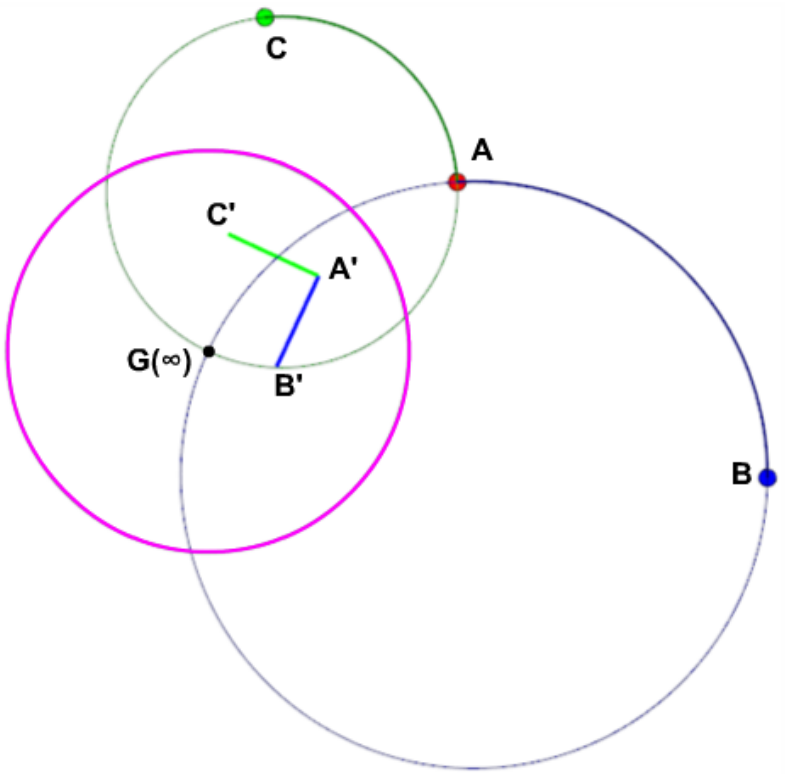


Figure 11.2: A magenta circle of arbitrary radius is centered at $\mathbf{G}(\infty)$, where **G** is a Möbius transformation defined by the Möbius frame $\{A, B, C\}$. Let **R** be the inversion over the magenta circle. Transforming the Möbius frame by **R** results in a similarity frame $\{A', B', C'\}$ with the opposite orientation of the Möbius frame, because the x- and y-axes of the Möbius frame pass through the center of the inversion circle, which causes their images to be lines.

- A cline orthogonal to I is unchanged by **R**.
- A line not through C is mapped to a circle through C.
- A circle through C is mapped to a line not through C.

Figure 11.3 shows an example circle inversion of a pattern of rectangles. The result is a pattern of deformed rectangles whose sides are circular arcs. Figure 11.2 also shows a circle inversion from a Möbius frame to a similarity frame of opposite orientation.

11.1.2 Complex number form of planar Möbius transformations

Although we like to think of Möbius transformations in terms of either Möbius frames or compositions of an even number of circle inversions and orientation-reversing similarities,

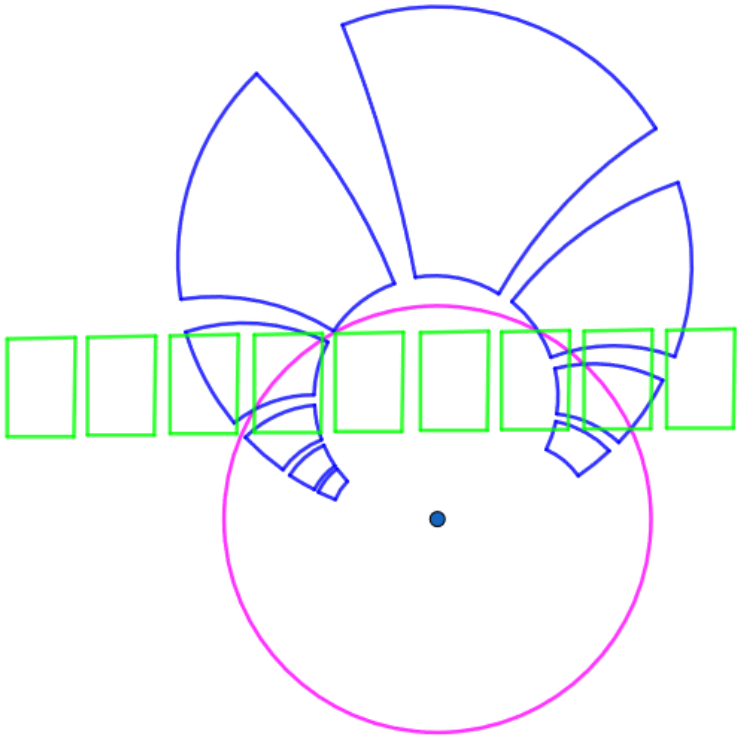


Figure 11.3: The blue pattern is the circle inversion of the green pattern over the magenta circle. This image was produced with the Geogebra geometry software [17]

.

a complex number form of Möbius transformations is more common and is useful for computation. We briefly present the complex number form here.

A Möbius transformation $\mathbf{M}$ is typically defined as a transformation, of the extended complex plane, in the form $\mathbf{M}(z) = \frac{az+b}{cz+d}$, where z, a, b, c, and d are extended complex numbers, and the determinant $ad - bc \neq 0$ [31].

The inverse of a Möbius transformation is $\mathbf{M}^{-1}(z) = \frac{-dz+b}{cz-a}$ [31].

The number z represents the point we wish to transform, and we represent a point (x, y) in the plane as the complex number $x + yi$, where i is the imaginary unit. Special care should be taken to ensure that the division of the extended complex numbers behaves as expected when dividing by either 0 or ∞. For example, division of a non-zero value by zero should yield infinity, and similarly, division of a non-infinity value by infinity should yield zero.

The Möbius transformation, in complex number form, is specified by four complex numbers a, b, c, and d. However, a Möbius transformation is uniquely determined by just three complex numbers, so different choices of a, b, c, and d may yield the same transformation. In fact, the transformation remains the same after multiplying a, b, c, and d each by the same non-zero complex number [30]. It is common to normalize a Möbius transformation so that $ad - bc = 1$. This normalization is computed by dividing a, b, c, and d by $\sqrt{ad - bc}$ [30]. We always normalize Möbius transformations between operations, because we found that doing so improves numerical precision.

The composition of two Möbius transformations is given by

$$\mathbf{M}_2(z) \circ \mathbf{M}_1(z) = \frac{(a_2a_1 + b_2c_1)z + a_2b_1 + b_2d_1}{(c_2a_1 + d_2c_1)z + c_2b_1 + d_2d_1} \tag{11.1}$$

where $\mathbf{M}_1(z) = \frac{a_1z+b_1}{c_1z+d_1}$ and $\mathbf{M}_2(z) = \frac{a_2z+b_2}{c_2z+d_2}$ [31].

The fixed points of a Möbius transformation are computed as $\frac{a-d\pm\sqrt{(d-a)^2+4bc}}{2c}$, which is derived by equating $z = \frac{az+b}{cz+d}$ and solving for z. Notice that two fixed points generally exist. Also, notice that infinity is a fixed point when $c = 0$, which implies that the transformation is an orientation-preserving similarity [30].

11.1.3 Conversion between the complex number form and the frame form

Given a Möbius transformation $\mathbf{M}$ in its complex number form, we convert $\mathbf{M}$ into its Möbius frame form $\{A, B, C\}$, by letting $A = \mathbf{M} \circ (0, 0)$, $B = \mathbf{M} \circ (1, 0)$, and $C = \mathbf{M} \circ (0, 1)$.

Now, given the Möbius frame $\{A, B, C\}$, we want to convert it into the complex number form. We do this by summarizing a method, described in [31], for computing the Möbius transformation that takes any three distinct complex numbers to any other three distinct

complex numbers. Consider the parameterized Möbius transformation in the form

$$\mathbf{F}(z_0, z_1, z_\infty) = \frac{(z - z_0)(z_1 - z_\infty)}{(z - z_\infty)(z_1 - z_0)} \tag{11.2}$$

where z_0, z_1, and z_∞ are complex numbers. Transformation **F** satisfies the constraints $\mathbf{F}(z_0) = 0$, $\mathbf{F}(z_1) = 1$, and $\mathbf{F}(z_\infty) = \infty$, which is easily checked. We want a transformation **T** that satisfies $\mathbf{T}(0, 0) = A$, $\mathbf{T}(1, 0) = B$, and $\mathbf{T}(0, 1) = C$. The solution is to define the desired Möbius transformation as the composition

$$\mathbf{T} = \mathbf{F}^{-1}(A, B, C) \circ \mathbf{F}((0, 0), (1, 0), (0, 1)) \tag{11.3}$$

This method may be used to compute the Möbius transformation between any two Möbius frames, by replacing the points $\{(0, 0), (1, 0), (0, 1)\}$, with the points of another Möbius Frame.

11.2 Planar Möbius steady 1-patterns

The Möbius steady 1-patterns are a generalization of the similarity steady 1-patterns.

A general Möbius steady pattern has two fixed points F_1 and F_2. If one or both of the fixed points are ∞, then the Möbius transformation that defines the pattern is a similarity. If both fixed points are ∞, then the Möbius transformation is a translation.

If neither fixed point is ∞, then the path traced by a point that is iteratively transformed by the Möbius transformation **U** is a **loxodrome** (also called a **double spiral**). In special cases, the loxodrome may degenerate into a cline. If one fixed point is ∞, such that **U** is a similarity, then the path traced by an iteratively traced point is a logarithmic spiral. Loxodromes are a generalization of logarithmic spirals that may be obtained by transforming a logarithmic spiral by a circle inversion.

If the template shape $\mathrm{P}[0]$ of a Möbius steady pattern is a cline, then all shapes in the pattern are clines.

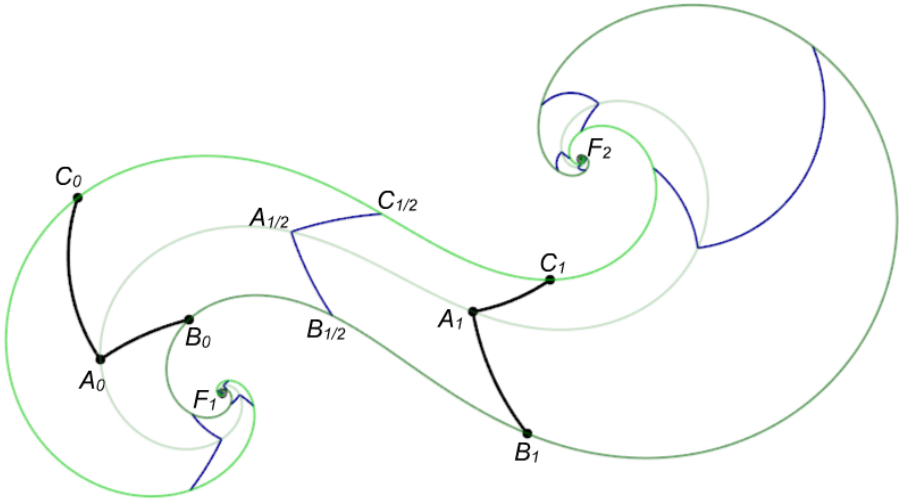


Figure 11.4: A Möbius steady pattern of Möbius frames. Let **G** be the Möbius transform that takes $\mathrm{P}[0]$ to $\mathrm{P}[1]$. The transformation between the displayed consecutive frames is $\mathbf{G}^{1/2}$. The two fixed points F_1 and F_2 of **G** are shown. The pattern appears to emerge from F_1 and converge towards F_2. Three loxodromes are shown in green, and each passes through corresponding points of the Möbius frames. If we were to transform this pattern by an inversion about a circle centered at either F_1 or F_2, then the result would be a similarity steady pattern of frames, and the loxodromes would become logarithmic spirals.

11.2.1 Möbius steady patterns are similarity steady patterns under a circle inversion

Consider the Möbius steady pattern $\mathrm{P}[i] = \mathbf{U}^i \circ \mathrm{P}[0]$. Let $\mathrm{P}[0]$ be a Möbius frame, so all consecutive pairs of shapes in P are Möbius frames that are related by the same Möbius transformation **U**. Such a pattern of Möbius frames is shown in Figure 11.4.

The pattern P of frames may be composed by a circle inversion on a similarity steady pattern of (possibly curved) Möbius frames. Consider the case where **U** is a non-similarity Möbius transformation such that **U** has two distinct fixed points. If P was to be transformed by a circle inversion **R** over any circle centered at one of the fixed points F, then the resulting pattern is a similarity steady pattern $\mathrm{P}' = \mathbf{R} \circ \mathrm{P}$, because the image of F is $\infty = \mathbf{R} \circ F$.

11.2.2 Controlling a Möbius steady pattern by two frames

Given two Möbius frames $\mathrm{P}[0] = \{A_0, B_0, C_0\}$ and $\mathrm{P}[1] = \{A_1, B_1, C_1\}$ that define a Möbius steady pattern P, the i^{th} frame $\mathrm{P}[i]$ is computed with the formula

$$\mathrm{P}[i] = \mathbf{R} \circ \textbf{Similarity}(\mathbf{R} \circ \{A_0, B_0\}, \mathbf{R} \circ \{A_1, B_1\})^i \circ \mathbf{R} \circ \{A_0, B_0, C_0\} \tag{11.4}$$

where **R** is a circle inversion over any circle centered at a fixed point of the Möbius transformation that takes $\mathrm{P}[0]$ to $\mathrm{P}[1]$.

This formula works by using **R** to first transform the given frames of the Möbius steady pattern P into frames of a similarity steady pattern P'. Then it computes $\mathrm{P}'[i]$, and finally it uses **R** to transform $\mathrm{P}'[i]$ into $\mathrm{P}[i]$.

It is possible to compute real number powers of similarities, so it is possible to compute a smooth, steady interpolation between two Möbius frames, which we demonstrate in the following function where the u parameter is the power of the Möbius transformation between the given frames

$$\begin{aligned} &\textbf{MöbiusFrame}(\{A_0, B_0, C_0\}, \{A_1, B_1, C_1\}, u) = \\ &\qquad \mathbf{R} \circ \textbf{Similarity}(\mathbf{R} \circ \{A_0, B_0\}, \mathbf{R} \circ \{A_1, B_1\})^u \circ \mathbf{R} \circ \{A_0, B_0, C_0\} \end{aligned} \tag{11.5}$$

Figure 11.4 shows a Möbius steady pattern of frames defined using Equation 11.5. The pattern includes a frame $\{A_{1/2}, B_{1/2}, C_{1/2}\}$ that is a real number power $\mathbf{G}^{1/2}$ of the transformation between the control frames $\mathrm{P}[0]$ and $\mathrm{P}[1]$. The interpolation between two Möbius frames may be useful for controlling pleasant animations with bending.

11.2.3 Computing powers of Möbius transformations

Given a Möbius transformation **U**, we are interested in computing its power $\mathbf{U}^u$, for real number u. Imagine creating a Möbius steady pattern of frames where
$\mathrm{P}[0] = \{(0,0), (1,0), (0,1)\}$ is the identity frame and $\mathrm{P}[1] = \{A, B, C\}$ is the frame rep-

resentation of **U**. The transformation from $\mathrm{P}[0]$ to $\mathrm{P}[1]$ is the Möbius transformation **U**. Therefore, the frame **MöbiusFrame**$(\{A_0, B_0, C_0\}, \{A_1, B_1, C_1\}, u)$ is the frame representation of $\mathbf{U}^u$, which may be converted into complex number form if desired (see subsection 11.1.3).

11.3 Planar Möbius steady slab patterns, maps, and lattices

The Möbius steady 1-patterns can be extended into 2-patterns, much like the similarity steady 2-pattern extension. The Möbius steady 2-patterns and 2-maps have the same form as their similarity steady counterparts, with the only difference being that the transformations are Möbius transformations.

A Möbius steady 2-pattern can be controlled by three Möbius frames X, Y, and Z. Frame X controls the placement and warping of the template shape $\mathrm{P}[0, 0]$. Then, we let the transformations **U** and **V** be the Möbius transformations that take X to Y and take X to Z respectively. However, although simple to define, we do not believe this offers an intuitive control scheme. Improved controls schemes should be investigated. A Möbius steady 2-pattern has 18 DoF, due to the 6 DoF per Möbius frame.

A Möbius steady 2-map has 14 DoF, because both Möbius transformations have 6 DoF each plus 2 more DoF for the origin point. The additional DoF offer a larger design space compared to the similarity steady 2-patterns and 2-maps. For example, Möbius steady 2-maps allow for the creation of iso-curves with inflection points, which is not possible with similarity steady 2-maps.

The two transformations of a Möbius steady 2-pattern or 2-map do not generally commute. Every shape in a Möbius steady 2-pattern belongs to a Möbius steady 1-pattern in the **V** direction, but shapes in the 2-pattern do not generally belong to a Möbius steady 1-pattern in the **U** direction.

If the template shape $\mathrm{P}[0, 0]$ is a cline, then all other shapes in the 2-pattern are clines. Möbius steady 2-patterns and 2-maps may be useful for creating rectified warped slab lat-

tices with nice regularities. A warped slab lattice may be defined using a Möbius steady 2-map, and we want to rectify the lattice with a post processing step. Assume that some black box process has been used to obtain the rectified template node-group $\mathrm{N}[0,0]$. Then, all other rectified node-groups can be computed as $\mathrm{N}[i,j] = \mathbf{V}^j \circ \mathbf{U}^i \circ \mathrm{N}[0,0]$, which constructs the node-groups as parts of a Möbius steady 2-pattern. Finally, cone-beams (trapezoids in 2D) are constructed to smoothly join the appropriate nodes, in the same way as was done for similarity steady lattices. Figure 11.5 shows an example Möbius steady 2-pattern of circles and a Möbius steady rectified slab lattice.

All rectified nodes created with this process are guaranteed to be clines. For practical purposes, the nodes can be guaranteed to be circles, because useful lattices will not have nodes with infinite radius. Each node-group belongs to a Möbius steady 1-pattern in both the **V** and the **U** directions.

However, if the Möbius transformations **U** and **V** of the 2-pattern do not degenerate into similarities, then no beams belong to a Möbius steady 1-pattern, because we defined our beams to be straight cone-beams even though the Möbius transformations generally warp geometry. The lack of steadiness in the beams makes querying rectified Möbius steady lattices more difficult than querying rectified similarity steady lattices, in addition to the added difficulty due to Möbius transformations being more complex than similarities. One possible solution to the lack of steadiness is to generalize the definition of a rectified beam to allow beams to be Dupin cyclides instead of cones, because Dupin cyclides are the result of transforming a cone by a Möbius transformation.

An investigation of acceleration strategies for queries on Möbius steady lattices has not yet been performed. However, we conjecture that accurate integral property queries cannot generally be accelerated beyond $\mathbf{O}(\overline{u}\,\overline{v})$ time, due to the warping caused by Möbius transformations, which may cause extreme deformations. We do however conjecture that PMC and BIQ queries can be accelerated to $\mathbf{O}(\overline{u})$ time for some Möbius steady slab lattices. The idea behind this conjecture is that a Möbius steady 1-pattern of clines is a similarity

steady 1-pattern of clines transformed by a circle inversion. Although lattice beams are neither circles nor part of a Möbius steady 1-pattern, it may be possible to bound the beams by a Möbius steady 1-pattern of circles that can be transformed into a similarity steady 1-pattern of circles. If so, then RangeFinder (presented in chapter 7) can be used to accelerate PMC or BIQ on the resulting similarity steady 1-pattern.

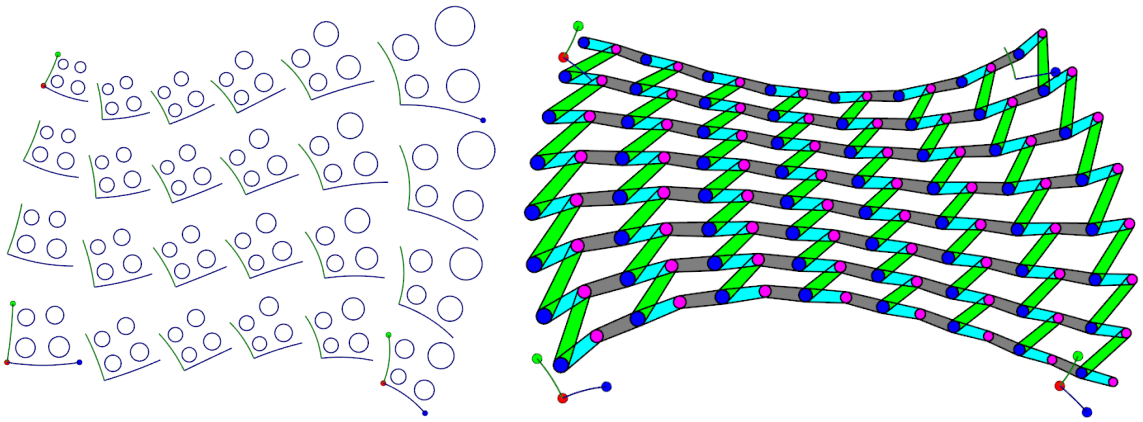


Figure 11.5: (Left) A Möbius steady 2-pattern of circles. Each node-group contains 4 circles. The template node-group is on the bottom-left. The **U** transformation creates the steady pattern from left to right, and **V** creates the pattern from bottom to top. The Möbius frames associated with each node-group are also shown. (Right) A Möbius steady rectified slab lattice. Notice that the global shapes of both patterns cannot be modelled as similarity steady slab patterns.

11.4 Planar Trans-Möbius Interpolant maps, patterns, and lattices

The **Trans-Möbius Interpolant (TMI)** maps, presented here, are a special case of the Möbius steady 2-maps that are also a generalization of the COTS maps, as well as a generalization of several other maps. A TMI has the **Trans-Möbius** property, which means that translations in parameter space are mapped into Möbius transformations. Also, a TMI map **M** is an "interpolant" in the sense that it is controlled by five co-planar points (A, B, C, D, and E) that the image of the unit-square passes through such that $A = \mathbf{M}(0,0)$, $B = \mathbf{M}(1,0)$, $C = \mathbf{M}(0,1)$, $D = \mathbf{M}(1,1)$, and $E = \mathbf{M}(.5,0)$. A TMI map has 10 DoF, because it is defined by 5 points with 2 DoF each.

TMI is a special case of the Möbius steady 2-maps for which the transformations **U** and **V** commute. The transformations **U** and **V** commute when they have the same fixed points. Also, TMI is a generalization of COTS where the transformations **U** and **V** are commutative Möbius transformations instead of commutative similarities.

A TMI may be composed by either a cline inversion or a Möbius transformation of a COTS map, where the cline inversion or the Möbius transformation transforms the similarities of the COTS map into Möbius transformations. Understanding the TMI as a cline inversion of a COTS helps to understand the properties of TMI. For example, the iso-curves of a TMI are loxodromes, because the iso-curves are a cline inversion of the logarithmic spiral iso-curves of a COTS. Also, TMI maps are what we call **cline replicating**. That is, given a shape S in the parameter space that maps to a cline $\mathrm{X} = \mathbf{M}(S)$, if S is translated in the parameter space, then X remains a cline, because translations are mapped to Möbius transformations, which preserve clines.

All tiles in a TMI map are related by a Möbius transformation. A TMI has a uniform angular distortion, because all tiles are related by a Möbius transformation and Möbius transformations are conformal (preserve local angles). A TMI map is **cline replicating**, which means that if the template shape is a cline, then all other shapes in the pattern are clines.

TMI maps may be used to define TMI patterns and lattices. All shapes in a TMI pattern are related by a Möbius transformation. If the template shape is a cline, then all other shapes in the pattern are clines. A TMI pattern of circles is shown in Figure 11.6-Right.

TMI maps have a closed-form inverse, because both COTS and cline inversions have closed-form inverses, and TMI is a composition of a cline inversion and a COTS. The closed-form inverse enables $\mathbf{O}(1)$ time PMC queries on TMI patterns.

However, is it not clear if the closed-form inverse is useful for implementing an $\mathbf{O}(1)$ PMC query for TMI rectified lattices, because like for Möbius steady slab lattices, the rectified beams do not belong to any Möbius steady 1-patterns.

Special cases of the TMI map include COTS, Möbius transformations, and the Four Point Interpolant [22], which is a composition $\mathbf{G}_2 \circ \mathbf{A}\,\mathbf{G}_1$ for Möbius transformations $\mathbf{G}_1$ and $\mathbf{G}_2$ and affinity $\mathbf{A}$.

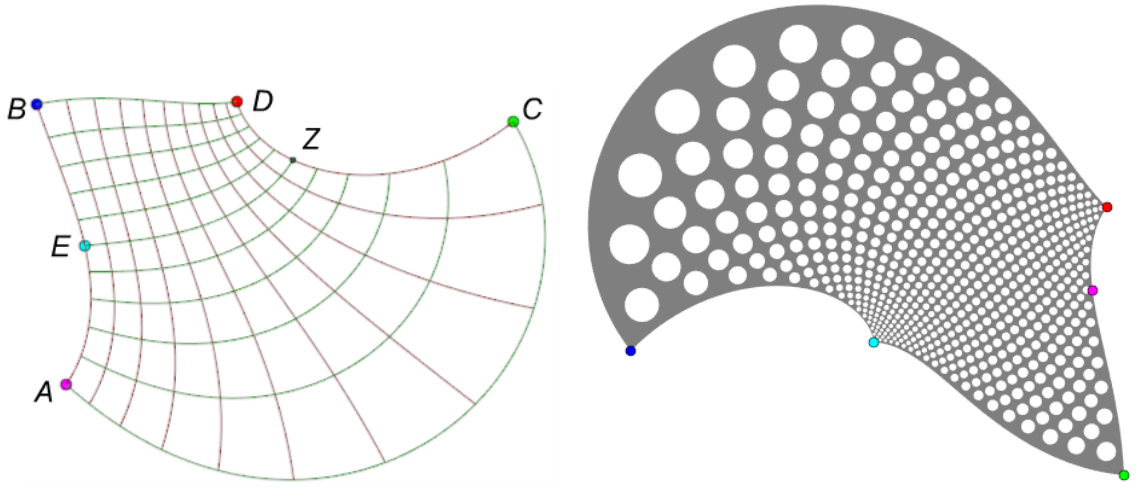


Figure 11.6: (Left) A TMI map controlled by 5 points. The iso-curves are loxodromes, and all tiles are related by a Möbius transformation. (Right) A TMI pattern of circles. Each circle is related by a translation in the parameter space (which is a Möbius transformation in the image).

11.4.1 TMI construction from six points

A TMI map is uniquely defined by five points. However, to understand the construction of a TMI map from five points, it is useful to first understand the construction of a TMI map from six points. Note, however, that the six point construction of a TMI map results in a weak version of TMI for which the image of the unit-square only interpolates four of the six control points. The five point construction of a TMI, described in subsection 11.4.2, results in a proper TMI for which the image of the unit-square interpolates all five control points.

Consider six co-planar control points that are grouped into two Möbius frames $\mathrm{H}_1 = \{A, B, C\}$ and $\mathrm{H}_2 = \{X, Y, Z\}$. Let $\mathbf{U}$ be the Möbius transformation that takes H_1 to H_2. $\mathbf{U}$ has two fixed points that we call F_1 and F_2. Let $\mathbf{R}$ be a circle inversion over a circle of any radius centered at either F_1 or F_2. Let $B' = \mathbf{R} \circ B$, $C' = \mathbf{R} \circ C$, $Y' = \mathbf{R} \circ Y$,

and $Z' = \mathbf{R} \circ Z$ be four of the control points transformed by inversion **R**. Let **M'** be a COTS map defined from the corners B', C', Z', and Y' such that the transformations $\mathbf{U}' = \textbf{Similarity}(\{B', C'\}, \{Y', Z'\})$ and $\mathbf{V}' = \textbf{Similarity}(\{B', Y'\}, \{C', Z'\})$ are the commutative similarity transformations that define $\mathbf{M}'$. Finally, let $\mathbf{M} = \mathbf{R} \circ \mathbf{M'}$ be the TMI map composed as a circle inversion of the COTS map **M'**. The commutative Möbius transformations that define TMI **M** are $\mathbf{U} = \mathbf{R} \circ \mathbf{U}'$ and $\mathbf{V} = \mathbf{R} \circ \mathbf{V}'$, and the origin point is $B = \mathbf{M}(0, 0)$.

For this construction of a TMI, the image of the unit square transformed by **M** interpolates four of the six points such that $B = \mathbf{M}(0, 0)$, $C = \mathbf{M}(0, 1)$, $Y = \mathbf{M}(1, 0)$, and $Z = \mathbf{M}(1, 1)$. However, the points A and X are not generally interpolated by the image of the unit square, and in fact, A and X do not have a very intuitive role in controlling **M**. Remember, a TMI map has 10 DoF, but the construction from six points has 12 input variables (2 per point). Therefore, one control point (representing 2 of the input variables) is redundant. Indeed, only one of either A or X is needed to uniquely construct a TMI. To see this, consider that in the case where the Möbius frames H_1 and H_2 are similar to each other, the transformations **U** and **V** degenerate into similarities and **M** is a COTS map with corners B, C, Y, and Z. Notice that this COTS map does not depend on the points A and X, so the same COTS map will be obtained for any choice of A and X (given the corners B, C, Y, and Z) as long as H_1 and H_2 are similar.

11.4.2 TMI construction from five points

Now, we consider the construction of a TMI map **M** from five points such that $A = \mathbf{M}(0, 0)$, $B = \mathbf{M}(1, 0)$, $C = \mathbf{M}(0, 1)$, $D = \mathbf{M}(1, 1)$, and $E = \mathbf{M}(.5, 0)$. Figure 11.6-Left shows a TMI map defined by these points. We already have a construction for TMI maps using six points, so we want to reuse the existing construction here. However, we only have five points, so we will need to compute a sixth point Z that allows us to reuse the six point construction. An arbitrary choice of Z, for example as the mid point between C

and D, results in a valid Trans-Möbius map. However, an arbitrary choice of Z will not yield a TMI map that satisfies the desired interpolation constraint of $E = \mathbf{M}(.5, 0)$. To compute Z, we first let $\mathbf{I}$ be the Möbius transformation that takes Möbius frame $\{A, B, C\}$ to frame $\{D, C, B\}$. Then, we let $Z = \mathbf{I} \circ E$. Finally, we construct two Möbius frames $\mathrm{H}_1 = \{E, A, B\}$ and $\mathrm{H}_2 = \{Z, C, D\}$ that we use to construct the TMI map, as was done in the six point construction.

Admittedly, we do not fully understand why our choice of Z works to satisfy the interpolation constraint $E = \mathbf{M}(.5, 0)$. We accidentally discovered this choice of Z after noticing that the transformation $\mathbf{I}$ behaves like a $180°$ rotation of the parameter space unit-square about its center $(.5, .5)$. Therefore, in our intuition based solution, if E is to represent $\mathbf{M}(.5, 0)$, then applying $\mathbf{I}$ to E will result in a point $Z = \mathbf{M}(.5, 1)$. Further investigation of this result is needed.

Sometimes, for a smooth movement of the control points, the resulting TMI map may not transform smoothly, resulting in a visual jump. These jumps may even break the interpolation constraint of $E = \mathbf{M}(.5, 0)$ for some configurations of control points. The jumps in the TMI map are due to jumps in the COTS map when, for a small change in control points, the rotation angles, represented in the COTS map's transformations, jump, for example, from just above $-180°$ to just below $180°$. By carefully tracking the changes in the control points of COTS, it is possible to detect and prevent these jumps by allowing the rotation angles to be outside of the range $(-180, 180)°$.

11.5 3D Steady Möbius patterns, maps, and lattices

Here, we briefly discuss conjectured 3D extensions of the planar Möbius steady patterns and maps. Many of the properties of planar Möbius transformations generalize well to 3D Möbius transformations. 3D Möbius transformations are a generalization of 3D similarities. 3D Möbius transformations are conformal and preserve angles locally. 3D Möbius transformations generally cause warping and do not preserve shapes. Like for planar

Möbius transformations, the 3D Möbius transformations preserve clines. However, 3D Möbius transformations also preserve splanes, where a **splane** is either a sphere or a plane. Here, it is useful to think of planes as spheres that pass through the point at infinity, denoted ∞. A plane may also be thought of as a sphere with infinite radius.

A 3D Möbius transformation may be composed of an even number of **splane inversions**, where a splane inversion is the 3D generalization of a cline inversion. Generally, a splane inversion is a reflection over a sphere, where the sphere may degenerate into a plane. The properties of splane inversions generalize straightforwardly from cline inversions, so we do not discuss them in detail. A 3D Möbius transformation may also be composed of an even number of splane inversions and orientation reversing 3D similarities.

For representing and controlling a planar Möbius transformation $\mathbf{G}$, we introduced the Möbius frame $\{A, B, C\}$ such that $A = \mathbf{G}(0, 0)$, $B = \mathbf{G}(1, 0)$, and $C = \mathbf{G}(0, 1)$. We might wish to generalize this frame representation and control scheme into 3D as a frame of four 3D points, but unfortunately, four 3D points offer 12 DoF, which is more DoF than the 10 DoF that a 3D Möbius transformation has. To see that a 3D Möbius transformation has 10 DoF, consider representing a Möbius transformation by an orientation reversing similarity and a sphere inversion. The 10 DoF is the sum of 7 DoF from the orientation reversing similarity and 3 DoF from the center of the inversion sphere. The radius of the inversion sphere does not contribute to the DoF, because the same Möbius transformation can be obtained for different radii of the inversion sphere given a different orientation reversing similarity. Creating intuitive control schemes for 3D Möbius transformations is an open problem.

It is possible to create 3D Möbius 1-, 2-, and 3-patterns, 1-, 2-, and 3-maps, and row, slab, and brick lattices. Note however that we have not actually implemented any of these. Creating control schemes and representations for these is non-trivial, given the difficulty of controlling and representing 3D Möbius transformations. However, these ideas may still be understood when compared to their planar counterparts. For example, a 3D Möbius

1-pattern is a splane inversion of a 3D similarity 1-pattern, and a 3D Möbius 2-pattern is a 3D Möbius 1-pattern of 3D Möbius 1-patterns. The properties of the 3D Möbius patterns, maps, and lattices can be analyzed in the same way that we have already analyzed the properties of their planar counterparts.

Finally, we conjecture that it is possible to create a generalization of the BeCOTS maps and patterns where the commutative similarities are commutative Möbius transformations. This generalization would also be a non-planar generalization of the TMI maps and patterns, and it would be a special case of 3D Möbius steady 2-patterns for which the transformations commute. We call this conjectured map a **Bent TMI (BeTMI)**, because it is a non-planar version of TMI. We expect that a BeTMI map would be a composition of a splane inversion and a BeCOTS map. We expect that the BeTMI would lie on the surface of a Dupin cyclide, because BeCOTS lies on a cone and a splane inversion or a Möbius transformation of a cone is a Dupin cyclide. We do not know if an intuitive control scheme (five 3D points for example) exists for such a BeTMI map.

REFERENCES

[1] S. J. Ahn, W. Rauh, and H.-J. Warnecke, "Least-squares orthogonal distances fitting of circle, sphere, ellipse, hyperbola, and parabola," *Pattern Recognition*, vol. 34, no. 12, pp. 2283–2303, 2001.

[2] G. Allen, "White paper: Ntopology's implicit modeling technology," nTopology, Tech. Rep.

[3] S. R. Almeida, G. H. Paulino, and E. C. Silva, "Layout and material gradation in topology optimization of functionally graded structures: A global–local approach," *Structural and Multidisciplinary Optimization*, vol. 42, no. 6, pp. 855–868, 2010.

[4] A. O. Aremu, J. Brennan-Craddock, A. Panesar, I. Ashcroft, R. J. Hague, R. D. Wildman, and C. Tuck, "A voxel-based method of constructing and skinning conformal and functionally graded lattice structures suitable for additive manufacturing," *Additive Manufacturing*, vol. 13, pp. 1–13, 2017.

[5] A. Barbier and E. Galin, "Fast distance computation between a point and cylinders, cones, line-swept spheres and cone-spheres," *Journal of Graphics tools*, vol. 9, no. 2, pp. 11–19, 2004.

[6] A. H. Barr, "Global and local deformations of solid primitives," in *Readings in Computer Vision*, Elsevier, 1987, pp. 661–670.

[7] P. Bo, H. Pottmann, M. Kilian, W. Wang, and J. Wallner, "Circular arc structures," *ACM Transactions on Graphics (TOG)*, vol. 30, no. 4, pp. 1–12, 2011.

[8] L. Busé, A. Galligo, and J. Zhang, "Extraction of cylinders and cones from minimal point sets," *Graphical Models*, vol. 86, pp. 1–12, 2016.

[9] P. F. Egan, S. J. Ferguson, and K. Shea, "Design of hierarchical three-dimensional printed scaffolds considering mechanical and biological factors for bone tissue engineering," *Journal of Mechanical Design*, vol. 139, no. 6, p. 061 401, 2017.

[10] G. Elber, "Precise construction of micro-structures and porous geometry via functional composition," in *International Conference on Mathematical Methods for Curves and Surfaces*, Springer, 2016, pp. 108–125.

[11] A. G. Evans, M. He, V. S. Deshpande, J. W. Hutchinson, A. J. Jacobsen, and W. B. Carter, "Concepts for enhanced energy absorption using hollow micro-lattices," *International Journal of Impact Engineering*, vol. 37, no. 9, pp. 947–959, 2010.

[12] O. Fryazinov, T. Vilbrandt, and A. Pasko, "Multi-scale space-variant frep cellular structures," *Computer-Aided Design*, vol. 45, no. 1, pp. 26–34, 2013.

[13] A. Gupta, G. Allen, and J. Rossignac, "Quador: Quadric-of-revolution beams for lattices," *Computer-Aided Design*, vol. 102, pp. 160–170, 2018.

[14] A. Gupta, K. Kurzeja, J. Rossignac, G. Allen, P. S. Kumar, and S. Musuvathy, "Programmed-lattice editor and accelerated processing of parametric program-representations of steady lattices," *Computer-Aided Design*, vol. 113, pp. 35–47, 2019.

[15] J. C. Hart, "Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces," *The Visual Computer*, vol. 12, no. 10, pp. 527–545, 1996.

[16] C. Hoffmann, V. Shapiro, and V. Srinivasan, "Geometric interoperability via queries," *Computer-Aided Design*, vol. 46, pp. 148–159, 2014.

[17] International Geogebra Institute, *Geogebra - geometry*, https://www.geogebra.org/geometry, 2021.

[18] B. Kim and J. Rossignac, "Collision prediction for polyhedra under screw motions," in *Proceedings of the eighth ACM symposium on Solid modeling and applications*, ACM, 2003, pp. 4–10.

[19] K. Kurzeja and J. Rossignac, "Rangefinder: Accelerating ball-interference queries against steady lattices," *Computer-Aided Design*, vol. 112, pp. 14–22, 2019.

[20] ——, "Becots: Bent corner-operated tran-similar maps and lattices," *Computer-Aided Design*, vol. 129, p. 102 912, 2020.

[21] D. Li, W. Liao, N. Dai, G. Dong, Y. Tang, and Y. M. Xie, "Optimal design and modeling of gyroid-based functionally graded cellular structures for additive manufacturing," *Computer-Aided Design*, vol. 104, pp. 87–99, 2018.

[22] Y. Lipman, V. G. Kim, and T. A. Funkhouser, "Simple formulas for quasiconformal plane deformations," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 5, p. 124, 2012.

[23] ——, "Simple formulas for quasiconformal plane deformations," *ACM Trans. Graph.*, vol. 31, no. 5, 124:1–124:13, 2012.

[24] X. Liu and V. Shapiro, "Multiscale shape–material modeling by composition," *Computer-Aided Design*, vol. 102, pp. 194–203, 2018.

[25] I. Llamas, B. Kim, J. Gargus, J. Rossignac, and C. D. Shaw, "Twister: A space-warp operator for the two-handed editing of 3d shapes," *ACM transactions on graphics (TOG)*, vol. 22, no. 3, pp. 663–668, 2003.

[26] J. Martınez, J. Dumas, and S. Lefebvre, "Procedural voronoi foams for additive manufacturing," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 44, 2016.

[27] R. Mesnil, Y. Santerre, C. Douthe, O. Baverel, and B. Leger, "Generating high node congruence in freeform structures with monge's surfaces," in *Proceedings of IASS Annual Symposia*, International Association for Shell and Spatial Structures (IASS), 2015.

[28] L. R. Meza, A. J. Zelhofer, N. Clarke, A. J. Mateos, D. M. Kochmann, and J. R. Greer, "Resilient 3d hierarchical architected metamaterials," *Proceedings of the National Academy of Sciences*, vol. 112, no. 37, pp. 11 502–11 507, 2015.

[29] D. Montoya-Zapata, D. A. Acosta, C. Cortés, J. Pareja-Corcho, A. Moreno, J. Posada, and O. Ruiz-Salguero, "Approximation of the mechanical response of large lattice domains using homogenization and design of experiments," *Applied Sciences*, vol. 10, no. 11, p. 3858, 2020.

[30] D. Mumford, C. Series, and D. Wright, *Indra's Pearls: the Vision of Felix Klein*. Cambridge University Press, 2002.

[31] J. Olsen, "The geometry of möbius transformations," University of Rochester, 2010.

[32] A. Pasko, O. Fryazinov, T. Vilbrandt, P.-A. Fayolle, and V. Adzhiev, "Procedural function-based modelling of volumetric microstructures," *Graphical Models*, vol. 73, no. 5, pp. 165–181, 2011.

[33] N. M. Patrikalakis and T. Maekawa, *Shape interrogation for computer aided design and manufacturing*. Springer Science & Business Media, 2009.

[34] J. Rossignac, "Squint fields, maps, patterns, and lattices," Georgia Institute of Technology, Tech. Rep., 2018.

[35] ——, "Corner-operated tran-similar (cots) maps, patterns, and lattices," *ACM Trans. Graph.*, vol. 39, no. 1, Feb. 2020.

[36] J. Rossignac and Á. Vinacua, "Steady affine motions and morphs," *ACM Transactions on Graphics (TOG)*, vol. 30, no. 5, pp. 1–16, 2011.

[37] J. R. Rossignac and J. J. Kim, "Computing and visualizing pose-interpolating 3d motions," *Computer-Aided Design*, vol. 33, no. 4, pp. 279–291, 2001.

[38] J. R. Rossignac and A. A. Requicha, "Offsetting operations in solid modelling," *Computer Aided Geometric Design*, vol. 3, no. 2, pp. 129–148, 1986.

[39] T. A. Schaedler and W. B. Carter, "Architected cellular materials," *Annual Review of Materials Research*, vol. 46, pp. 187–210, 2016.

[40] T. A. Schaedler, A. J. Jacobsen, A. Torrents, A. E. Sorensen, J. Lian, J. R. Greer, L. Valdevit, and W. B. Carter, "Ultralight metallic microlattices," *Science*, vol. 334, no. 6058, pp. 962–965, 2011.

[41] C. Schreck, D. Rohmer, S. Hahmann, M.-P. Cani, S. Jin, C. C. L. Wang, and J.-F. Bloch, "Nonsmooth developable geometry for interactively animating paper crumpling," *ACM Trans. Graph.*, vol. 35, no. 1, Dec. 2016.

[42] T. W. Sederberg and S. R. Parry, "Free-form deformation of solid geometric models," *ACM SIGGRAPH computer graphics*, vol. 20, no. 4, pp. 151–160, 1986.

[43] P. Shirley and S. Marschner, *Fundamentals of Computer Graphics*. AK Peters, Ltd., 2009.

[44] M. Van Emmerik, A. Rappoport, and J. Rossignac, "Simplifying interactive design of solid models: A hypertext approach," *The Visual Computer*, vol. 9, no. 5, pp. 239–254, 1993.

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

[46] E. W. Weisstein, *"inversion." from mathworld–a wolfram web resource*, http://mathworld.wolfram.com/Inversion.html, Accessed: 2019-17-01.

[47] Y. Wu, A. Gupta, K. Kurzeja, and J. Rossignac, "Chocc: Convex hull of cospherical circles and applications to lattices," *Computer-Aided Design*, vol. 129, p. 102 903, 2020.

[48] L. Yang, O. Harrysson, H. West, and D. Cormier, "Mechanical properties of 3d re-entrant honeycomb auxetic structures realized via additive manufacturing," *International Journal of Solids and Structures*, vol. 69, pp. 475–490, 2015.