Size-Driven All-Quad Meshing for Geographic Data

By

LANCE SIMONS
DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Nina Amenta, Chair

Zhaojun Bai

Julian Panetta

Committee in Charge

2024

Contents

Abstract

We present a full pipeline for producing all-quadrilateral, high-quality, low-singularity 2D surface meshes, that strictly adhere to input boundary geometry while being guided by a prescribed size function. The meshes are built for the simulation of groundwater flow, where input geometries are highly irregular due to following *e.g.* rivers and mountain ranges, but are sampled at somewhat regular intervals. The size of the produced elements are guided by a size function, coarsening or refining the mesh, allowing local control over the trade-off between simulation speed and accuracy.

The irregular boundaries set groundwater flow apart from typical quad mesh generation applications, where boundaries are typically smooth and able to be subdivided. Early testing with existing field-aligned methods showed significant drawbacks to using the boundaries directly, so we developed a tool to splice a non-conforming but reasonably-sized mesh into the fixed boundary, allowing arbitrary mesh generation techniques to be applied to the problem of groundwater flow.

This left the problem of generating size-driven meshes, where a minimal number of singularities is added to affect a change in size of the produced quads. Existing methods tended to focus on quads with uniform size, or with size determined by the input geometry. By treating the target size function as a Riemannian metric, we are able to exploit the curvature of the Levi-Civita connection to apply surface meshing techniques to our problem of size-aware planar meshing. We build off of the approach in *Globally Optimal Direction Fields* to first produce a cross field that approximates the desired growth, from which a quad mesh can be extracted.

When analyzing the resulting cross fields, we found it difficult to estimate how well they matched the desired size function without simply meshing them and measuring the resulting quads. Attempts to quantify the growth in size implied by a cross field led to an analysis of anisotropy, size functions that are not uniform in all directions. We were able to demonstrate that harmonic cross fields admit isotropic size functions, while general cross fields still admit orthogonal anisotropic size functions. This gives us a method to quantify how accurately a cross field matches the desired size function, independent of any size inaccuracy that arises from the meshing process.

Finally, we use a combination of existing techniques to extract a quad mesh from the cross field, which can then be stitched to the original boundary. We conclude with simple Laplacian smoothing as a post-processing step to improve the quality of the resulting meshes.

Acknowledgments

First and foremost, I'd like to thank my advisor Nina Amenta, who has been wonderful to work with. She has set the standard for the kind of researcher, mentor, and collaborator I want to be. Her intuition is unparalleled, and I know I'm on to something good on the rare occasions I'm able to surprise her. It often seems as if somehow *everyone* knows Nina, which is a testament to her policy of never passing up on an opportunity to make new friends.

Always go to the party.

I want to thank my dissertation committee members: Zhaojun Bai for believing in me in the 7 years (!) between my Qualifying Exam and graduation, and Julian Panetta for incredibly thorough and insightful feedback, and for keeping me honest about my math.

I will forever miss the Geometry Lab. Computational Geometry is a wide enough umbrella to have wildly disparate projects - my first memories of the lab were of the "GPU Graveyard" of graphics cards that couldn't keep up with Fatemeh Abbasinejad's GPGPU code, and of a row of plaster macaque skulls that lined the entryway of the lab - but still has enough common themes where everyone in the lab could jump in on a conversation about any of the ongoing projects. The lab's final cohort, Shengren Li, Stewart He, Carlos Rojas, and Alex Tsui, were great researchers, collaborators, and friends. After staying in the lab until around midnight working on a graphics project with Stewart, being transfixed by the pixelated shadows on the ground from the new-ish LED-grid outdoor lighting, and thinking that this was yet another shader glitch that we needed to fix, was the perfect (and recurring) amount of mental exhaustion. I never felt tired.

I will also miss the greater Academic Surge research community, or at least those that we could regularly steal on our pilgrimages to Black Bear Diner: Julia Matsieva, Roxana Bujack, and Jennifer Chandler. Between the game nights, movies, and willingness to appease Stewart's never-ending hunger for the largest burger in town, it was nice to get to be people outside of the lab from time to time too.

I am incredibly thankful to have met Nick Toothman, whose exile to the MoCap Lab meant he was out of assimilation range for the Geometry Lab mischief, but who ended up becoming one of my closest friends and biggest supporters. His positivity and enthusiasm were always welcome and contagious, whether it was moral support from afar or combating shared exhaustion at the end of a game jam. I look forward to many more projects together.

We don't apologize for victories here.

I especially need to thank John Owens, who took the time to answer my questions during an event for potential graduate students, and was instrumental in my decision to come to UC Davis in the first place, as well as Nina, who was willing to let me sit in on one of her classes.

I am deeply grateful for my parents, Bob and Laura, who were always supportive, even when the world was on fire.

Enough is as much as a feast.

Finally, I'd like to thank Can Dogrul and Tariq Kadir at the California Department of Water Resources, for funding, advice, datasets, and most importantly for inviting me to take the first step down this rabbit hole.

CHAPTER 1

Introduction

Groundwater refers to water *in* the ground (*e.g.*, in aquifers) as opposed to surface water (*e.g.*, in rivers and lakes). Computer simulations are used extensively in groundwater management. They fill in when physical measurement would be expensive and time consuming, like when comparing different management schemes [**WA95**], or even dangerous, such as when analyzing the potential spread of a contaminant [**CW84**]. They are often more reliable than scale models, as many effects do not scale linearly between the laboratory and reality.

These groundwater simulations rely on subdividing the region of interest into small cells, called *elements*, and establishing relationships between neighboring elements that will capture the behavior of the whole system. The geometry and connectivity of these elements is referred to as the *mesh*. Based on the physical characteristics of the material represented by an element, it will have different water storage and flow characteristics. So, regions composed of mostly solid rock will have low permeability, reducing flow between neighboring elements, forcing any flow to bypass the region via more-permeable materials. The goal of these simulations is typically finding steady-state flow, where the water flowing into each element matches the water leaving each element, except at elements specially marked as *sources* (such as along rivers) or *sinks* (natural springs or man-made pumps) with fixed incoming or outgoing flow.

The initial goal of this research was the production of all-quadrilateral meshes for *Finite Element Analysis* (FEA) simulations, as part of a project with the California Department of Water Resources. The inputs to the meshing problem consist of polyhedral boundaries of the regions to be meshed, along with interior features such as rivers, mountains, and wells. The datasets used are on a geographic scale, on the order of $10,000km^2$. Features must be approximated to prevent the meshes from being subdivided too finely. So, the features that make up large portions of the region boundary and internal features have already been simplified to a given scale; this scale should be honored by the resulting elements, as further subdivision will not result in recovering the accuracy lost from the initial simplification. Additionally, these meshes may interlock with other meshes along common borders, so it is important that some perimeter segments do not get subdivided, so that individual meshes can be combined along the common border.

Because FEA techniques are used across many disciplines, there are many meshing techniques that cater to specific inputs. Many of these assume a smooth boundary that can be subdivided arbitrarily, allowing the meshing tool to segment the boundary as appropriate. Some methods are specialized for the case where there is no boundary at all, as in the case of generating a 2D mesh on the surface of a 3D object. The jagged, zig-zagging boundaries of geographic datasets make it especially difficult to generate meshes that strictly match the boundary edges.

The accuracy of Finite Element Analysis is dependent on the size and shape of the individual elements. Smaller elements increase the accuracy of the analysis at the expense of increased computation cost, but the shape of the elements plays an important part of the accuracy of a simulation as well.

While there is an established collection of fast and robust algorithms for generating triangle meshes, quadrilateral meshing is still an area of active research. In addition to being able to use FEM solvers that operate on all-quad meshes, quad meshes offer more accurate interpolation than triangular elements for a given element area [D'A00], and we are hoping that the reduction in element count in comparison to a triangle mesh with the same target edge length will yield an increase in simulation speed. We must ensure that the quads we generate are of sufficient quality. Unfortunately, there are no quality metrics that universally guarantee accurate simulation results [Cha12]; different metrics are often used on a case-by-case basis. This is a difficult, long-standing issue; therefore, our goal is to develop meshing methods that are largely agnostic to specific quality definitions.

For a triangle mesh, the minimum angle within an element serves as a good indicator of quality. For quadrilaterals, the minimum angle is clearly not a good enough metric; a rectangle with ideal, 90-degree angles can still be arbitrarily skinny, and this poor aspect ratio can negatively impact simulation accuracy.

Another indication of mesh quality is the presence of *singularities*, often called *irregular vertices*, in the mesh. For a quad mesh, singularities are vertices with more, or less, than four incident edges. Singularities are a byproduct of changes in orientation or size of the elements in a quadrilateral mesh, and are commonly found along complex boundary regions and wherever elements scale in size. Singularities reduce the accuracy of linear FEM simulations, and increasing the order of the interpolation (by using quadratic instead of linear interpolation over an element in the mesh) often incurs a prohibitive increase in computational cost [CMP98]. Singularities induce problems with other types of meshing and simulation [VS17], so there is a rich body of literature around preventing or removing singularities in meshes.

Most meshing tools designed for FEA produce *graded meshes*, where the size of the elements scale with a desired sizing function over the mesh. This is a trade-off between accuracy and solution speed; large regions with relatively little change in function value can be approximated with larger elements, requiring fewer elements, resulting in a faster solve.

So, the fundamental problem we are trying to solve is generating an *all-quadrilateral* (every element is a quadrilateral) *high quality* (keep the minimum angle in the mesh as large as possible, while keeping aspect ratio within user-specified bounds) *low-singularity* (generate as few singularities as possible) mesh, that strictly adheres to the input geometry and any internal features, while trying to ensure that the sizes of the generated quads follow a prescribed size function (allowing for high-density regions where high accuracy is needed, and low-density regions where it is not).

The premier software for groundwater simulation is GMS, the Groundwater Modeling System. This is a proprietary software package purpose-built for groundwater simulations. GMS handles both FEA and FDA (*Finite Difference Analysis*, a simpler formulation particularly suited to regular grids) analysis, supports automatic mesh generation, and runs the simulations. Their automatic mesh generator produces graded triangulations, and uses *triangle matching* to join adjacent triangles into quadrilaterals if requested. This matching process means that an all-quadrilateral mesh cannot be guaranteed, so only solvers that support mixed meshes can be used. GMS also implements an approach called *paving* to produce guaranteed all-quad meshes, but that approach is unable to honor internal boundaries within the mesh. [**OJH96**]

There is an open-source software package called Gmsh that is designed to generate meshes specifically for FEA simulations. Like GMS, Gmsh produces triangle meshes and then uses a matching step when quadrilateral meshes are desired. Gmsh provides guaranteed all-quad meshes by offering a Catmull-Clark subdivision step, which divides all triangles in the mesh into three quads, and all quads into four quads (see Figure 1.4). This process induces many singularities in the resulting mesh. [**GR09**]

The DWR's interactive IWFM-MG software tool generates triangle meshes, using the high-quality generic 2D meshing program Triangle [**She96**]. Both Triangle and Gmsh are recognized for producing high-quality meshes. Figure 1.1 shows a section of a typical DWR dataset, as meshed by Triangle and Gmsh. The difficulty that both programs face here demonstrates how difficult these datasets can be to mesh. While Gmsh does a good job in the interior, the complexity of the boundary and inner features cause a drastic reduction in quality. The thin region along the left edge where the triangle matching breaks down is of particular interest, as well as the presence of many very-obtuse angles, such as at the small bump on the upper-right perimeter.

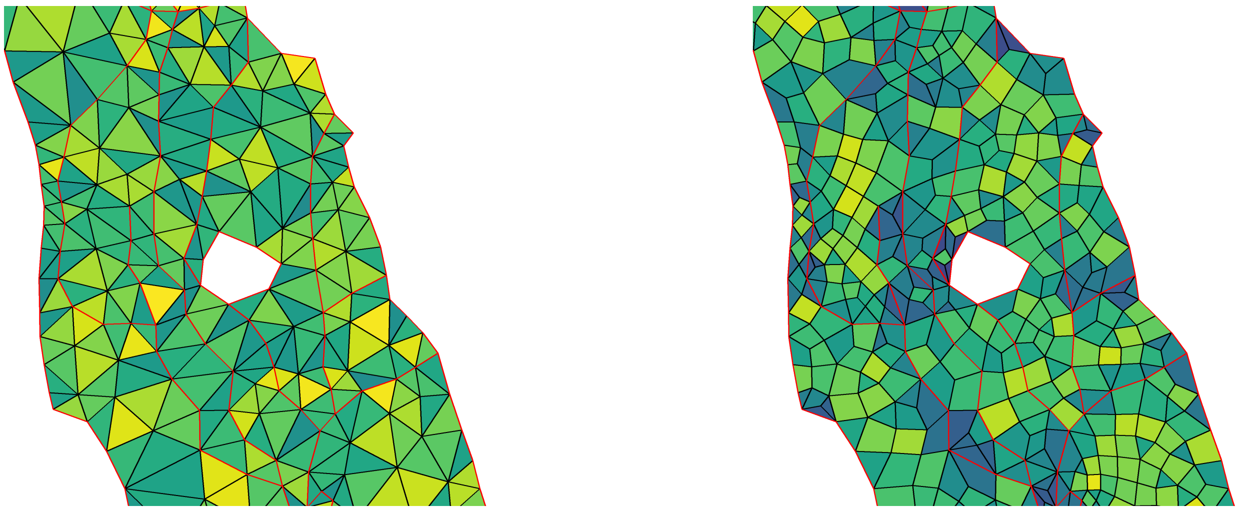

(a) All-triangle mesh from Triangle (b) Mixed mesh from Gmsh

FIGURE 1.1. A slice from California's Central Valley, with boundaries and internal features shown in red. Elements are colored according to the angle that deviates furthest from ideal, 60° and 90°, respectively, where brighter elements have better angles. It is important to note the areas where the complex internal features cause Gmsh to produce worse elements.

1.1. Background Concepts

1.1.1. Triangulation and Quadrangulation. *Triangulation* is the process of subdividing a polygon into triangular faces. The most common approach is called *Delaunay triangulation*. The Delaunay criterion mandates that, for each triangle in the mesh, no vertices in the mesh lie inside its circumcircle. Enforcing this constraint maximizes the minimum angle in the resulting mesh; the smallest angle in any other triangulation of the same polygon will meet or exceed the smallest angle in the Delaunay triangulation [**She02**].

Mesh *refinement* is the process of inserting additional vertices into the mesh, which are referred to as Steiner vertices. These vertices are added to split existing triangles into smaller triangles of better quality. This process is repeated until all triangles in the mesh meet a minimum quality criterion. This is not always possible; poor input meshes or specifying too high of a quality bound can cause refinement algorithms to subdivide indefinitely [**She02**].

Many forms of quadrilateral mesh generation begin with an existing triangle mesh. Some algorithms generate this mesh themselves, while others accept it as input. This mesh is a *background mesh* if it is used only for the representation or calculation of some value over the region. In other cases, the mesh will undergo transformations that lead to the final quad mesh. Typically, the mesh will be a triangulation computed via Delaunay refinement, in order to provide accuracy for calculations or for better triangle quality which will hopefully yield better output quadrilaterals.

In order for a polygon to be divided into quadrilaterals, it must have an even number of edges along its boundary. This follows directly from the *Euler characteristic*, $|V| - |E| + |F|$ which relates the number of vertices, edges and faces in a mesh, and equals 2 for a planar graph. Every edge is either incident to two faces or lies along the boundary, so $4|F| = 2|E| - |B|$. Multiplying the Euler characteristic equation by 2 yields $2|V| - 2|E| + 2|F| = 4$. Solving for $2|E|$ and substituting results in $2|V| - 2|F| - |B| = 4$. Solving for $|B|$ gives $|B| = 2(|V| - |F| - 2)$, showing that the length of the boundary must be even.

1.1.2. Mesh Quality. While the relationship of mesh quality to simulation accuracy is a difficult question, the quality of a mesh interpolation of function values given at the vertices is relatively straight-forward.

The *accuracy* of a mesh element is the measure of the relative error between the value produced by interpolating values calculated at the vertices over the element and the actual value of the function being estimated over the surface. Reducing the size of the element will typically increase its accuracy, by decreasing the distance between the interpolant and sample points; subdividing each element in the mesh into n elements should reduce the approximation error by a factor of $1/n$, for the first-order elements we are using [D'A00].

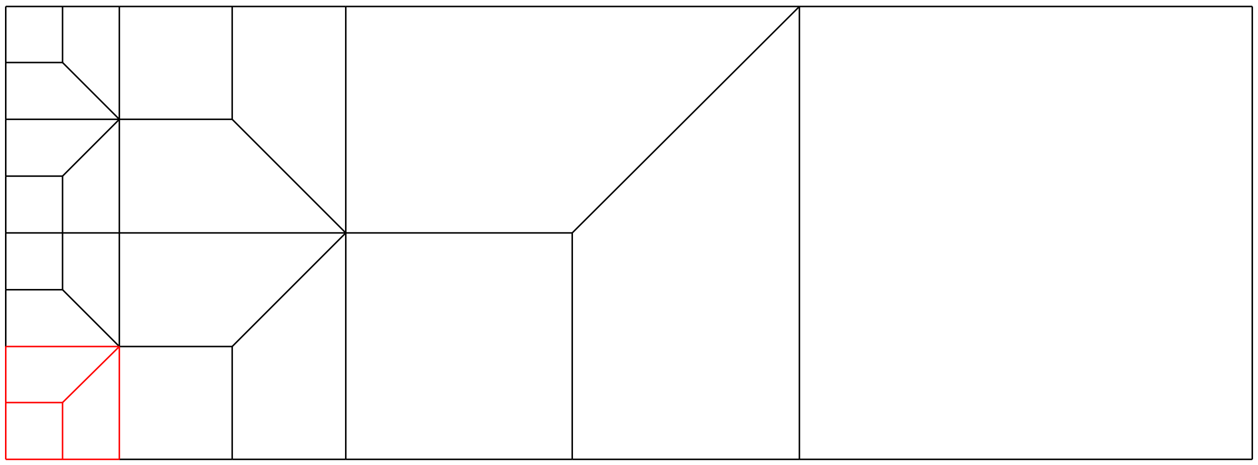


FIGURE 1.2. A common pattern seen when quad meshing quadtrees without T-junctions.

While area is a natural choice for metric when discussing element sizes, one-dimensional length is used to maintain consistency when comparing edges and elements. Because this metric will change over the region, it is called the *sizing function*. Figure 1.2 shows a common growth pattern for increasing element size, which is self-similar and shows a growth factor of doubling the edge length over every two edges along the bottom of the image. This demonstrates that the growth in the metric over some distance can be directly proportional to that distance while maintaining element quality.

The *efficiency* of a mesh element is the accuracy of the element divided by its area. This gives a metric for mesh quality estimation relatively independent of resolution. For example, a very long, thin triangle may not be accurate, but can be made more accurate by subdividing it. The resulting triangles will still not be efficient, however, if their shapes do not improve on that of the original triangle. Quadrilateral elements have been shown to have better efficiency than triangular elements [**D'A00**].

These are the two main driving forces behind mesh refinement; regions may be refined to improve element quality (increasing efficiency) and to reduce their size (increasing accuracy). We are attempting to subdivide the mesh as little as possible, so we must ensure that we produce highly efficient elements.

Common quality measures for planar quadrilaterals include minimum angle, maximum angle, length ratio (or aspect ratio), stretch (ratio between the longest diagonal and shortest edge), and skew (minimum angle between the two edges formed by connecting the midpoints of opposing edges).

A popular quality metric for a quadrilateral (or a hexahedron, in three dimensions) is the determinant of the Jacobian matrix, sometimes confusingly just called the Jacobian. The Jacobian matrix is the linear approximation of the nonlinear mapping between local coordinates of the quadrilateral to local coordinates on an ideal quadrilateral, usually an axis-aligned square with side length 2, centered at the origin. The Jacobian matrix encodes the orientation, scale, skew, and aspect ratio of the quadrilateral, and the determinant reduces that to a single useful value. Typically, only the skew and aspect ratio (together referred to as 'shape') are desired, as the orientation and scale of the mesh elements are not considered mesh quality metrics. In these cases, the orientation and scale values are factored out, giving a determinant in the range [0, 1], referred to as the normalized Jacobian (determinant). As the Jacobian determinant will vary over the surface of the quadrilateral, it is calculated at predefined sample coordinates within the quadrilateral and the worst Jacobian determinant is reported as the quality measure. [**Knu01**]

1.1.3. Cross Fields and Singularities. The topology of quad meshes gives rise to some useful constructs. A good quad mesh will have most vertices of degree 4. At vertices of degree 4, the incident edges can be joined into paths, so that opposite edges form two crossing groups of paths. Borrowing terminology from animation, these are called *line loops* and are defined to begin and end at either the perimeter of the surface or at *poles*, or *singular vertices*, which are vertices with a degree other than 4 (typically 3, 5 or 6). A *chord*, sometimes called a *quadrilateral strip*, is the set of elements sandwiched between two neighboring parallel line loops [**DSC09**].

A continuous analog of a quad mesh is a *cross field*, that defines four vectors at every point on a surface. These four vectors exhibit $\frac{\pi}{2}$ *rotational symmetry*, meaning a vector rotated by $i\frac{\pi}{2}$ for any integer i is equivalent to the original vector, similar to the 2π rotational symmetry in the Euclidean plane. At *singularities*, or *singular points*, the direction vectors are not defined or are not smooth. Away from singularities, the four orthogonal vector fields that make up a cross field are smooth.

The singularities in cross fields correspond to the singular vertices that terminate line loops. Dealing with these points is a critical problem in quad meshing; three major approaches to generating quad meshes are effectively heuristics as to where singular points should be placed [**AFTR15**]. Paving and medial axis approaches attempt to restrict singular points to the center of the region, while cross field approaches try to find an optimal layout over the entire surface. As seen in Figure 1.2, a change in sizing often requires many singular vertices, so our goal is strike a balance between size growth and singularity count; our complex boundaries will induce singularities anyway, so reducing singularities within the mesh interior should keep the overall singularity count as low as possible.

A *streamline* is a path that is always tangent to a local vector or cross field. Therefore, every regular point of a cross field lies on two perpendicular streamlines, just like every regular vertex in a quadrangulation lies on two line loops; another way to look at this is that every regular point has four outgoing streamlines, and every regular vertex has four outgoing edges. Just as an extraordinary vertex has more or fewer outgoing edges, a singular point will have more or less than four outgoing streamlines, which are called *separatrices*. Each *separatrix* begins and ends at a singularity or at a point along the boundary. The separatrices divide the domain into singularity-free regions whose interiors are topological rectangles [**KLF15**], which we call *simple tiles*. Simple tiles are not necessarily suitable as quadrilaterals on their own, but are often used as a coarse initial mesh, within which some subdivision of each region is chosen to produce high-quality quads. The subdivision process produces only quadrilateral elements, with the only singularities in the mesh along the perimeters of the simple tiles, although we will also explore cases where additional singularities can be used to improve the resulting meshes.

1.2. Overview

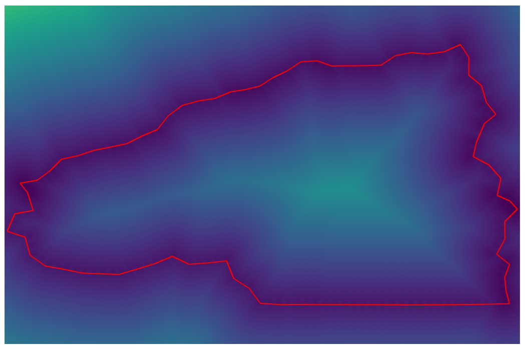
Groundwater flow is not a typical application for quad meshing, and in investigating existing approaches we encountered many factors that differed from typical problem domains. Our regions of interest are 2D with complex boundaries, unlike many approaches which operate on a manifold embedded in 3D, often having no boundaries at all. While reducing the singularity count and maximizing quality metrics for the resulting quads are common goals, we differ in that our primary constraint is element *size* instead of element *alignment*.

The irregularity of the input perimeters, often tracing geological features like mountain ranges or rivers, make alignment with the input edges especially difficult. Our intuition was that that an "ideal" mesh in *any* sense, whether a repeated template like in Figure 1.2 or simply a regular grid, is going to be ill-suited to matching the boundaries given. This led us to create an approach where we can take an arbitrary "ideal" mesh that does *not* match the boundaries and mate it to the boundaries in a way that preserves as much of the interior region as possible. This allows us to generate a mesh while focusing only on the size constraints, and then post-process it to ensure the result matches the boundaries.

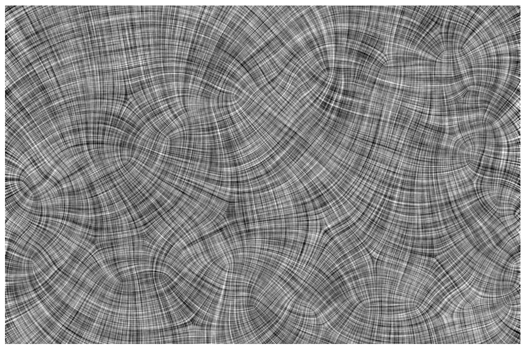
Instead of trying to generate a quad mesh directly from the target size function, we noticed that many state-of-the-art meshing techniques start by generating a cross field, from which a quad mesh can be extracted. A common trend is for the cross field to be optimized to fit alignment constraints, in particular the principal curvature directions for manifolds embedded in 3D. With the insight that a changing metric induces curved geodesics just like on a curved surface, we were able to generate cross fields using an existing method by aligning with the metric's curvature instead of the surface's.

However, even in our 2D case without boundary conditions, we found that field generation is harder than expected; the sizes implied by our generated cross fields often differed from our target size functions in unexpected ways, even appearing to be a scalar function with a non-conservative gradient, which is impossible! This led us to develop visualization tools and perform analysis that led to theoretical insights into the relationship between cross fields and size functions, with a critical result being that many cross fields do not have compatible scalar size functions and necessarily correspond to anisotropic metrics over the domain.

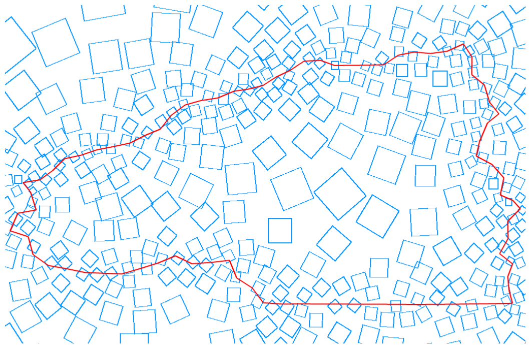
With this better understanding of the relationship between cross fields and their size functions, we still needed to extract a quad mesh from our generated cross field. Here, we were able to take advantage of existing techniques, pulling inspiration from several approaches to generate all-quad meshes.



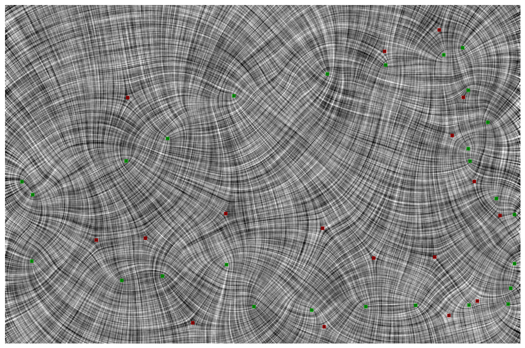
(a) Region boundary and computed size function



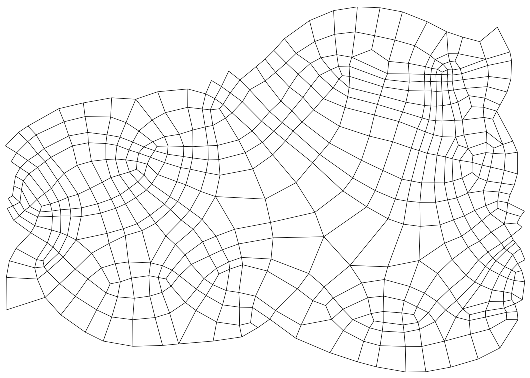
(b) Direction field as computed in Chapter 3



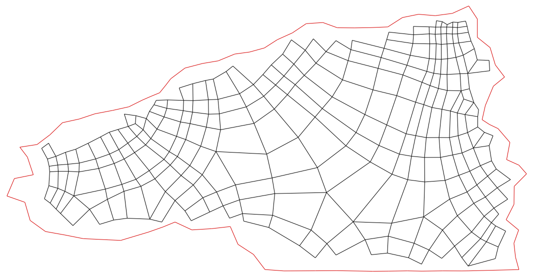
(c) Visualization of size and orientation



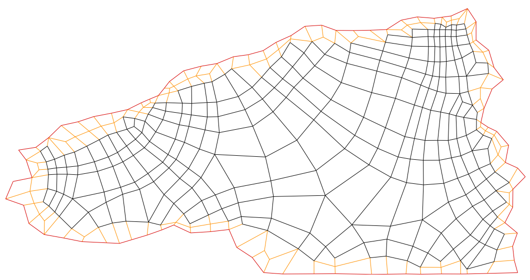
(d) Singularities detected in direction field



(e) Extracted mesh



(f) Removed quads near or outside original boundary



(g) Region stitched as described in Chapter 2

FIGURE 1.3. Overview of the meshing pipeline

1.2.1. The Meshing Pipeline. We propose the following pipeline for mesh generation. The inputs to this process are the perimeters of the region to be meshed, expressed as closed loops of connected line segments, as well as a desired size function σ, which defines a target edge length at every point in the region. Some of the size functions we will look at take the value of the edge lengths at the perimeters, and grow as some function of distance to the perimeters, where the growth rate is a tradeoff between mesh grading and singularity count; this can be seen in Figure 1.3a. The more aggressively the size function changes, the more singularities will be induced in the interior of the mesh.

This mesh generation process is split into boundary and interior phases. Chapter 2 covers our approach from *All-Quad Meshing for Geographic Data via Templated Boundary Optimization* [**SA17**] that allows us to take an interior mesh and stitch it to the original boundary, allowing the mesh generation to focus on size control and element quality. We accomplish this by carving away any quads from the interior mesh that are outside of the region or are close to the original perimeter. This creates an empty channel between the mesh and the outer perimeter that we fill using an optimization process described in Section 2.2.8. This introduces many singular vertices, but due to the complexity of most boundary regions, this is typically unavoidable.

We then split the interior mesh generation into two steps: generating a *cross field* compatible with the input size function and then extracting an all-quad mesh from the cross field.

In Figure 1.3a, the target size function increases linearly with the distance from the perimeter. This size field is the only input to the process described in Chapter 3, which produces a cross field as shown in Figure 1.3b. When taken together, these two fields define a size and orientation at every point, visualized in Figure 1.3c by drawing a quad of the desired alignment and size at sampled points. Our field generation step is based on the work of Knoppel *et al.* in *Globally Optimal Direction Fields* [**KCPS13**]. This is the focus of Chapter 3, where we spend a significant amount of time delving into the mathematical underpinnings of their approach.

Chapter 4 explores the connection between cross fields and size functions, including simpler derivations of some of the results from Chapter 3. In particular, we address *anisotropy*, where the size function implied by a cross field is actually two separate scalar fields, each representing length aligned with a different pair of opposed vectors in the cross field. Each cross field implies a *family* of compatible size functions, and we optimize within this space to find the size function that minimizes anisotropy. We published this chapter as *Anisotropy and Cross Fields* [**SA**], although we have added some supplementary datasets here.

Finally, Chapter 5 covers our approach for extracting a quad mesh from the cross field, based off of the approach from Campen *et al.* in *Quantized Global Parameterization* [CBK15]. It is straightforward to create an all-quad mesh such that the only singular vertices in the mesh are the singularities in the cross field by tracing out the separatrices from each singularity. However, it is difficult to ensure that the resulting mesh also obeys the target size function. QGP uses a generalization of the separatrix graph to produce a subdivision of the graph that more closely follows the target size function. Once a parameterization has been found with QGP, we use *Coons patches* [Coo67] to produce quads, which then serve as the interior mesh for the approach outlined in Chapter 2.

Much of our focus is on the underlying mathematics and techniques, aiming for understanding of the concepts over producing an optimized software product. As raw performance has not been a primary concern, we have implemented all of these techniques in Python. We will explicitly highlight exceptions, such as our C implementation of the computationally-expensive patch ranking process from Chapter 2, when we leverage the optimized solvers from SciPy [VGO+20], and when we include functions provided by authors of a paper as we do with *Globally Optimal Direction Fields*.

1.3. Prior Work in Quad Meshing

1.3.1. Indirect Quad Mesh Generation. Indirect mesh generation methods start with an existing mesh, which is transformed into the final quadrilateral mesh.

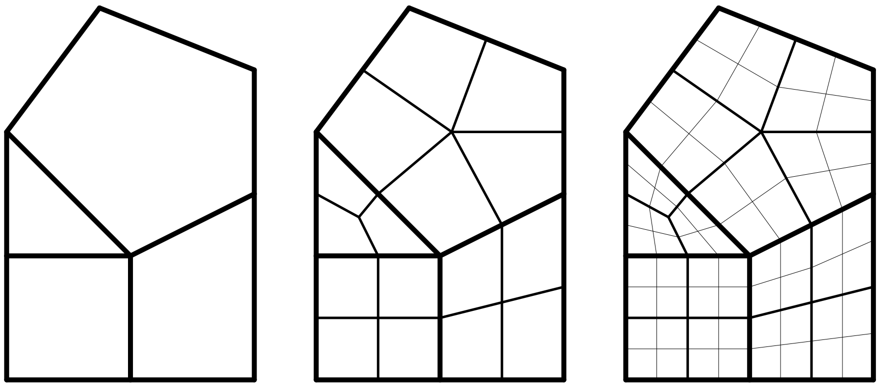


FIGURE 1.4. Two iterations of Catmull-Clark subdivision, with thinner edges for each successive division. Note that the first iteration creates extraordinary vertices for any non-quadrilateral face.

1.3.1.1. *Catmull-Clark Subdivision.* Catmull-Clark subdivision [CC78] is a generalization of bicubic B-spline patch subdivision. While initially created to smooth surfaces in 3D, it is also used extensively in two dimensions to produce a quad mesh from a mixed mesh. Each face generates a new vertex at the average of its incident vertices, and each edge generates a new vertex at its midpoint. Each new face vertex is connected by an edge to each new edge vertex. So, regardless of input geometry, the mesh has been reduced to a set of quadrilaterals, each composed of one face vertex, two edge vertices, and one vertex from the previous generation of the mesh. This process is repeated until the desired element size is achieved. Unfortunately, the face vertex created by any non-square face will become a singularity, and any vertices that started with valence $\neq 4$ will remain as singularities. As the result of an iteration is all quadrilaterals, these singularities are only created during the first iteration, but they persist in all subsequent iterations. Often only a single iteration of the operation is performed in order to produce guaranteed all-quadrilateral meshes; this is a common post-processing step for many approaches.

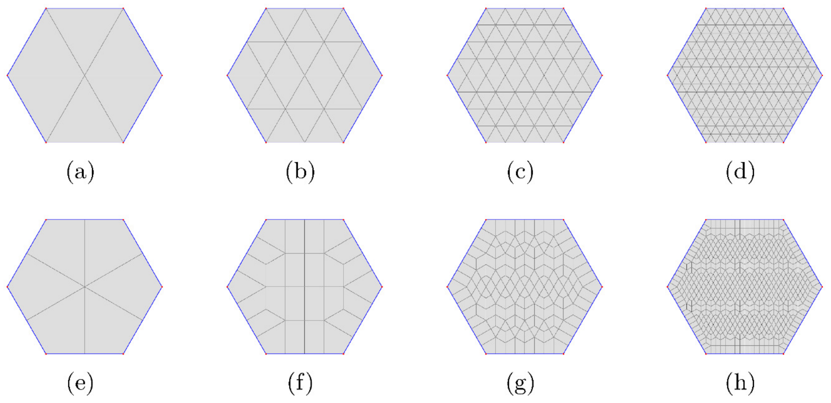


FIGURE 1.5. Q-TRAN uses matching and local topological operations to guarantee an all-quadrilateral output mesh (bottom row) from a triangular input mesh (top row). The resulting meshes have many singularities, although fewer than an iteration of Catmull-Clark subdivision would produce from the given input mesh. Image is Figure 4 in [**EKMD10**]

1.3.1.2. *Triangle Matching.* A common approach for converting a triangle mesh into a quad mesh is known as *matching*, where two triangles that share an edge are merged into a single quadrilateral. The Blossom algorithm [**Edm65**] can be used to find a *perfect matching*, where every triangle is matched to a neighbor and the resulting mesh is exclusively quadrilaterals. However, there is no guarantee that a perfect matching exists, so matching algorithms often add an iteration of Catmull-Clark subdivision if an all-quad mesh is necessary [**RLS$^+$12**]. Even if a perfect matching is able to produce an all-quad mesh, any vertex whose valence is not reduced to exactly four (ideal triangle mesh vertices have valence 6) becomes a singularity. Even approaches that do not depend on Catmull-Clark to guarantee all-quad meshes, such as Q-TRAN [**EKMD10**], typically generate many singularities; see Figure 1.5.

1.3.2. Direct Quad Mesh Generation.

1.3.2.1. *Paving.* One approach to direct quadrilateral mesh generation that is successful in many applications is called *paving*, introduced by Blacker and Stephenson [BS91]. Their approach is to start at the perimeter of a region to be meshed, adding a single row of quads based on the local shape of the boundary. The resulting hole is recursively meshed, with terminal cases for a perimeter of length 6 or less. Angles along the perimeter determine how quadrilaterals are added to form the row, but a concave boundary might result in quads along different sides of the perimeter overlapping one another, requiring special cleanup operations. The resulting meshes typically have high-quality perimeters, with singularities and lower-quality elements relegated to the deep interior of the mesh.

While very useful for following the alignment of smooth boundaries, paving is not appropriate for our inputs. Our difficult boundaries inevitably produce many singularities and low-quality elements near the boundary, and paving additionally generates these in the interior.

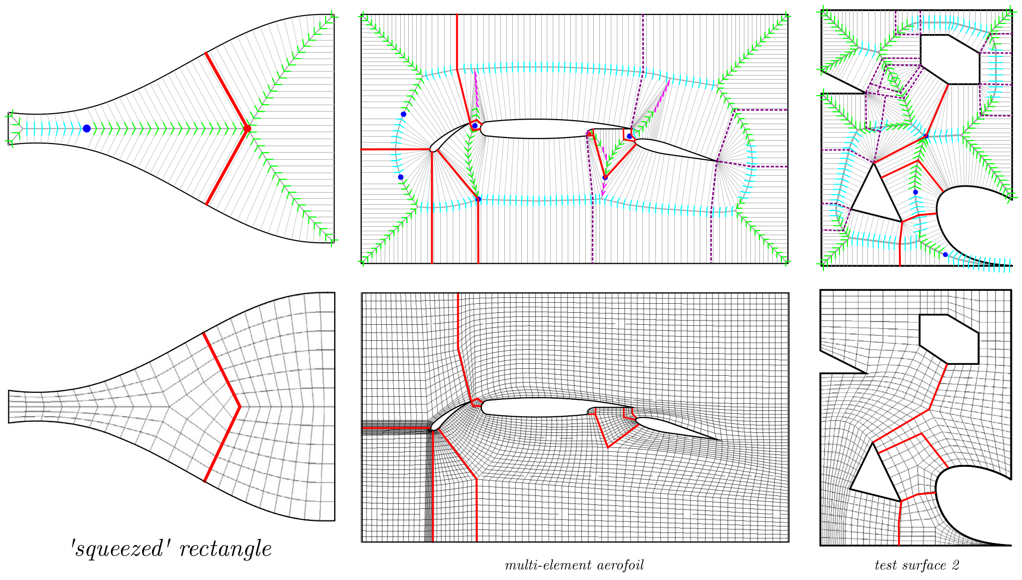


FIGURE 1.6. The approach in [FAR16] is based on the medial axis of the region and optimizes the number and placement of singularities at the expense of nonuniform element sizes. On top, lines perpendicular to the perimeter are traced inward to the medial axis, a subset of which is used to subdivide the region into meshable tiles, with singularities marked in blue. This then gets meshed (bottom), with element sizes heavily influenced by distance between the perimeter and medial axis. Images are from Figure 25 in [FAR16].

1.3.2.2. *Medial Axis.* In an attempt to predict where these poor interactions will occur, the *medial axis transform* can be used to predict where opposite sides of the boundary will eventually meet [QRPG04, FAR16]. The medial axis of a shape is the set of all points that have two or more closest points along the boundary, or analogously, the set of all points where the distance field is not smooth. This approach cleans up the regions where paving would introduce poor-quality elements and/or singularities, at the expense of forcing opposed elements to align to the edge in the medial axis that separates them. This approach also forces elements to scale with the distance between perimeter and medial axis, as seen in Figure 1.6.

Again, this approach would introduce a great deal of unnecessary complexity on our inputs, where the medial axis is as complex as the boundary. Like paving, the ability for medial axis methods to produce boundary-aligned elements is not helpful for our case, and they do not lend themselves to matching a given size function.

1.3.2.3. *Cross Fields.* Our approach belongs to a collection of methods that produce quad meshes by first solving for a cross field over the surface. Interestingly, the formulation of (and solution to) the cross field generation problem is very similar to the groundwater flow itself, as it is typically solved as a finite-element problem over a triangular background mesh. Iterative methods tend to be preferred here; the biconjugate gradient method is used by [**PZ07**], and Gauss-Seidel iterations are used by [**JTPSH15**]. These will produce a cross vector at each vertex in the background triangle mesh, so any singularities in the field will lie inside the background mesh elements.

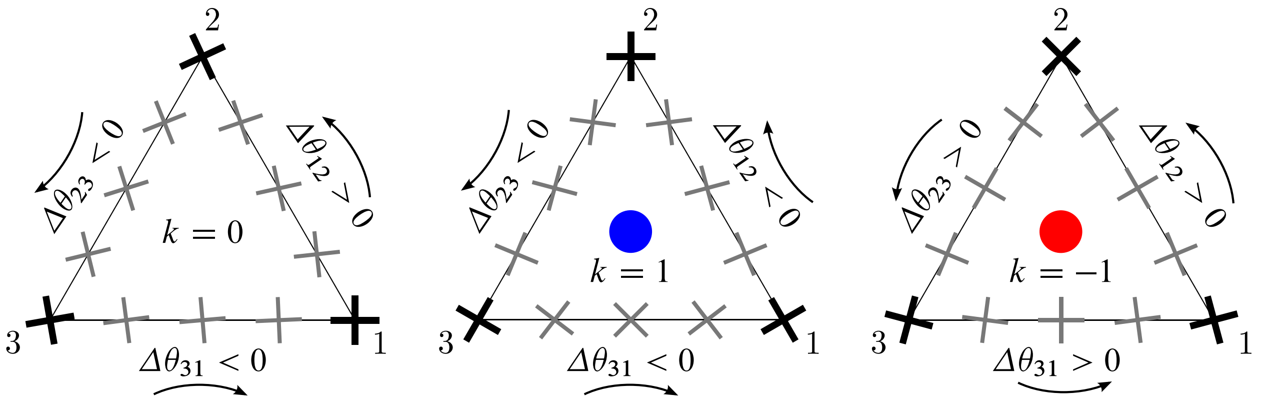


FIGURE 1.7. Singularity detection in triangles, from Figure 10 in [**FAR15**]. No interior singularity results in zero total rotation (left). Net singularity count can be detected by dividing total rotation by $\pi/2$, with the sign indicating positive (center) or negative (right) singularities.

An element that contains a singularity can be found by following a path along the perimeter of the element and measuring the rotation of the cross field alignment along the way. The sum of the rotations of the cross field will indicate the presence of singularities; see Figure 1.7 for an example over a triangle. This can be done for any closed region but only detects the sum of all singularities in that region; two valence-5 singularities are indistinguishable from a single valence-6 singularity, just as valence-3 and valence-5 singularities will cancel out and no singularity will be detected. Thus, the element size of the background mesh plays a crucial part in the construction of the cross field; a coarse background mesh will force singularities to merge together.

Extracting a quality quad mesh from a cross field has proved to be surprisingly difficult. There are several ways to create a quad mesh from a cross field; this process is called *extraction* in the literature. In [**JTPSH15**], a second solve is performed, creating what they call a *position field*, encoding the relative position of each vertex in the background mesh within its quad in the final mesh; in effect, each background vertex produces four potential vertices for the output mesh, which are clustered and merged in order to form the actual final output vertices. The other common approach is to generate an *integer-grid map*, which is a parameterization that maps the surface to $\mathbb{R}^2$ such that the position field is axis-aligned after the mapping. Then, in the inverse map, points in $\mathbb{Z}^2$ map back onto the surface as the final vertices. However, this approach is susceptible to errors in the integer grid-map accuracy, and is highly dependent on machine precision, and thus requires considerable cleanup post-processing to ensure quality output [**EBCK13**].

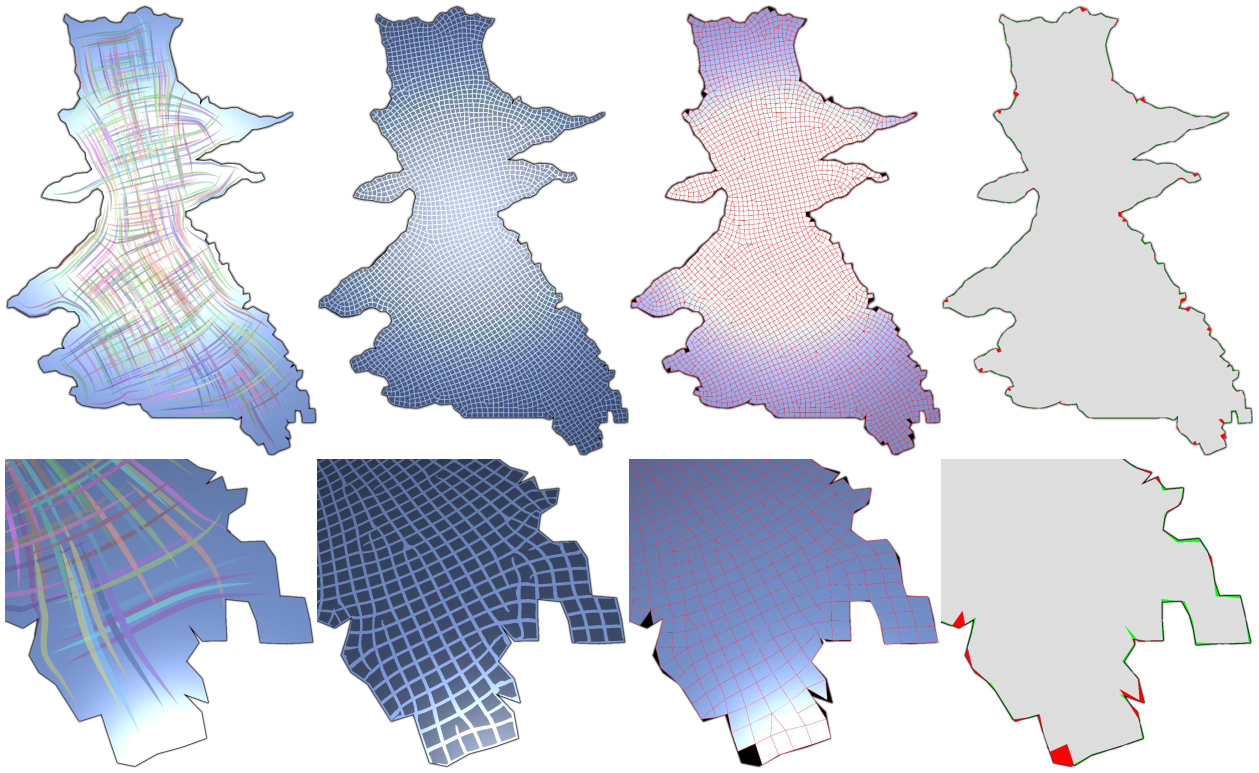


FIGURE 1.8. The phases in *Instant Field-Aligned Meshes* [JTPSH15] when run on the Sacramento Valley dataset. First, a cross field is computed that generally aligns with the region boundaries, then a local parameterization step finds vertex positions. Finally, the geometry is extracted. The final frame shows the deviation of the final mesh from the initial region boundary, contraction in red and expansion in green.

These methods are currently very popular in the geometric modeling community, and we experimented with using them directly on our input. While these methods generally produce very high quality elements, they break down in situations like ours in which we have complicated outer boundaries. For example, in *Instant Field-Aligned Meshes* [JTPSH15], most elements are nearly-perfect squares. However, throughout the mesh, there are triangles and a few pentagons, and even non-convex elements, and more importantly for us, the perimeter fails to follow the original input shape, as seen in Figure 1.8.

We found that this method did not handle varying size functions well, although we did use it for a boundary-aligned fixed-scale interior mesh in Chapter 2.

1.3.2.4. *Frame Fields.* A generalization of the cross field is the *frame field*, which relaxes the orthogonality and length constraints of a cross field to form a field comprised of the four vectors $v, w, -v$, and $-w$. They were used in [PPTSH14] to produce surface meshes that match a set of sparse orientation and size constraints, which sounds like an ideal candidate for our conditions. However, reported instability in cases with size changes of more than a factor of 10 in [PPTSH14] and the potential reduction in element quality from non-orthogonal frame fields (forcing skewed elements) in general, were significant enough drawbacks for us to focus on approaches based on cross fields instead.

Interestingly, Chapter 4 is dedicated to anisotropic sizing, and as we will discuss in Chapter 5, even with an orthogonal cross field, the meshing process is likely to induce some skew in the resulting quads.

1.3.2.5. *Spectral Methods.* There is an interesting approach used to generate all-quad meshes on closed surfaces called *spectral quadrangulation*. They calculate a harmonic function over the surface, and then use its *Morse-Smale Complex* to extract a mesh. The Morse-Smale Complex places a vertex at the location of all minima, maxima, and saddles in the function and connects saddles to their neighboring extrema with edges. This creates quadrilateral faces, each composed of one maximum, one minimum, and two saddles. The original work [DBG+06] focused on creating subdivision surfaces, but other work [LHJ+14] used higher-frequency functions to build a quad mesh directly. This method naturally forms singularities at maxima and minima in the standing wave.

Unfortunately, this approach has difficulties along mesh boundaries. While [LHJ+14] discusses boundary curve alignment, existing methods currently do not respect boundary vertices. This could be solved with either an integer programming step assigning each vertex as a minimum, maximum, or saddle in the function, which is computationally infeasible, or requires a subdivision step analogous to Catmull-Clark subdivision, which we are trying to avoid.

1.3.2.6. *Grid-Based Methods.* Grid-based methods fill the interior of the region with a regular structure, and then connect the outer perimeter (or shell) with the internal structure. These approaches are typically template-based, for the internal and/or external elements. In two dimensions, the internal structure is typically a grid, potentially with templates that allow a quadtree subdivision of the interior if a graded mesh is desired [EDF10, Sch96]. In 3D, the internal structure may be a body-centered cubic lattice [LS07], or a lattice of acute tetrahedra that may offer better angle bounds [DCB13], with a similar templated approach to using an octree for internal grading.

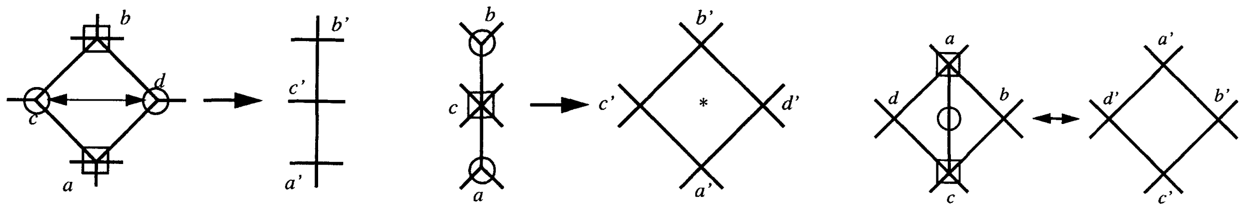
This is useful for domains with complex boundaries. For smooth but complex boundaries in $\mathbb{R}^3$, [LS07, DCB13] give the best existing tetrahedral mesh quality bounds. They give strict bounds on resulting angles, $[10.7°, 164.8°]$ or $[8.9°, 158.8°]$ depending on the settings used.

Our approach is similar to the methods used in [Sch96] and in [EDF10], where the internal structure is overlaid onto the domain, and any elements that cross over a boundary or feature edge are removed. This results in an unmeshed *channel* that must be filled. Both of these prior approaches connect edges from vertices on the interior region to the perimeter, such that these edges form a division of the channel into quads. Our approach, greatly simplified, could be described as the exhaustive search to find a subdivision of the channel into quads when considering a set of possible edges across the channel that each vertex could be incident to, seeking to produce the the highest-quality perimeter.

1.3.3. Mesh Improvement.

In addition to generating the mesh, there are many methods of improving a mesh after it has been generated. One class of methods, *smoothing*, involves repositioning vertices to improve element quality, while maintaining the connectivity of the mesh. We also consider *local improvement* methods that change the connectivity of small groups of elements in the hopes of improving the mesh overall. Here we consider improvement methods for all-quad meshes.

One of the most common forms of smoothing is *Laplacian smoothing*, the goal of which is to have each vertex placed at the centroid of its neighbors. It is common in practice because it is easy to implement and produces generally good results for many metrics; unfortunately, as it is not directly tied to any metric, it will not be as effective as a smoothing process that is designed specifically to optimize a given metric [ABE99].



(a) Element Collapse (b) Element Open (c) Doublet Insertion and Removal

FIGURE 1.9. Topological improvement operations, from Figures 3, 4, and 6 in [CMP98]. Extraordinary vertices are marked with a square or circle depending on degree. Doublet insertion will be followed up by another operation that will remove one of the two degenerate quadrilaterals.

There are topological methods [ABO09, VT11, CMP98, VS17] that seek to improve the connectivity of the mesh, reducing the number of singularities while maintaining, or even improving, the mesh resolution and quality.

For example, the *element collapse* operation, shown in Figure 1.9a, removes an element from the mesh by merging two vertices. The degree of the vertex resulting from the merge is two less than the sum of the original vertices degrees; if both were degree-3 singularities, then the resulting vertex is a non-singular degree-4. The other two vertices have their degree reduced by one. Element collapse and element open are the basic operations from which other operations are composed; while *doublet removal* (see Figure 1.9) is useful to reason about, it is a special case of element collapse [**CMP98**]. In the preliminary stages of this work, we added an element collapse cleanup phase of the Department of Water Resources IWFM-MG plugin as it was an effective way of removing quadrilaterals with two very large angles without resorting to cutting them into triangles. While these operations may be useful for singularity reduction for our approach as well, we focus on limiting the number of singularities we produce.

1.4. Contributions

While there were many existing techniques that we were able to leverage in order to build our meshing pipeline, the particular requirements of groundwater flow led us to the following contributions:

- A template-based optimization method that fills thin polygon boundaries with quadrilaterals, in such a way that maximizes the lowest-quality element. We support arbitrary quality metrics, but our implementation follows the standard max-angle metric.
- The insight that a non-uniform size function on a surface distorts the path of geodesics similarly to how they would be distorted if the surface was curved. This allows us to apply the process from *Globally Optimal Direction Fields* [**KCPS13**], which was designed to generate a cross field that closely follows surface curvature, but in our case generates a cross field that closely follows our sizing field.
- An analysis of the mathematics used in *Globally Optimal Direction Fields* [**KCPS13**], with derivations of many of the core concepts, with the hope that a thorough breakdown of the technique the may make the material accessible to a more general audience.
- An analysis of the relationship between cross fields and compatible size functions, in particular how the gradient of a cross field relates to the gradient of compatible size functions, including a significantly simpler derivation of that relationship compared to the approaches taken in other works.
- Demonstrating that a cross field must be harmonic to admit an isotropic metric, and that non-harmonic cross fields are still able to produce orthogonal (skew-free) anisotropic metrics.
- The observation that a pointwise rotation of a cross field, achieved by adding some uniform value (the *phase*) to the angle of the cross at every point, has an impact on the compatible size functions.

This is not encountered in typical contexts, where field alignment is prescribed so phase is not considered, and only has an effect on anisotropic size functions.

- A combination of existing techniques to form a more robust method of extracting an all-quad mesh from a cross field; in particular, allowing the addition of additional singularities as seen in *Automatic generation of multiblock decompositions of surfaces* [**FAR15**] to improve the parameterization of *Quantized Global Parameterization* [**CBK15**], and using *Coons patches* from *Surfaces for Computer-Aided Design of Space Forms* [**Coo67**] to robustly extract a quad mesh directly from the quantization produced by QGP.

CHAPTER 2

Perimeters and Constraints

2.1. Introduction

The following is a novel algorithm for all-quad meshing. As complicated boundary geometry and input features are the source of most of the problems with existing methods, we work to separate the boundary and features from open regions that are easier to mesh. We can use a variety of methods to mesh these open regions; a regular grid works well in many cases, but we propose another method in Chapter 3.

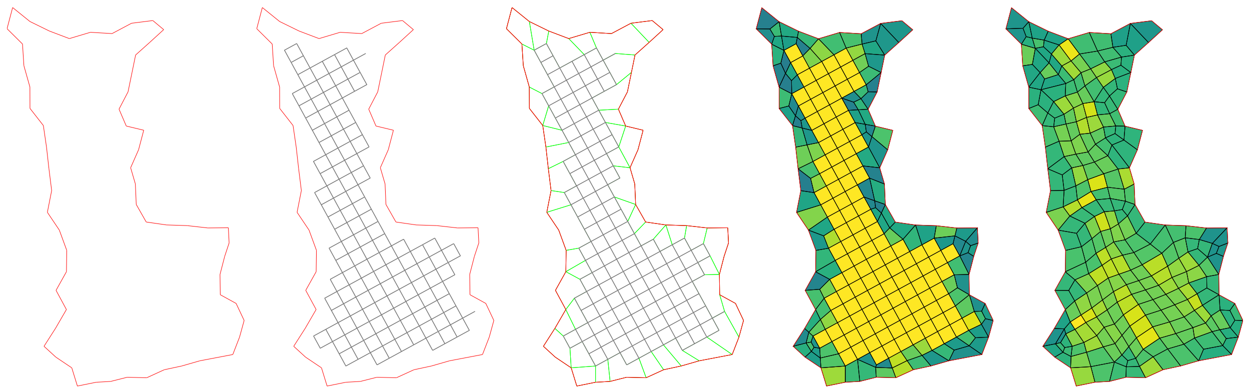
We accomplish this by removing elements that are too close to input features, creating a thin contiguous *channel* that separates the interior mesh from the boundary. Finding a high-quality all-quadrilateral decomposition of this channel constitutes the bulk of the contribution in this chapter.

The implementation in this chapter uses a simple grid to mesh the interior regions. This process fills the interior of the region with squares, removing any that would be placed too closely to the boundary or any internal features. This has the side-effect of making the interior singularity-free; all singularities in the resulting mesh will be inside or along the channel, which is likely to be high in singularities regardless of meshing method applied, due to the complex boundaries. We cannot guarantee an optimal number of singularities, but we clearly generate significantly fewer than the matching processes used in Gmsh or GMS.

The inputs to our groundwater meshing problem consist of the polygonal boundaries of the regions to be meshed, along with interior features such as rivers, material boundaries, political boundaries (*e.g.* water districts, counties, states), fault lines, and wells.

Our method is illustrated in Figure 2.1. We place a section of a regular grid, with a user-defined orientation, in the interior of the region, leaving a channel between the grid and the region boundary. We then construct a collection of small patches, each of which might contribute to a covering of the channel, and score these by the quality of quadrangulation of the patch. We next select a covering of the channel by a subset of the patches. Finally, we smooth the resulting mesh.

The intuition behind this grid-based approach is that the difficulty of meshing a geographic region lies in the complexity of its boundary, to which the orientation of elements has to adapt; unlike the situation



(a) Input boundary (b) With inlaid grid (c) Channel subdivided into patches (d) Meshed region (e) Smoothing to improve angles and aspect ratio

FIGURE 2.1. Overview of the meshing process, with an optional smoothing step. Colored using a perceptually-uniform color scale [vdWS05], with brighter colors indicating better internal angles.

in which there are smooth boundaries along which elements should be aligned, here the boundaries force nearby element orientations to change frequently. A-priori gridding of the interior limits the singularities to this difficult region and avoids unforced singularities in the interior.

Our approach produces all-quad meshes with element quality on par with modern tools that produce mixed meshes, such as Gmsh or GMS, while providing a large reduction in the number of singular points. We compare in depth in Section 2.3.

2.2. Construction

Like other grid-based approaches, our method fills the interior of the region with a regular grid. We remove elements from the inner grid that are near the perimeter, creating an unmeshed buffer zone that we call a *channel*. Channel width is a user-defined parameter that should be roughly one and a half times the desired element size. In order to preserve the geographic boundaries without introducing extra vertices, we use a wider channel and search for its best possible quadrilateral decomposition. We exploit the fact that the channel is narrow to perform a search of a great many (but not all) of the possible quadrangulations of the channel.

We organize this search by considering potential subdivisions of the channel into smaller polygons, with 4, 6 or 8 sides, that we call *patches*; we chose patches of this size as a trade-off between quality and computation time. We consider all possible local patches, select a quadrangulation of each one, and assign it a quality score, or *rank*. Then we select a compatible set of patches to produce the full channel covering that maximizes the minimum rank.

To assign a quadrilateralization to each patch, we consider a set of templates for the decomposition of 6-sided and 8-sided patches. Each template is applied to each patch, and quality of each of the resulting quads is measured; the quality of the template is the quality of its worst quad, and the rank of the patch is the quality of its best template.

The combinatorial search for the best covering of the channel by templates is the most algorithmically difficult part of the algorithm. We propose a process that simplifies the channel by iteratively merging extrusions and loops, by using a form of Dijkstra's algorithm, eventually reducing the channel into a simple ring, which can be optimally traversed using the same algorithm.

We do not guarantee that our mesh is optimal. However, our algorithm is polynomial in the size of the mesh, and we demonstrate some high-quality results. The rest of this section covers the details of the algorithm and our implementation.

2.2.1. Input Format. The input region is given as a Planar Straight-Line Graph (PSLG). This is a collection of vertices, and an unordered list of segments that connect pairs of vertices and do not intersect except at vertices. We assume that input geometry has been discretized to approximately the desired edge length h, with all incoming input segment lengths in the range $[\frac{1}{2}h, 2h)$. Each segment must be preserved in the output mesh; generally, input segments may not be be subdivided. This restriction may be relaxed in order to improve angle bounds, in which case perimeter segments may be split at most once.

We also need to require a density bound on the vertices so that the neighborhood of radius $4h$ around any input vertex contains at most a constant number of other input vertices. Also, we note that small angles in the input force small angles in the output.

Some subset of the segments must form a fully closed outer boundary, defining the region to be meshed. The region may also contain holes, also defined by closed loops. The boundary of every region, including its holes, has to have an even number of edges; otherwise there is no decomposition into quadrilaterals.

We allow the PSLG to include interior features, which are paths that either terminate inside the region or divide the region into subregions. Both sides of any path that terminates in the interior are included in the boundary of the region, so they always add an even number of edges. Interior paths that split the region must also have an even number of edges. All of the input PSLG edges become *outer channel edges*.

In preprocessing, we combine the polygonal elements, identify the region and subregions to be meshed, and orient their boundaries counter-clockwise, by "walking" each connected component of the PSLG.

2.2.2. Grid. A regular grid with edge length h is overlaid onto the region to be meshed. Any grid vertices outside the region are culled, as well as any vertices that are within a fixed distance δ from the perimeter of the region; we use $\delta = 1.4h$, but it can be set by the user.

Grid cells where all four corners remain are promoted to quads and form the interior of the mesh; remaining edges that connect grid vertices are enforced as edges that must remain in the output mesh, even if both of the two incident grid cells have been removed; this can be seen in Figure 2.2. Isolated vertices with no neighbors are discarded. Grid edges that are not in the interior of the grid are called *inner channel edges*.

Just as the input segments were "walked" to produce polygons with consistent orientation, the edges of the interior regular grid are walked in the opposite direction, producing a polygon with inverted orientation. Specifically, we are walking along the half-edges that are adjacent to the channel, where some edges, such as the hanging edges on the right side of Figure 2.2, will be traversed in both directions. These edges are not culled when the channel is carved as they serve to limit the width of the channel, as well as the lengths of the edges that will eventually cross it.

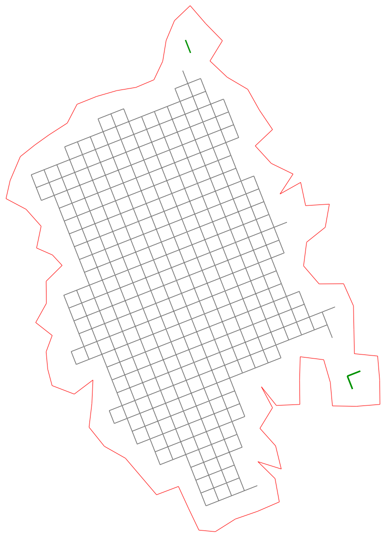


FIGURE 2.2. Input PSLG with inlaid grid

2.2.3. Cross-Edge Construction. Our next goal is to identify cuts that split the channel into nicely-quadrilateralizable patches. In addition to the outer and inner channel edges, we create additional *cross*

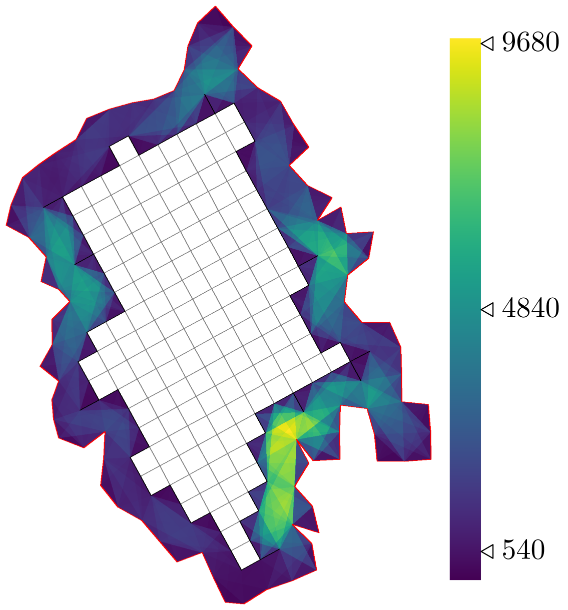


FIGURE 2.3. Histogram of patch footprints for a subregion of the Sacramento Valley dataset. Brighter pixels indicate more patches overlap that location.

edges that divide the channel; cross edges will separate patches from other patches. Cross edges usually connect inner and outer channel vertices, but there are also usually cross edges that connect non-consecutive vertices along the outer perimeter, and similarly along the inner grid. In order to produce the family of candidate cross edges, we test all pairs of vertices within a multiple of the desired mesh edge length h, keeping only those that do not cross an inner or outer edge. Our current test length is $4h$.

2.2.4. Patch Generation. We begin by finding all triangles in the graph formed by the inner and outer channel edges and the candidate cross edges. Then, any triangles that share a cross edge are glued together, forming a four-sided patch. This process of gluing along cross edges is continued between all patches (not just patches with the same number of edges), and all resulting patches with between 4 and 8 edges are kept as candidates for the following covering step. Note that we keep patches with odd edge lengths 5 and 7; as we will describe in Section 2.2.5, we sometimes introduce Steiner points along long cross edges, increasing a patch's side number. Three iterations of joining are sufficient to create all possible patches of side-length at most eight.

There is significant overlap in the resulting patches; typically, a point in the channel is covered by hundreds of, and possibly a few thousand, overlapping patches, as seen in Figure 2.3.

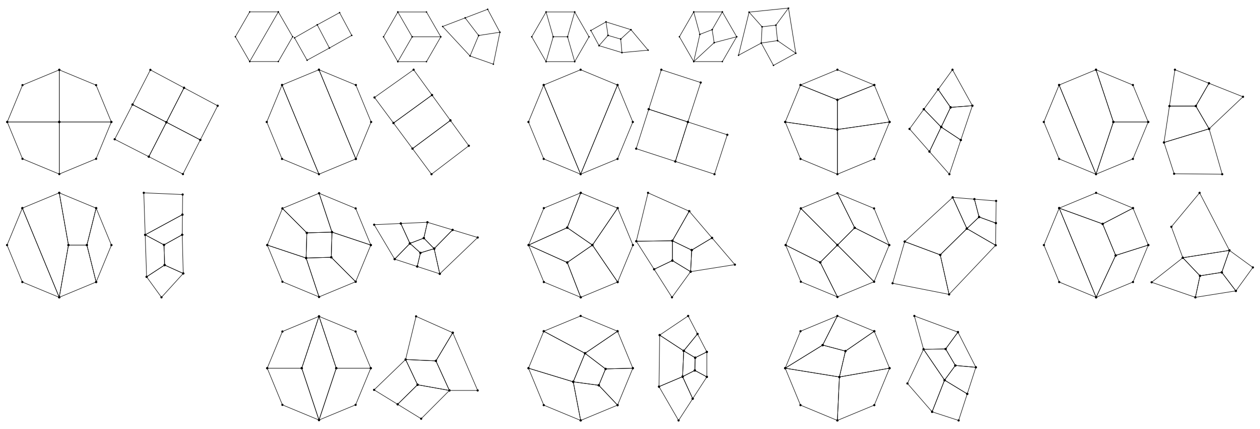


FIGURE 2.4. Templates for quadrilateral decomposition of patches, each paired with an example of a patch "in the wild" that was optimally subdivided with that template. The four degree-6 templates are shown above the thirteen degree-8 templates, with both sets ordered by decreasing rank of the example patch.

2.2.5. Patch Subdivision and Ranking. Our next step is to quadrilateralize all patches, using a set of templates, and rank each patch by how well it can be divided into quads. The templates we use are shown in Figure 2.4, and their construction is discussed in Section 2.2.6. While the vertex positions along the perimeter of the patch are fixed, a template may have internal Steiner points, which are free to move within the patch. For each template applied to each patch, we place the interior points using a Tutte embedding, such that every vertex is located at the barycenter of all connected vertices. A Tutte embedding does not optimize any specific mesh quality function; we use it for efficiency and because it produces generally high-quality results. We rank the results using the chosen quality metric. In our implementation, we use the maximum deviation of any quad angle from 90 degrees; we ignore quad aspect ratio at this point because all edges are guaranteed to be roughly the same size. We will use an explicit quality metric that includes aspect ratio when smoothing, as described in Section 2.2.10.

We use four possible templates for 6-sided patches and thirteen templates for 8-sided patches, but many of the templates are asymmetric and must be tested in different rotations on each patch. For any given patch, one rotation of one template will yield the highest quality subdivision, and that will determine the rank (and final quadrangulation) of that patch.

Cross edges may be split by adding a vertex at the midpoint, so that a 5-sided patch with a split edge will be meshed as a 6-sided patch, a 7-sided patch will be meshed as an 8-sided patch, most 6-sided patches will be meshed as both 6-sided patches and 8-sided patches, and most 4-sided patches will be meshed as both 4-sided patches and 6-sided patches. In the patch selection processes described below, two patches that

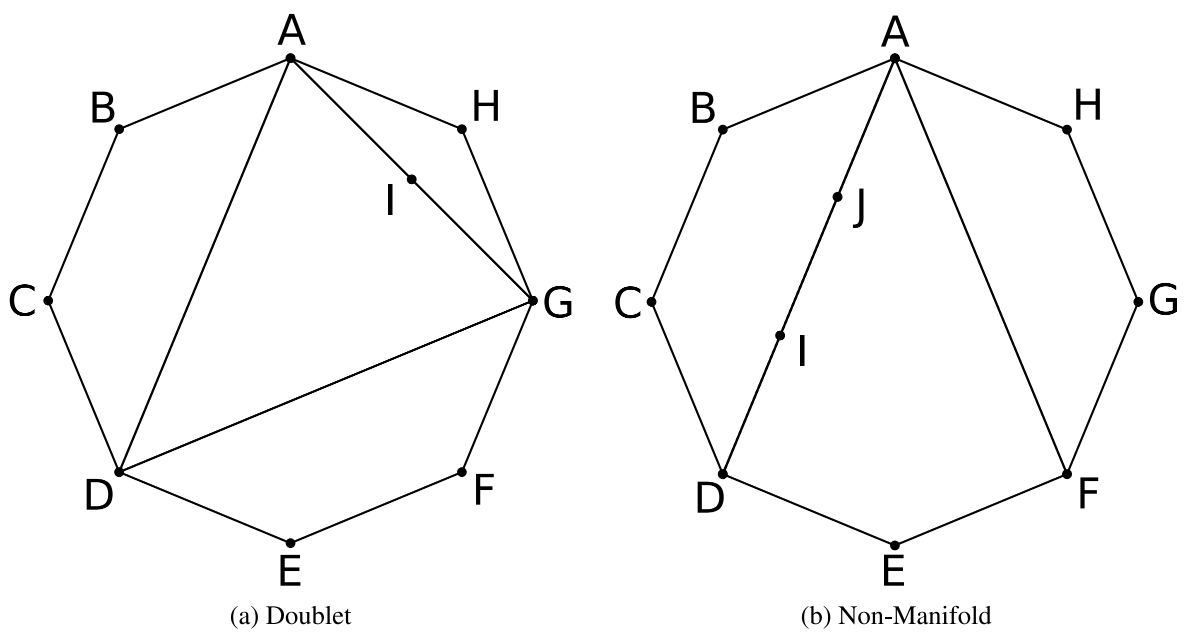


FIGURE 2.5. Example degenerate templates. In the doublet case, quads *ADGI* and *AIGH* share two edges *AI* and *IG*. In the non-manifold case, edge *AD* is shared by the three quads *ABCD*, *ADEF*, and the zero-area *ADIJ*.

share a cross edge will only be considered neighboring if both patches agree on whether to split the shared cross edge or not. In practice, this means that a single patch will actually be $2^{|\text{cross edges}|}$ different patches, each with a different set of split edges.

2.2.6. Template Generation. We consider templates that use at most three Steiner vertices to decompose 6-sided patches and at most four for 8-sided patches. Some experiments using more Steiner vertices did not improve the results for any of our examples.

There are four possible 6-sided templates, one for each of 0-3 Steiner vertices. There are other possible 2- and 3-vertex configurations, but they result in a degeneracy where two interior quads share two connecting edges. The first three templates cover all configurations listed in “Figure 10: Closure of boundaries with six nodes” in the seminal paper on paving [**BS91**]. Our fourth case improves the quad-decomposition of concave patches, which are not likely to arise in paving except perhaps on pathological input.

We used a recursive definition to enumerate all of the templates for 8-sided patches using at most four Steiner vertices. As shown in Figure 2.6, an 8-sided patch can be decomposed in one of five ways, corresponding to cutting off one, two, or three vertices from the patch. The leftmost template shows a single vertex being cut off; a single Steiner vertex must be added to ensure that the both subregions still have even perimeters; we do not consider adding three Steiner vertices in this case because it would produce a 10-sided patch. The two middle templates represent two vertices being cut off of the template, with zero and

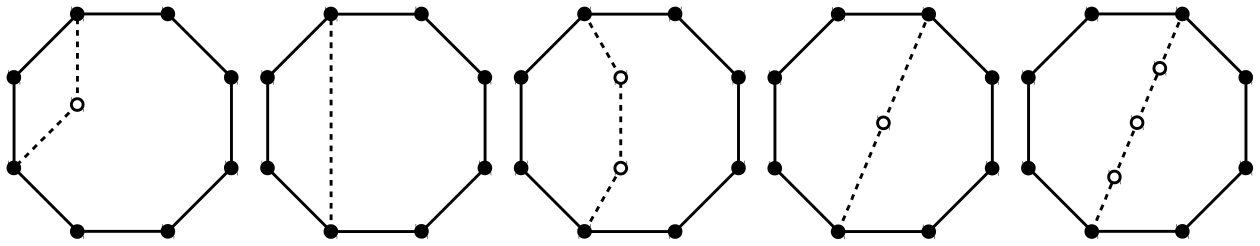


FIGURE 2.6. Recursive steps for creating 8-sided templates. From the left: corner vertex split, quad ear-cut, 6/8 split, 6/6 split, and 8/8 split. Steiner vertices are shown as hollow circles.

two Steiner vertices being added along the cut, respectively. The final templates show the patch being cut in half, with one or three vertices being added to ensure even perimeters. The resulting 6-sided and 8-sided regions are then recursively decomposed, stopping when at most four Steiner points are used. If the result is a quadrangulation of the original 8-sided patch, it is kept.

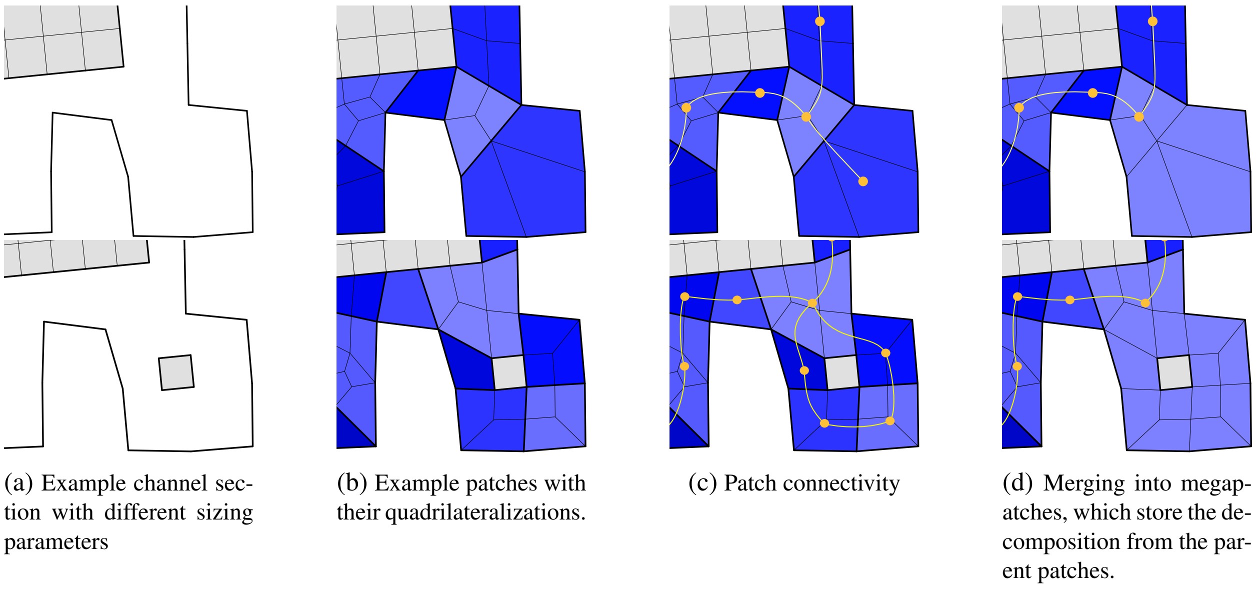
This recursive process produces the same topological decomposition many times. Eliminating duplicate topologies still leaves us with 1,006 unique 8-sided templates. The vast majority of these, however, are degenerate. Figure 2.5 shows two examples of degenerate templates, which are either doublets or non-manifold regions. Removing degenerate templates reduced the set of possible templates to only 36.

Even 36 templates, however, are expensive to try on each of hundreds of thousands of 8-sided patches, especially since each is tried in multiple rotations. To reduce the number of options, we tried the entire set on the Sacramento Valley and Merced datasets provided by the DWR, inserting the grid at different rotations and slightly different sizes to produce different channels, and then divided these into patches for testing. Interestingly, we observed that almost all templates were the best choice for at least one patch from this large dataset. But, templates that were used rarely were the best choices for "difficult" patches, which tended to be low-ranked, so that they were not selected for the final mesh anyway. Based on this experiment, we selected the thirteen most-used eight-sided templates, which are shown in Figure 2.4.

In Figure 2.4, the first of the six-sided templates, and two of the eight-sided templates, add no Steiner vertices. One might be tempted to think that these templates are redundant, since the quads they define should already have been found as four-sided patches. However, due to the fact that a cross edge may be split, the diagonals of these templates may not be cross edges, so we do need to consider them at this stage.

2.2.7. Megapatches. We refer to a patch that includes just one cross edge as a *cap*, a patch that includes two as a *bead*, and a patch that involve three or more cross edges as a *junction*. Any cover of the channel consists of some selection of caps and junctions, connected by sequences of beads.

A patch without any neighbors sharing one of its cross edges cannot be a part of a complete covering, and is removed from consideration. This may in turn result in further patches being removed.


(a) Example channel section with different sizing parameters

(b) Example patches with their quadrilateralizations.

(c) Patch connectivity

(d) Merging into megapatches, which store the decomposition from the parent patches.

FIGURE 2.7. Examples of the two megapatch formation processes. Above, a cap is merged with its neighboring junction. Below, a loop is merged together, forming a bead. Color and bold edges are used to distinguish the patches from their underlying quads.

Similarly, a *cap* represents a dead-end in the channel; see Figure 2.7. The cap can be removed by merging it with all of its neighbors. The resulting *megapatches* internally remember the originating patches, but can be treated like a single patch for the purposes of finding an optimal covering. Just as with the original patches, the quality of a megapatch is the quality of the worst contained quad, so it is simply the minimum of the ranks of the two merging patches. If the resulting megapatch is again a cap, it can be further merged to form larger megapatches. This process of forming megapatches is repeated until no caps remain.

Each megapatch that is topologically a cap represents some complete covering from its single cross edge back to the originating cap. So, for each (oriented) cross edge, only a single megapatch will ever need to be stored, representing the highest-quality meshing of that portion of the channel. This makes growing megapatches from caps computationally efficient, as inferior megapatches are easily identified and discarded. However, as megapatches expand over junctions, one is stored for each *pair* of oriented cross edges, so we stop the expansion process to avoid the combinatorial explosion of potential megapatches.

For some very thin input regions, where the boundary and features prevent an interior grid from being placed at all, the channel has genus zero. In this case, the best covering megapatch will represent the final meshing of the region, so our algorithm can terminate here.

In the more common case, the input region is large enough to allow placement of the interior mesh, so the channel has genus one or greater. In this case, the process of growing megapatches from caps cannot cover the region; as each megapatch grows from the originating cap, it will eventually hit a junction patch, and cannot continue growing. It does form a megapatch containing the junction patch, and the resulting megapatch has one fewer cross edge than the junction did; it typically becomes a bead with only two cross edges, as seen in the top of Figure 2.7. This process continues until every cap has grown into bead (or junction) megapatches.

2.2.8. Covering the Channel with Patches. At this point, there are no caps remaining in the set of patches, the region has non-zero genus, and the covering we seek is a collection of cycles, possibly connected by junctions; Figure 2.1c shows the case where the cover is a single cycle of patches.

The case of a single cycle is solved using a variant of Dijkstra's algorithm. Here, we search for the *widest path*, which is similar to a shortest-path problem, except that the width of a path is the minimum rank of any of its edges, which we aim to maximize. An arbitrary perimeter edge along the cycle is selected; any single-cover of the region will contain exactly one of the patches incident to this edge. These patches are iteratively tested for the widest path around the loop back to itself, with the caveat that once a patch has been reached, the search forward from that patch must not traverse the cross edge that initially reached that patch. This condition is necessary to ensure a single-cover of the region. The widest path from all of the tested patches is taken to be the final cover for the region.

This approach can also be used to reduce the genus of a channel by one. The genus of a channel is determined by the number of connected components in the interior. Taking all patches incident to one of the isolated interior components yields a loop much like the genus-one case, except that patches in the loop may have cross edges that are not incident to the isolated component and thus do not have a neighbor across that edge in this subset of patches. Instead of taking only the widest path from the best patch, all paths from all patches are considered. For each combination of unsatisfied cross edges along the path, a new megapatch is formed, representing this particular path around the interior component. This can be seen in the lower portion of Figure 2.7. This will be performed for all isolated components in the interior except for the largest, which is left as the genus-one case.

2.2.9. Computational Complexity. We consider the complexity in terms of the outer channel edge count p and the total area of the region. First, we note that the interior grid produces $O(area/h^2)$ elements, which are easy to determine within that time bound. The final channel cover contains $O(p)$ quads, since each outer channel edge ends up incident to a single patch.

The complexity of our channel covering algorithm is greater than the output size, however, since we generate a significantly greater number of potential patches. Since we only connect vertex pairs within distance $4h$ with cross edges, and we assume that there are not more than a constant number of these, a vertex will participate in at most a constant number of cross edges. A single vertex, in turn, will belong to only a constant (although large!) number of patches, giving us a total bound of $O(p)$ on the number of patches considered. For convenience, we will count each junction patch multiple times, in particular $k - 1$ times, where k is the number of cross edges on the boundary of the junction patch; k is at most eight.

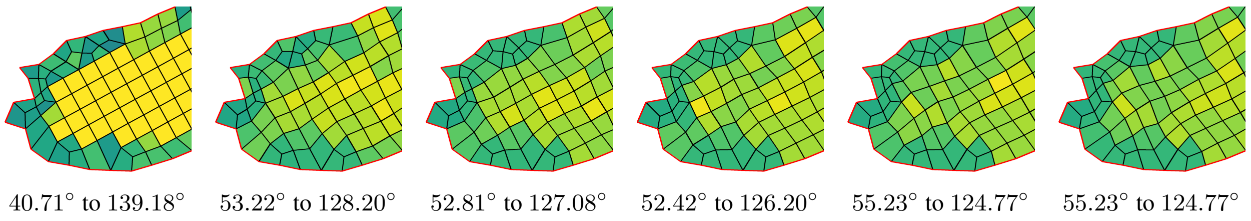
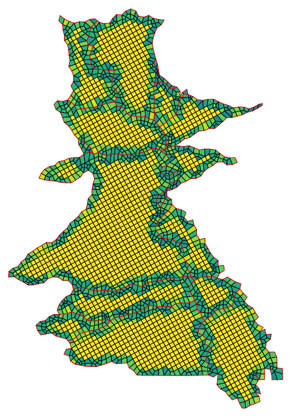


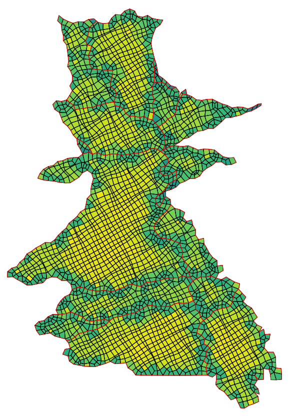
FIGURE 2.8. Successive smoothing iterations. Angles and aspect ratios of inner elements are sacrificed for overall angle improvement.

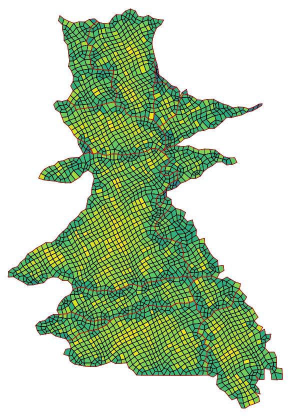
Let us consider the process of forming a set of megapatches, starting from a cap. This is equivalent to forming a widest path tree, starting at the cap, with distinct junction patches at the leaves. The set of patches considered when computing this tree includes anything reachable from the cap without passing through a junction. It can be computed in time $O(p_i \lg p_i)$ in the number of patches considered, using our current $O(n \lg n)$ implementation of widest path based on Dijkstra's algorithm. There is an $O(n)$ algorithm [**Pun91**] that we could also use here. The cap patch, and all bead patches examined while making the tree, will be removed in subsequent sub-problems and represented by one or more of the megapatches associated with the junction patches at the leaves. So each of them is processed once. Each (leaf) junction patch will end up with one of its cross edges inside the megapatch. The junction patch may appear in subsequent sub-problems, either as a bead patch, or as a junction with one fewer cross edges. Since a junction with k cross edges loses a cross edge every time it is used, and then finally is counted once as a bead, it is counted $k - 1$ times.

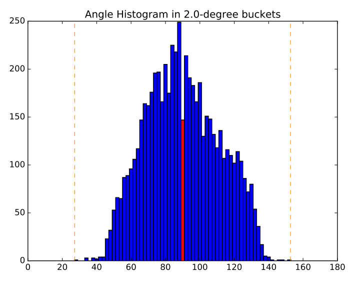
Genus reduction is a relatively expensive process. There is a large (but constant) bound on the number of source patches considered, which we will call the patch density d. This means each genus reduction phase is a $O(dp_i \lg p_i)$ search, replacing the p_i patches with $O(d^j)$ megapatches for a loop with j junctions.

Treating d and j as constant, the total time is $O(p \lg p)$ in the size p of the input; using the theoretically better widest path implementation would bring this down to $O(p)$. Unfortunately, the constant is very large because the number of patches, while $O(p)$, is very large. Also, it is important to note that the solve to find the highest-quality channel cover is currently one of the fastest phases in the overall process, in practice, so we do not feel the need to replace it yet.

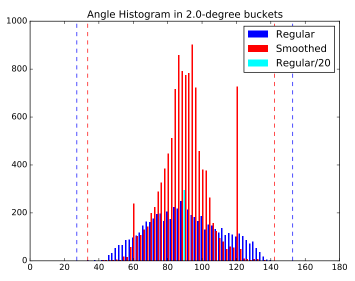

(a) Sacramento Valley, meshed


(b) With smoothing to enhance angles

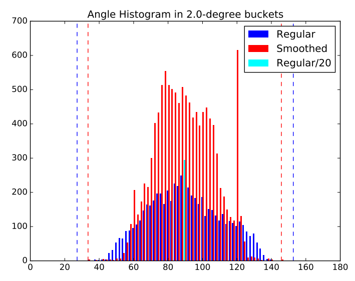

(c) With smoothing to enhance angles and minimize stretch



Angle histogram for the mesh above. The red bar, which includes all 90° angles, is drawn at 1/40 of actual height. The dashed lines show extreme angles.



Angle comparison between regular (blue) and smoothed (red) meshes. Note the spikes at 120° and 60°, corresponding to valence-3 and valence-5 singularities.



As expected, preserving element aspect ratios results in a drop in average angle quality but still improves the poorest angles.

FIGURE 2.9. Analysis of angles in the resulting meshes, with optional smoothing steps. Smoothing only to enhance angles will produce arbitrarily skinny rectangular elements, creating visible "pinches" in the mesh.

2.2.10. Smoothing. We perform smoothing as a post-processing step. Instead of the common Laplacian smoothing, we try to improve the angles directly by repositioning each Steiner vertex such that it minimizes the deviation of all dependent angles from 90°. We currently use the Nelder-Mead Simplex algorithm [NM65, Bro01], which is available in the mesh smoothing package Mesquite [KFDT] and recommended for non-differentiable functions, such as taking the maximum of several metrics. We use the implementation in SciPy [VGO+20] for our angle-based smoothing. A benefit to this approach is that our ranking function may be directly used here; should we switch to a quality ranking function that is a combination of aspect ratio, maximum angle, and deviation from ideal size, that ranking function can still carry over to the smoothing step. An example of the smoothing is shown in Figure 2.8. We also employ a smoothing function that is a combination of angle deviation and *stretch*, the scaled ratio between the longest diagonal of the quad and the shortest edge, as seen in Figure 2.9.

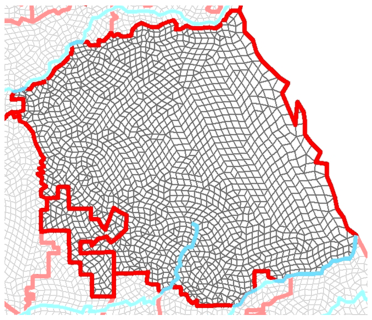
An obvious benefit to the smoothing is that it relaxes the interior grid and allows elements along the perimeter to take on better shapes. However, the templates that were applied to the perimeter elements may no longer represent the ideal topological configuration. Local topological mesh improvement method may be able to improve these.

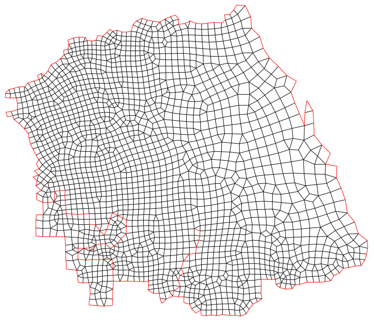
2.3. Results

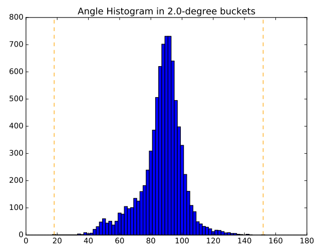
Our working code is written in Python and was executed with PyPy 5.1.0 for the timings we report here. Even though we take advantage of the Just-in-Time compilation of PyPy, the code is rather slow. Initially, Figure 2.9a, without smoothing, took just under 30 minutes to run. The smoothing to produce Figure 2.9c took an additional 10 minutes for 30 iterations; we require more smoothing passes as it takes longer for the interior regions to converge to a configuration that is balanced with the perimeter.

We profiled the code to guide our optimization efforts. The most expensive phases of the algorithm, in decreasing order of overall runtime, are template application and patch ranking (89%), patch generation (5%), perimeter edge reordering and isolated region detection (3%), inner grid generation and channel formation (1%). The time taken to actually solve for the optimal channel cover was under four seconds, making it one quarter of a percent of the overall runtime; the most complex channel in this run was only genus two, so we expect quick solution times. A full 10% of overall execution time is spent simply measuring the quality of potential quadrilaterals within the patch ranking phase.

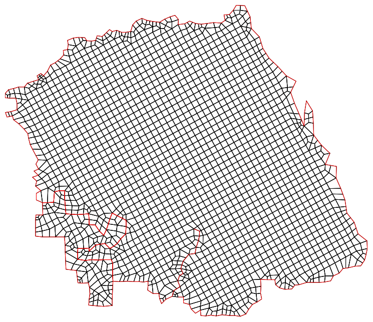
We expect that a speed-focused reimplementation in C++, taking advantage of ample opportunities for parallelism, would produce Figure 2.9a (again, without smoothing) in under a minute. However, our profiling has indicated that the template application phase was an obvious bottleneck. Parallelizing just that

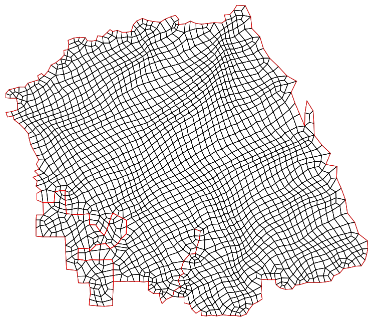

(a) Mesh produced by GMS

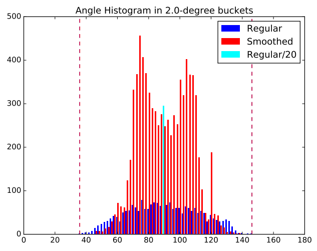

(b) Mesh produced by Gmsh



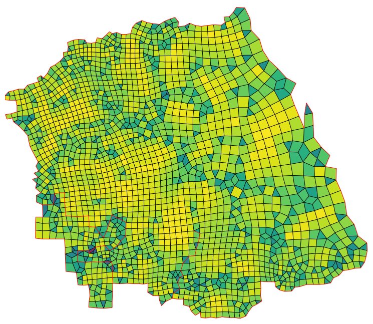
(c) Angle histogram for Gmsh

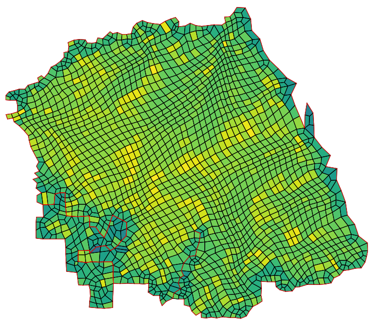

(d) Our method


(e) Our method, smoothed



(f) Angle histogram for our approach


(g) Gmsh


(h) Our method, smoothed

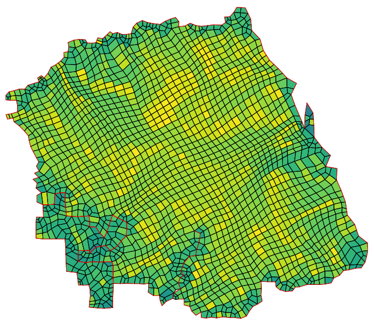

(i) Our method, allowing perimeter edges to be split in half, smoothed

FIGURE 2.10. Eastern edge of Merced County, California. The border is extracted from (a), and meshed with Gmsh (b,g) and our tool (d,e,h,i).

phase, by running 8 separate Python threads, reduced the total runtime to just 526 seconds. We further reduced the overall runtime to 165 seconds by porting just that code to C++, parallelized with OpenMP.

We cannot guarantee that any given channel can be meshed, but testing shows that finding a cover is very likely. We ran a series of tests against the Merced dataset, with different combinations of desired edge length

TABLE 2.1. Statistics for Merced County Meshes

	Gmsh		Our Method		Allowing Perim. Splits	
	Delaunay	DelQuad	Base	Smoothed	Base	Smoothed
Image	-	2.10b,2.10g	2.10d	2.10e,2.10h	-	2.10i
Elements	3412	2161	2017	2017	2082	2082
Triangles	778	262	-	-	-	-
Quads	2634	1899	2017	2017	2082	2082
Angle Min.	25.23	18.06	35.84	35.88	40.77	40.77
Max.	149.42	151.97	146.20	146.20	136.02	138.21
Singular Points	1146	286	134	134	153	153
Degree-3	277	55	93	93	101	101
Degree-5	849	228	41	41	52	52
Other	20	3	0	0	0	0

h, channel width δ, grid orientation and offset. The desired edge length was linearly sampled from the mean input edge length to one standard deviation below. Among 360 test cases, 18 failed to produce a covering of the channel. Eight of these succeeded after increasing the maximum cross edge length from $4h$ to $5h$, at the expense of almost doubled runtime and memory usage. Nine tests failed due to groups of internal edges that formed zero-area polygons, bypassing an orientation check and allowing them to be walked in the wrong direction. These cases succeeded after manually reversing the direction of the offending group of edges. There were only four grid offsets tested, with a zero or one-half phase shift in either dimension, and every failed test case had at least one corresponding successful test case that differed only in offset.

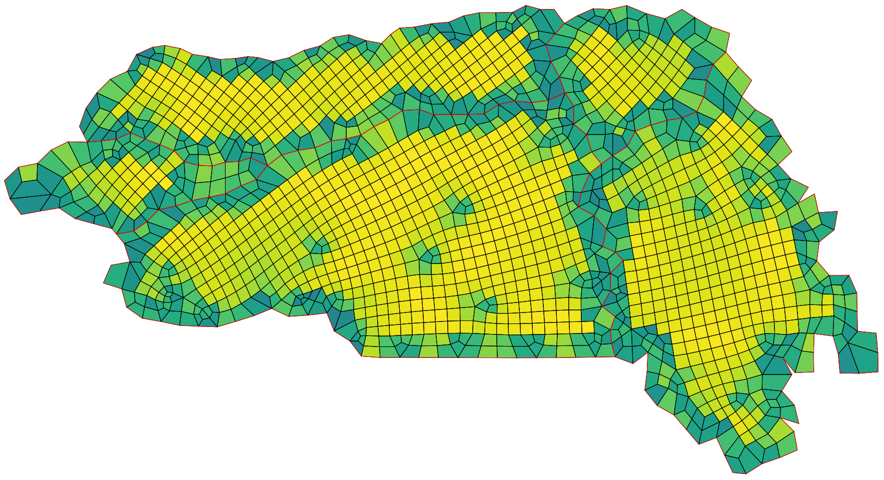


FIGURE 2.11. Replacing the inner grid with the boundary-aligned output of *Instant Field-Aligned Meshes* [**JTPSH15**]. The target element count was set at one-quarter the desired amount, and then Catmull-Clark subdivision was applied to ensure an all-quad interior at the target resolution.

Figure 2.9 shows our method on the northernmost portion of the Sacramento Valley in California. While our objective function during the initial meshing process is based exclusively on element angles, that is not sufficient for the smoothing step, shown in Figure 2.9b, as it squishes some interior elements into arbitrarily-skinny rectangles. For the smoothing step, we use an objective function that takes element shape into account (Figure 2.9c).

We compare our method against Gmsh [**GR09**] in Figure 2.10, using data from Merced County, California, which was extracted from a mesh produced by GMS [**OJH96**]. We show the meshes without color to better contrast the three methods. Angle histograms are shown for both methods; note that the histogram for Gmsh (Figure 2.10c) is skewed toward smaller angles as it also produces triangles. If the boundary conditions are relaxed, and perimeter edges are allowed to be split at most once, at their midpoint, we produce the mesh seen in Figure 2.10i; while the interior quality is quite similar, the poorest angles along the perimeter are significantly improved. Table 2.1 shows statistics for some of the meshes shown in Figure 2.10.

We are exploring the use of alternate interiors for our meshing process. For example, Figure 2.11 shows the output of *Instant Field-Aligned Meshes* [**JTPSH15**] as a replacement for the interior grid. The result is a mesh where the interior aligns with the boundaries while also strictly adhering to the complicated boundary of the region.

CHAPTER 3

Size-Aware Cross Fields

This chapter includes a novel extension to the construction found in *Globally Optimal Direction Fields* [**KCPS13**] that allows for the generation of size-compatible cross fields, as well as our definition of size-compatibility, which is covered in Section 3.2.1. As such, significant portions of this chapter are dedicated to explaining the mechanisms underlying the *GODF* approach, as well as the derivation and evaluation of our extension. In Chapter 5, these fields will be used to produce all-quad meshes.

We take an input *size field* and optimize toward a *compatible cross field*, which we may not exactly achieve. The size field must be continuous, but is otherwise unrestricted, although higher rates of change in the size field tend to lower the quality of the resulting mesh. We estimate the sizing implied by a cross field in order to evaluate the resulting fields here, with a more rigorous definition of size explored in Chapter 4, followed by mesh extraction in Chapter 5.

This chapter is split into four main sections. Section 3.1 covers the foundational topics for cross fields, including intuition and implementation details where appropriate. Section 3.2 discusses the sizes implicit in meshes and cross fields, as well as our observation that a varying size function on a 2D surface behaves in many ways like a curved surface in 3D, allowing us to exploit tools that use surface curvature, like *GODF*. In Section 3.3, we derive much of the underlying math behind *GODF* and describe our implementation. We end by exploring a few cross fields produced with this method in Section 3.4.

3.1. Cross Fields

3.1.1. Conformal Maps.

DEFINITION 3.1.1 (Map). *A map M is a relation between two regions r_1 and r_2, such that every point $p \in r_1$ has a corresponding $M(p) \in r_2$. While the inverse map M^{-1} is a useful concept, some of the maps we will consider do not have a unique inverse: there may be $p_1 \neq p_2$ where $M(p_1) = M(p_2)$.*

A map M between two regions r_1 and r_2 may be arbitrarily defined and indeed is often a collection of independent functions mapping subregions of r_1 onto r_2. In practice, M is a piecewise function defined over a triangle mesh, but for the sake of simplicity many of our examples will be complex polynomials, such as the complex map $f(c) = c^2$ for example, which *maps* the point $c = i - 1$ to $f(c) = c^2 = -2i$. We can see this as a map from the complex plane to itself, or from one complex plane to another; also, treating source and destination regions as Euclidean planes, the same complex function $f(c) = c^2$ would map the point $p = (-1, 1)$ to the point $M(p) = (0, -2)$. Much of the related research is in the context of mapping a 2D surface embedded in $\mathbb{R}^3$ to the Euclidean plane, so this will often serve as a third mental model for the associated concepts.

DEFINITION 3.1.2 (Tangent Plane). *Given a point p on a surface S, a tangent plane T_p is a plane through p that is tangent to the surface S. The local neighborhood of S around p may be flattened by being projected onto the tangent plane. The tangent plane T_p defines a local coordinate system, with p as the origin.*

At any point on the Euclidean plane, the tangent plane is another Euclidean plane.

DEFINITION 3.1.3 (Differential). *The differential dM of a map M from r_1 to r_2 at a point $p \in r_1$ are the vectors $\frac{dM}{dx}$, $\frac{dM}{dy}$, given local coordinates x and y defined at p.*

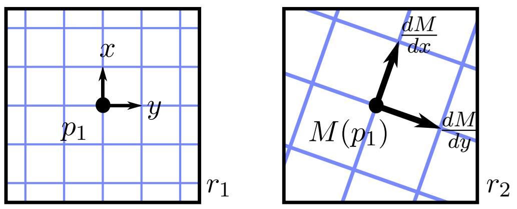


FIGURE 3.1. The map M takes $p_1 \in r_1$ to $M(p_1) \in r_2$.

This correspondence between the basis vectors at p and the differential vectors at $M(p)$ defines a relationship between the tangent planes T_p and $T_{M(p)}$, but not necessarily the regions r_1 and r_2; T_p and $T_{M(p)}$ are Euclidean planes, even if the regions r_1 and r_2 are not. The map M and a coordinate system defined on r_1 produces a (possibly curved) coordinate system on r_2; the differential dM and a coordinate system defined on T_p produces a straight coordinate system on $T_{M(p)}$.

The maps we are considering may be between arbitrary curved surfaces embedded in $\mathbb{R}^3$. The differentials of these maps map between flat tangent planes at the source and destination points. The utility of these tangent planes is that they establish a correspondence between the neighborhoods around p and $M(p)$ in a manner that is independent of the underlying coordinate systems of r_1 and r_2; any change in the coordinate system in r_1 will not change the tangent plane, but only the basis vectors that points in the tangent plane are expressed with respect to, which will cause a corresponding change in the differential vectors. The map M

induces this linear transform between the tangent spaces T_p and $T_{M(p)}$, and like any linear transformation, it is a combination of rotation, scale, mirror, and skew.

DEFINITION 3.1.4 (Conformal Map). *A conformal map M is an angle-preserving map: for any two curves $C_1, C_2 \in r_1$ that intersect at point p, the mapped curves $M(C_1)$ and $M(C_2)$ meet at $M(p) \in r_2$ at the same angle. The equality condition for angles implies that the inverse of a conformal map is also conformal.*

Conformal maps are of particular interest because the differential of a conformal map is orthogonal if the basis vectors in r_1 are orthogonal and have the same length. The orthogonality and size conditions means that the transform between tangent spaces T_p and $T_{M(p)}$ can only consist of rotation and scale; we will not consider *anti-conformal* maps where all angles are negated, which would be the case if M mirrored the tangent planes. In order to illustrate several concepts relevant to the construction of cross fields, we will explore a specific class of conformal maps: complex analytic maps. These are important because all conformal plane-to-plane maps are complex analytic functions, and vice-versa.

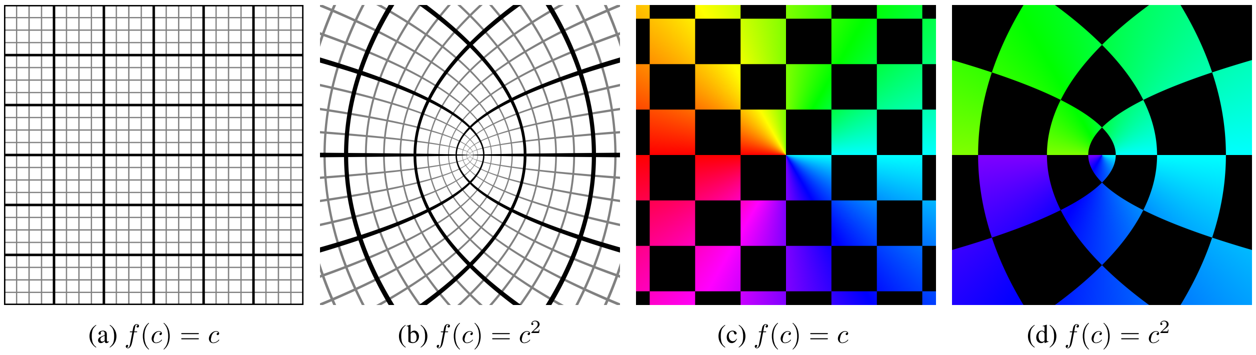

(a) $f(c) = c$ (b) $f(c) = c^2$ (c) $f(c) = c$ (d) $f(c) = c^2$

FIGURE 3.2. Iso-integer line drawing and checkerboard drawing for complex functions; the color information is calculated in the source coordinates c, and *mapped* onto the position $f(c)$.

Figure 3.2 illustrates the complex map $f(c) = c^2$ in two ways. The left figures draw lines along integer-valued coordinates, with additional quarter lines in gray, while the right images show a checkerboard pattern used in Will Bolden's Complex function plotter [Bol21]. This checkerboard pattern looks at the Euclidean encoding $c = x + iy$, and darkens any region where $\lfloor x \rfloor \equiv \lfloor y \rfloor \mod 2$, and the remaining pixels are colored by $H = -arg(c)$ in a HSV color space such that $\pm\pi$ is red; Figure 3.2c shows the full spectrum. The colors evaluated at c are then *mapped* onto the coordinates at $f(c)$. Practically, when actually rendering the images, these are evaluated by taking the pixel coordinates, translated and scaled rescaled to the area of interest, and evaluating the inverse map $f^{-1}(c) = \sqrt{c}$, yielding the "original" complex coordinate that the color information is derived from. It is important to note that the inverse map is not unique, as every nonzero complex number $z = c^2$ has two roots c; in polar coordinates, the two magnitudes are identical but the angles differ by exactly π, so we take the canonical root to be the one where the sign of the angle $(\mathrm{mod} 2\pi)$ matches the original angle. This only influences the choice of color, as both roots share the same parity when drawing the checkerboard.

Note that, in Figure 3.2d, the color of the non-checkered tiles can be used to determine the source coordinate in the Euclidean plane. So, within each square, each point has coordinates in both the origin surface r_1 and final surface r_2. This process is often called *reparameterization*, as there are many cases where the (often simpler) coordinate system of the source surface is preferred to the actual embedding of the mapped surface. A concrete example of this is texture atlasing in 3D graphics, where each point p on a 3D model has *model coordinates* in $\mathbb{R}^3$, expressing the location of the point in the local environment, and *texture coordinates* in $\mathbb{R}^2$, giving a location into a 2D texture that will be used for color information. This not only allows color information to be stored for just the surface of the object, but also defines a coordinate system on the surface of the object allowing for animation techniques that are inherently local. A widely used example of this is applying a *noise* function to add surface detail via a pseudorandom function of the local surface coordinates so the result is consistent regardless of where the model is located in the scene [1].

With the goal of drawing all-quad meshes in mind, this is a promising start. A conformal map from the Euclidean plane onto our surface gives a decomposition of the surface into quads; this chapter is an exploration of the properties we want such a map to have, and a method of computing one. A conformal map ensures that a grid on r_1 maps to quads on r_2 where the produced quads meet at perfect 90-degree angles, However, the mapped quads are curved, and once discretized to form quads with straight-line edges between all four vertices, the angles will necessarily deviate from ideal. However, this does mean that refinement can be employed to achieve arbitrarily-high quality bounds for *most* vertices; as seen in Figure 3.3, there is an irregular vertex at the center that cannot be improved, regardless of the amount

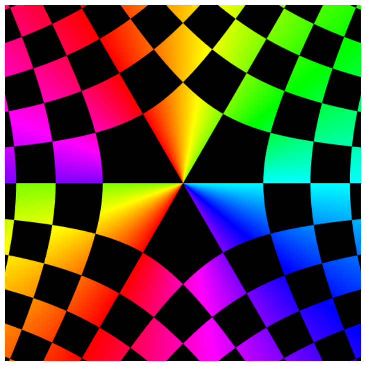


FIGURE 3.3. $f(c) = c^{2/3}$

[1]Ken Perlin's foundational paper on the topic [**Per85**] explicitly did not use surface coordinates but instead used local object coordinates for *solid textures*, implicitly defining color information for the entire volume. Noise based on surface coordinates simplifies consistent texturing of deformable objects.

of refinement performed. Six incident quads intersecting at that point means that the best-case scenario is that all angles are $\frac{\pi}{3}$; any attempt to bring one angle closer to the ideal $\frac{\pi}{2}$ necessarily worsens the other angles. These irregular vertices are known as *singularities* and are an instrumental part of how a cross field behaves.

3.1.2. Direction Fields.

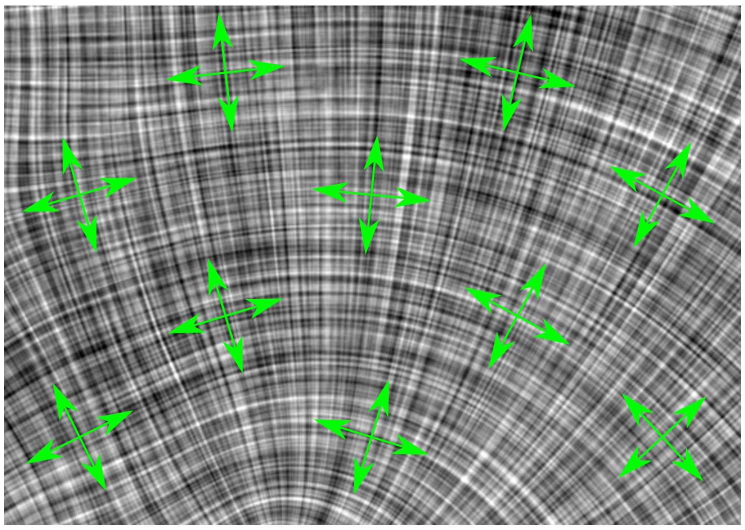
DEFINITION 3.1.5 (Direction Field). *A direction field γ over a surface $\mathbb{S}$ defines a direction at every point p of the domain: $\gamma : p \in \mathbb{S} \mapsto \theta \in [0, 2\pi)$*

A direction field differs from a *vector field* in that direction fields only provide directions, and do not include magnitude, at each point in the region. Except at critical points, which will be discussed later, this direction smoothly changes over the region; as the distance between two points approaches zero, the difference between their associated directions also approaches zero. A direction field is often computed over a region as a precursor to a final mesh, the two most common cases being a 2D manifold surface in 3D space or the interior of a polygon in 2D space. We focus on the latter case.

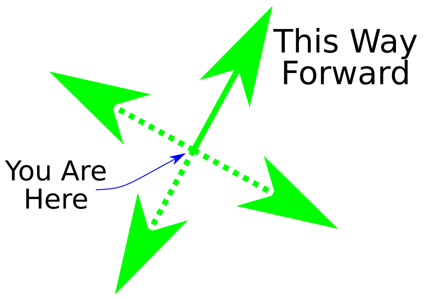
A generalization of direction fields allows *rotational symmetry*, where a rotationally-symmetric direction field of degree-*n* will define *n* evenly-spaced directions at each point. Analogous to how adding a multiple of 2π to an angle does not change its resulting direction, adding a multiple of $\frac{2\pi}{n}$ to an angle in a degree-*n* field does not result in any change, because each direction lines up with one of the symmetric copies. In this sense, the only meaningful values an angle can take are in the range $[0, \frac{2\pi}{n})$.

DEFINITION 3.1.6 (*n-RoSy* Direction Field). *An n-fold Rotationally-Symmetric direction field or n-RoSy direction field is a direction field that exhibits rotational symmetry of degree n. Directions that differ by $\frac{2\pi}{n}$ are considered equivalent: for every every point p of the domain: $\gamma : p \in \mathbb{S} \mapsto \theta \in [0, \frac{2\pi}{n})$, and the standard definition of continuity is extended such that* $\lim_{|\Delta x| \to 0} \theta_{x+\Delta x} - \theta_x = 0(mod\frac{2\pi}{n})$.

Due to their utility for generating quad meshes we focus on *cross fields*, where $n = 4$, and angles are in the range $[0, \frac{\pi}{2})$, but much of what is said here generalizes to $n \neq 4$. We will use the term *direction field* to refer to n-RoSy direction fields in general.

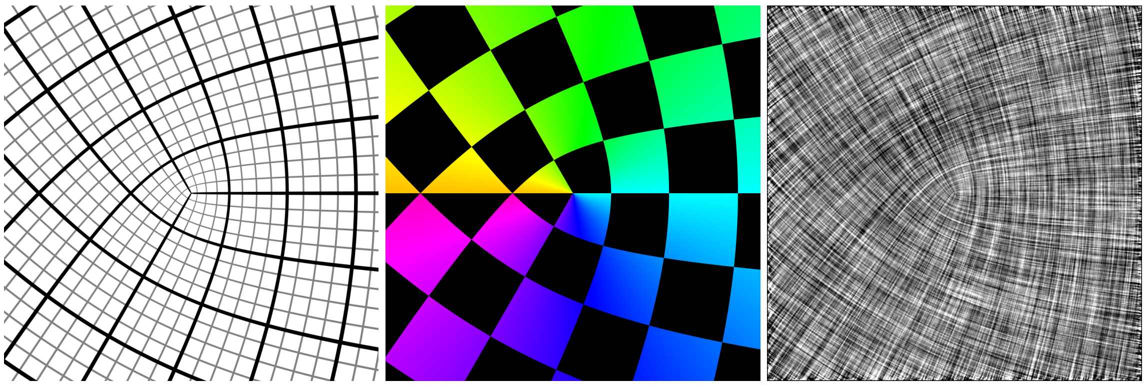


(a) A LIC rendering, an intuitive representation of the direction along the entire field, with the four direction vectors shown at some sample points.



(b) A simple representation of a point in the field. While only one direction is explicitly stored (shown solid), the three implicit directions (shown dashed) are equivalent at this point.

FIGURE 3.4. A *4-RoSy* direction field.



(a) Rendering (quarter-)integer-valued isolines

(b) Source coordinate parity checkerboard

(c) The Line Integral Convolution (LIC)

FIGURE 3.5. The complex map $f(c) = c^{3/4}$ rendered using three techniques. Discussed further in Section 3.1.11, the LIC is able to render arbitrary direction fields.

We now introduce one more visualization, particularly suited for direction fields: the *Line Integral Convolution*. Shown here in Figure 3.5c, it provides an intuitive display of a direction field by appearing to be combed along each direction simultaneously, resulting in a distinct knurled or cross-hatched effect. Section 3.1.11 covers how LIC images are generated.

As an example of a commonly-used direction field, we can consider navigation on Earth. This direction field is based on the Earth's magnetism, which defines the direction North at every point, except at the poles where there is no North at the North Pole, and every direction is North from the South Pole. This is a *1-RoSy* direction field, as a direction is effectively unchanged after rotating by $360°$. If this was a cross field instead, a compass would point at the closest cardinal direction, instead of always North. This is equivalent to a compass with an unmarked cross; while the magnet under the cross is always pointing North, it is impossible to tell which of the four arms of the cross is the actual Northward direction, so North, West, South, and East all become interchangeable. This is an example of a *4-RoSy* field that does *not* correspond to an underlying conformal map.

3.1.2.1. *Cross Fields and Quad Meshes.* There is a strong, albeit imperfect, relationship between quad meshes and cross fields.

Within any quad, there is an implicit coordinate system u, v that measures the position of some interior point p between the two pairs of opposing edges; typically, these are the same coordinates used when performing bilinear interpolation over the quad. However, these vectors are not guaranteed to be orthogonal or the same length, so a quad mesh does not directly imply a cross field, but it does come close (Section 3.1 of [BLH+17] describes a method to (and pitfalls of) extracting an orthogonal cross field from an arbitrary quad mesh).

This process may be reversed; around any point p in a cross field, an arbitrarily-sized square may be drawn. This does not produce a quad mesh, of course, as there is nothing to establish connectivity between quads, or to prevent their overlap. If however, we consider the infinitesimal edge length ϵ, the square instead becomes an infinitesimal neighborhood around p; This in effect produces a mesh of infinite density that is perfectly aligned with the given field. While illustrative, this is not exactly practical; the work required to extract a quad mesh from a cross field with more useful element sizes is the topic of Chapter 5.

The meshes produced in Chapter 5 will not have perfectly-rectangular quads, except in the case of a constant field. So, while there is a relationship between cross fields and quad meshes, there is information lost in each conversion. Therefore, given a field $\hat{\varphi}$ extracted from a quadrangulation Q, which was in turn extracted from a field φ, we expect that $\hat{\varphi} \neq \varphi$, although they should be quite similar.

3.1.2.2. *Direction Interpolation.* Our direction fields are represented by a discrete sample of directions, stored at the vertices of a triangle mesh, with the angles being interpolated over each triangle to define a continuous direction field over the region. We will now discuss some of the necessary considerations when deciding how to represent and interpolate directions.

Careful consideration needs to be taken when interpolating angles, especially in *n-RoSy* fields. Even with regular direction (*1-RoSy*) fields, correctly interpolating between two directions requires checking to see whether a clockwise or counterclockwise rotation would be correct. This is in contrast to interpolating between two *rotations*, which are inherently oriented. Interpolating half-way between the directions 0 and $\frac{3\pi}{2}$ will yield $\frac{7\pi}{4}$, because that is the shortest path between them. If the same values were treated as rotations, the sign of the second value implies it is a counter-clockwise rotation, and the correct midpoint between them would be $\frac{3\pi}{4}$.

Another way to represent directions in 2D is as 2-vectors, or equivalently as complex numbers. Interpolating between vectors no longer requires testing to see which is the correct direction to rotate in, as storing the direction as two components eliminates the $2\pi \leftrightarrow 0$ changeover that necessitates the direction check. However, this introduces another potential issue: interpolating between two directly opposed directions results in a vector where both components are zero, which cannot be normalized back to a proper direction. In practice, this case is unlikely to arise when interpolating two vectors, but when interpolating discrete directions to form a continuous field, we will encounter this situation at *singularities*, which we will discuss momentarily.

Rotational symmetry reintroduces the rotation problem when interpolating directions, even when those directions are represented as vectors or complex numbers. In a *4-RoSy* field, the direction vectors $(1, 0)$ and $(0, 1)$ are equivalent, so interpolating between them should result in the (unchanged) vector $(1, 0)$. Clearly, performing simple component-wise interpolation is insufficient.

In *GODF*, Knoppel et. al. created a representation where *n-RoSy* directions are stored as complex numbers, raised to the n^{th} power [**KCPS13**]. Multiplying complex numbers adds their rotations, so this effectively multiplies any rotation by n. Importantly, this means that the canonical values for a *n-Rosy* direction, which are in the range $[0, \frac{2\pi}{n})$, are mapped to the range $[0, 2\pi)$, regardless of n. These vectors may be interpolated just like non-symmetric directions in the complex plane, without the need to check both potential directions of rotation between them. Once interpolated, taking the fourth root of the rotation restores the canonical *n-RoSy* value in $[0, \frac{2\pi}{n})$. We also use this field representation, storing fourth-power rotations at every vertex of a triangle mesh; the direction at any point can be evaluated by using barycentric interpolation of the vertex rotations and then taking the fourth root to yield a complex vector, which can be normalized to produce a unit direction vector, but is most often passed to the two-parameter inverse tangent function **atan2** to produce a scalar rotation angle.

At first, this might seem like an excessive amount of work to avoid the bookkeeping of checking directions. The critical difference between representing directions as a complex value to the n^{th} power and any of the other discussed methods is that interpolation between the higher-order complex values is order-independent. Interpolating between three directions in any of the other representations can result in a different outcome just based on the order of the arguments, even when their relative factors are fixed; this is discussed further in Section 3.1.2.3.

3.1.2.3. *Direction Field Representation.* The direction field is stored in relation to the background triangle mesh on which it is computed. Each vertex in the triangle mesh stores a single direction, and the direction associated with any point p in a triangle t is a weighted sum of the angles at the vertices, weighted by the barycentric coordinates of $p \in t$.

These angles are stored relative to some baseline direction. Working in 2D, we can take the baseline direction to be the $+X$ axis, as this provides a consistent and valid reference direction over the entire surface. This is more difficult for a 2D surface embedded in $\mathbb{R}^3$, where each angle is in the plane tangent to the vertex, for which there is no globally-consistent baseline orientation. Storing a baseline vector for every vertex would be sufficient, but [**KCPS13**], for example, uses the lowest-indexed edge incident to the vertex to give a consistent baseline vector without incurring additional memory usage.

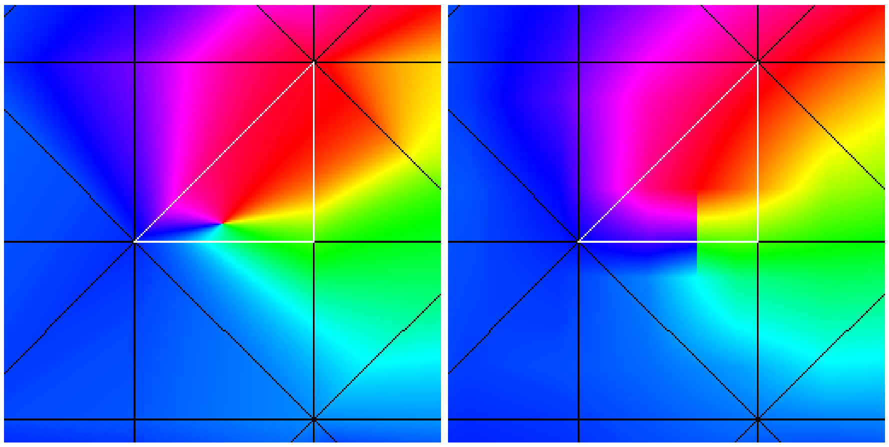

(a) Interpolation from complex values at background mesh vertices (b) Interpolation from intermediate rasterized angles

FIGURE 3.6. A point singularity in the highlighted triangle is exaggerated by naively bilinearly interpolating angles (right), compared to complex-valued interpolation (left). The colors represent the angle $\theta \in [0, \frac{\pi}{2})$, with the sharp transition around the singularity showing both "sides" of the singularity wrapping around in opposed directions.

As mentioned in Section 3.1.2.2, care must be taken when interpolating directions, especially in rotationally symmetric fields. In an attempt to improve the performance of queries into direction fields for our special case of a planar domain, we tried rasterizing the triangle mesh onto a regular grid, where the alignment at each point was stored as a rotation $\theta \in [0, \frac{\pi}{2}]$. This allows very efficient sampling of an arbitrary query point p by much the same process as texture mapping: rescaling p to $p' = (x, y)$ in coordinates relative to the rasterized texture T, use $f_x = x - \lfloor x \rfloor$ and $f_y = y - \lfloor y \rfloor$ as weights for bilinear interpolation between $T_{\lfloor x \rfloor, \lfloor y \rfloor}$, $T_{\lfloor x \rfloor+1, \lfloor y \rfloor}$, $T_{\lfloor x \rfloor, \lfloor y \rfloor+1}$, and $T_{\lfloor x \rfloor+1, \lfloor y \rfloor+1}$. In each of the of the three interpolation steps performed during bilinear interpolation, a check must be performed to determine whether a clockwise or counter-clockwise rotation is the shorter rotation. This works for singularity-free regions, but further distorts cells with a singularity as seen in Figure 3.6.

Instead, we use barycentric interpolation of angles over the triangles of the background mesh, with two different angle representations. For general use, we interpolate complex-valued angles raised to the 4th power, as described in GODF [**KCPS13**]. We also use a scalar-valued angle α in situations where we need to know $\nabla\alpha$, because barycentric interpolation of scalar values results in $\nabla\alpha$ being constant over each regular triangle. This requires extra processing to ensure that singular triangles are handled correctly, so singular triangles are split at the singularity to produce three sub-triangles that can be interpolated over without the possibility of interpolation order causing the discontinuity seen in Figure 3.6.

The general case for querying the direction field is when tracing paths through the field, which we will discuss momentarily. This process has very high spatial locality, with many queries being located in the same background mesh triangle as the preceding query. As the process for computing the resulting angle relies on the barycentric coordinates within the triangle, we start each query by computing the barycentric coordinates within the most recent triangle searched. If all barycentric coordinates are nonnegative, the query point is in the same triangle, and the resulting direction can readily be computed. If there is a negative coordinate, the query point is outside the current triangle, and the most-negative coordinate is used to select one of the three neighboring triangles that share an edge with the current triangle. This repeats with the neighboring triangle, *walking* across the mesh until the correct triangle is found or a boundary edge is crossed. This process depends on a background mesh with convex elements and no interior holes; cases like the airfoil dataset in Figure 4.10 with an internal (wing) boundary fall back to a *hash grid*. A hash grid is an axis-aligned subdivision of the space into small squares, where each square index can be rapidly computed from the query coordinates. Each square contains a list of all triangles it overlaps, so once the correct square is identified a relatively small number of triangle queries needs to be performed to find the containing triangle. The hash grid is the fallback option because it incurs higher memory usage and is typically slower.

3.1.3. Field Lines.

DEFINITION 3.1.7 (Field Line). *A field line is a continuous curve that is tangent to the direction field at every point.*

We consider curves whose movement is restricted to be aligned with the direction field. In these cases, each field line leaving a point represents the one-dimensional collection of points reachable in that direction. This gives an intuitive sense for moving within the direction field, as "following" the field simply means traversing along a field line. Every regular point p in an *n-RoSy* field has n field lines emanating outward from p. We will handle irregular points shortly.

For even rotational symmetries, each field line is bidirectional; for any heading θ, $\theta + \pi$ is symmetrically equivalent. Conversely, odd rotational symmetries exhibit monodirectional field lines. So, for an *n-RoSy* field, if n is odd, there will be n field lines that appear to originate from each point, which are themselves continuations of the n field lines leading into each point. If n is even, there will be $\frac{n}{2}$ field lines that appear to pass *through* each point, as the forward direction of each vector matches a reverse direction of another.

In the case of a *4-RoSy* field, field lines are also perpendicular to the field at every point, meaning that any given regular point will have two field lines passing through it, although it is possible that they are two different sections of the same field line, as seen in the self-intersections in the red paths in Figure 3.7.

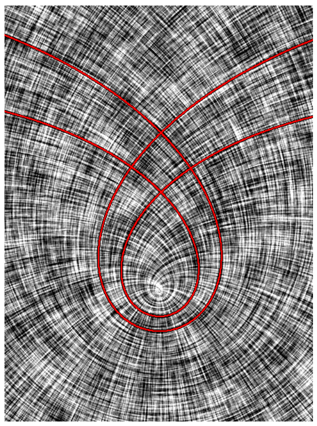


FIGURE 3.7. Two parallel field lines that intersect

As field lines are defined to be field-aligned, they may only cross at angles that are multiples of $\frac{2\pi}{n}$ in an *n-RoSy* field, so when dealing with cross fields, all field lines cross at angles of $\frac{\pi}{2}$. However, when tracing field lines, numerical inaccuracy may cause (degenerate) collisions with angles of 0 or π. These may arise naturally, if a field line naturally forms a closed loop, but they most commonly arise from errors in the tracing process, caused by excessively large step sizes in areas of high curvature. This is a failure case and is an indicator that the chosen parameters are not sufficient for the current cross field; decreasing the step size or regenerating the cross field with a denser background mesh are the most common fixes.

The restriction that field lines must be aligned with the underlying field gives rise to a notion of parallel field lines. A property of parallel lines in Euclidean space works well here: two field lines can be considered parallel if their angles of incidence with a common third field line sum to π radians (unlike parallel lines in Euclidean space, there is no guarantee of equidistance between the lines). It is even possible for them to intersect, although we will strengthen our definition of parallel later on to remove this case.

3.1.4. Singularities. There are several related edge-cases in the concepts presented above. As shown in Figure 3.2d, it is possible to have a continuous map from a region with inherent nondegenerate quadrilaterals, such as the Euclidean plane, and end up with apparent triangles where two edges of the resulting degenerate quadrilaterals are colinear. If a mesh was to be extracted from the checkerboard pattern, the center of the image would have a vertex with only two edges, while all other vertices would have four incident edges. It is as if the corresponding point in the direction field is *missing* two of the four directions. This can even be seen without needing to necessarily extract a mesh to look at the connectivity; simply attempting to evaluate the direction of a point in a direction field can have undefined results. For example, when interpolating in a system where directions are stored as complex values, it is possible for the interpolants to cancel out and result in an undefined direction. These different behaviors are directly related, and the point at which they occur is called a *singularity*.

DEFINITION 3.1.8 (Singularity). *Singularities, also referred to as singular, critical, or irregular points, are isolated points that represent irregular connectivity. In quad meshes, they are vertices that are not incident to four edges or, equivalently, not incident to four quads. In an n-RoSy direction field, a singularity sits at the intersection of some number $\neq n$ of field lines. Generally, points on the region boundaries are not considered to be singular as the full local neighborhood of the point is not known. We follow the convention of labeling these singularities, whether in a mesh or cross field, by their valence: the number of incident edges (in a quad mesh) or field lines (in a direction field).*

Not all complex maps produce direction fields with integer-degree singularities, such as $f(c) = c^{\frac{9}{8}}$, as shown in Figure 3.8. However, as a non-integer-degree singularities produce line discontinuities in the direction field, we restrict our singularities to having integer degrees to ensure the only discontinuities in the direction field are the singular points themselves. This restriction is implicit in our field representation, as we will discuss in Section 3.1.6.

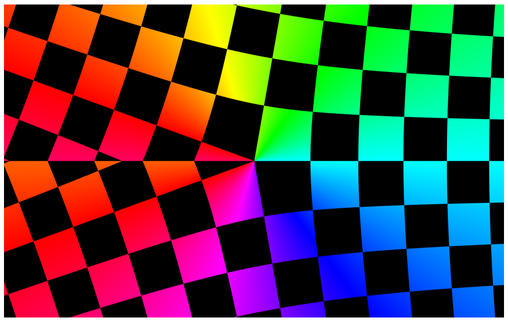


FIGURE 3.8. A singularity with non-integer degree produces a line of discontinuity in the direction field.

For some surfaces, there *must* be singularities in the direction field; this was first demonstrated for a sphere by Henri Poincaré in 1886 [**Poi86**] [2] and is now known as the *Hairy Ball Theorem*, as it demonstrates the impossibility of combing a hairy ball without introducing tufts where the hair sticks out or bald spots where hair around a point is combed in different directions. More formally, any direction field defined on a sphere must have at least two singularities.

3.1.5. Holonomy. *Holonomy* is a term from differential geometry which we will repurpose here in terms of a direction field. In this section, we will consider an *n-RoSy* direction field γ where the angle is always within the range $[0, \frac{2\pi}{n}]$.

DEFINITION 3.1.9 (Holonomy). *Holonomy measures the observed change in a direction field over a closed loop. Given a smooth parametric curve ℓ_t over an n-RoSy direction field γ, such that $\ell_0 = \ell_1$, the holonomy Ω is*

$$\Omega = \int_0^1 \min_{0 \leq i < n} \left((\gamma_{\ell_{t+dt}}) - (\gamma_{\ell_t}) + \frac{2\pi i}{n} \right) dt$$

with $i \in \mathbb{N}$ selecting for the minimum-magnitude rotation that obeys the rotational symmetry of the field γ. The minimization inside the integral serves to correct discontinuities in γ that are not discontinuities in the field: if we represent the field γ by a scalar-valued angle a, immediate changes in a of (integer multiples of) $\frac{\pi}{n}$ are not a rotation but simply a change in its representative within its symmetry equivalence class. These arise from the piecewise definition of the map M, where a single streamline in the field may be an isoline of u in one map and v in another.

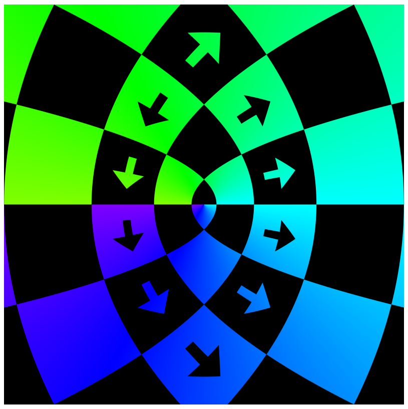
[2] What I can remember of French from high school is not sufficient to work through technical writing, so I cannot confirm if Poincaré singled out and named this specific case.

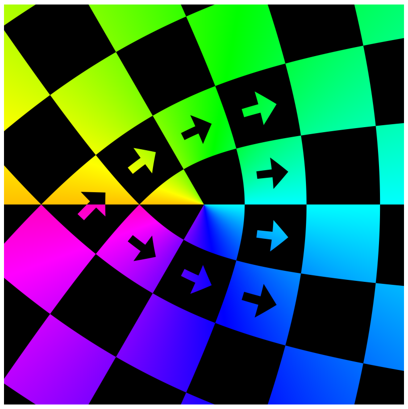
We compute holonomy in our discrete representation by evaluating γ at the vertices of a line-segment loop. The holonomy is simply a running total of the difference in field alignment observed along each *step* across a line segment, accounting for field symmetry. These segments must be sized appropriately to avoid selecting the incorrect minimum for a given step; a step from p_0 to p_1 is too large if it produces a different rotation than the sum of steps from p_0 to their midpoint p_m and then from p_m to p_1. This will happen when the rotation from p_0 to p_1 is greater than $\frac{\pi}{n}$ in either direction; because the angles themselves are only unique to a factor of $\frac{2\pi}{n}$, a larger rotation must be measured by accumulating smaller measurements. By adding additional quadrature points, the measurement is able to capture the actual behavior of the field, whereas the original measurement produced erroneous data, just as sampling a function below the Nyquist frequency yields an incorrect signal.

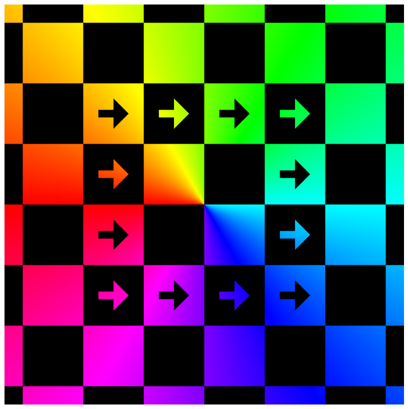
We define the holonomy of a simply-connected region r to be the holonomy of the region boundary. If a region is subdivided into subregions, the holonomy of the original region is the sum of the holonomies of the subregions. We focus on two types of region in particular: the triangles of the background mesh, and regions whose boundaries are everywhere tangent to the field, which we call *tiles*.

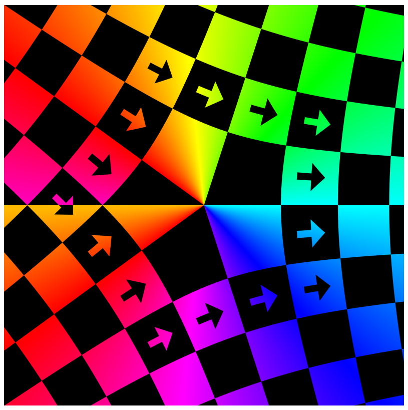
When our regions of interest are closed polycurves consisting of sequences of *field-aligned paths* (which will be further investigated in Section 3.1.3), the holonomy can be calculated by measuring the *turn* along each path; the field-aligned path never deviates from the field by definition, so the holonomy along the path is simply the difference between the headings at the start and end of the path. Importantly, the turn is a difference in *rotations* instead of *directions*, so there are no symmetry conditions and no bounds, as the turn is itself the integral of the difference in direction along the curve, with the rotation of each step too small to trigger symmetry behavior. Figure 3.7 shows two field lines in red that have total turn of almost 2π if viewed as traveling from right to left. In practice we focus on simply-connected tiles, where the maximum rotation magnitude would be 2π from, *e.g.*, an enclosing circle. The field-aligned paths must meet at some angle $\theta = i\frac{2\pi}{n}$, so there is no change in holonomy when transitioning from one field-aligned path to another. The total holonomy of a tile, bounded by a collection of field-aligned curves, is simply sum of the turn along each of the curves.

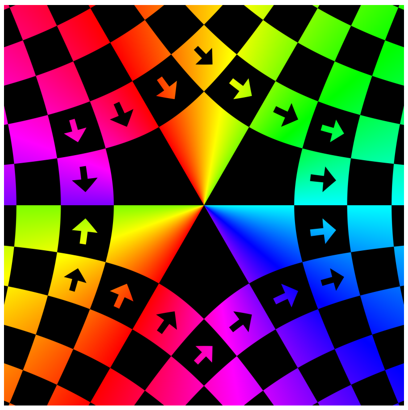
The holonomy of a region is a function of all of the singularities enclosed in the region; this is the Generalized Poincaré-Hopf theorem from [**RVLL08**]. If a region contains no singularities, the holonomy is zero; the rotations of the direction field along any enclosing path balance out to zero. A region containing a single degree-3 singularity will exhibit a $\frac{\pi}{2}$ holonomy; two degree-3 singularities or a single degree-2 singularity would both have holonomies of π. Just like software tests only expose the presence of bugs, but cannot guarantee their absence, this test can only expose singularities; a neutral holonomy does not indicate that a region is singularity-free, because it could contain singularities of opposed magnitudes whose holonomies cancel out.


(a) $f(c) = c^{4/2}$
Degree-2 Singularity; $\Omega = \pi$


(b) $f(c) = c^{4/3}$
Degree-3 Singularity; $\Omega = \frac{\pi}{2}$


(c) $f(c) = c^{4/4}$
No Singularity; $\Omega = 0$


(d) $f(c) = c^{4/5}$
Degree-5 Singularity; $\Omega = \frac{-\pi}{2}$


(e) $f(c) = c^{4/6}$
Degree-6 Singularity; $\Omega = -\pi$

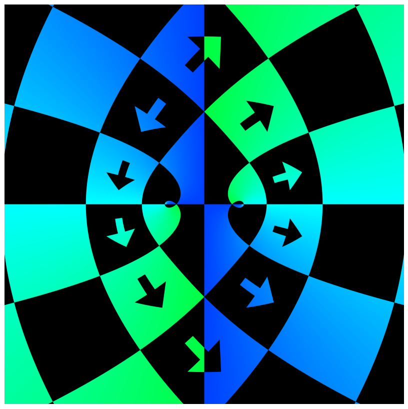

(f) $f^{-1}(c) = \big((c+6)(c-6)\big)^{1/4}$
Two Degree-3 Singularities; $\Omega = \pi$

FIGURE 3.9. Visualization of the holonomy of singularities of various degrees using complex functions that induce each singularity. The arrows demonstrate a closed path where the eventual inconsistency in the arrow alignment corresponds to the degree of the enclosed singularity, or their sum as seen in (f). Note that (f) is given in terms of the inverse function, which was used to generate the image, in order to highlight the two singularities which are complex roots.

3.1.6. Singularity Detection via Holonomy. Although immediately apparent to a human observer when looking at a LIC rendering of a direction field, such as Figure 3.5c, singularities are not explicit in the direction field representation, and a search must be performed over the entire region to locate them. As described in the previous section, singularities can be detected within any closed region by examining the *holonomy* of the region boundary. Figure 3.9 illustrates this process: the degree of the singularity at the center of the image can be identified by walking a path encircling the singularity, shown here as arrows drawn over the squares. The squares are drawn such that they align with any arrows drawn in adjacent squares, but in all images with a singularity present there is one pair of arrows that do not align; the holonomy of the region is the rotation between the two unaligned arrows. Note that the paths in Figures 3.9a and 3.9f have the same holonomy; a nonzero holonomy indicates a singularity is present but cannot distinguish between a single degree-2 or two degree-3 singularities.

When detecting the singularities in our computed direction fields, we simply measure the holonomy of every triangle in the mesh. Because we are computing the holonomy of triangles, we only have three sample points; any additional sample points would just be interpolations of the angles at the vertices and would provide no new information. For triangle t_{012}, the holonomy is the sum of the three transitions: $v_0 \rightarrow v_1$, $v_1 \rightarrow v_2$, and $v_2 \rightarrow v_0$. In a *4-RoSy* field, the maximum difference between two angles is $\frac{\pi}{4}$; if the difference was greater, one of the symmetric directions would be closer, resulting in a smaller difference. The same logic is why angles in Euclidean space can differ by at most π. So, by adding three values, each $\theta_i \in [-\frac{\pi}{4}, \frac{\pi}{4}]$, the sum must be $-\frac{3\pi}{4} < \sum_i \theta_i < \frac{3\pi}{4}$. However, as this sum must be a multiple of $\frac{\pi}{2}$, there are only three potential holonomies for each triangle, 0 and $\pm\frac{\pi}{2}$.

Thus, by using a triangular background mesh, detectable singularities are limited to degree 3 and 5, and there can be at most one per triangle. Figure 3.10 demonstrates what happens when a direction field with a degree-6 singularity gets encoded as rotations at the vertices of a triangle mesh: the holonomy from the single degree-6 singularity is pushed into neighboring triangles, where the central degree-6 singularity has been reduced to degree-3, and is now flanked by three degree-5 singularities in each of the adjacent triangles. So, an attempt to encode a direction field with higher-degree singularities, or simply too many singularities close together, will cause the holonomies of individual triangles to shift such that singularities maybe be merged, split, or moved. The resulting field is still a valid direction field, but it has a different set of singularities.

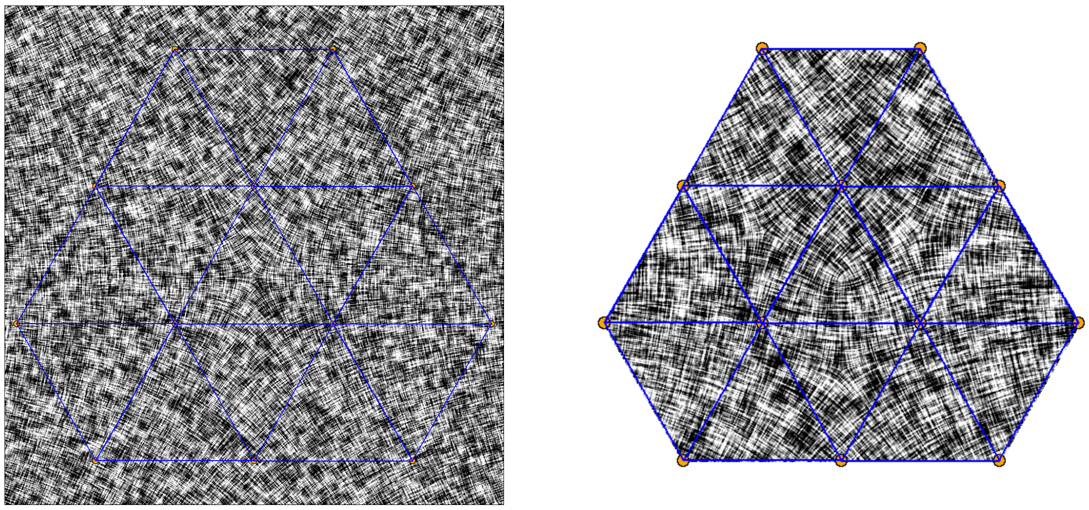


FIGURE 3.10. Direction field with a degree-6 singularity with a triangle mesh overlaid (left). Directions are sampled at the mesh vertices, from which a new direction field can be interpolated (right). Note there are now 4 singularities in the right image.

3.1.7. Singularities and Curvature. In the plane, a circle with radius 1 will have circumference 2π. Equivalently, the angle covered by sweeping all the way around a point is 2π. This does not hold for a curved surface. Sweeping around a point with *positive Gaussian curvature*, such as any point on a sphere, will sum to less than 2π, while a point with *negative Gaussian curvature*, like a point in a saddle, will have a sweep angle of more than 2π.

Every point in the Euclidean plane has a sweep angle of 2π. Singularities in a cross field, incident to some number $\neq 4$ of field lines, appear to have a sweep angle $\neq 2\pi$, as all field lines logically meet at $\frac{\pi}{2}$ angles. Singularities in a cross field imply curvature, as if the cross field was mapped from a curved surface, because the number of incident squares differs from 4. This is as if any observed deviation of the angle between two streamlines at a singular point from $\frac{\pi}{2}$ is due to perspective error while viewing a perfect square rising out of (or into) the plane.

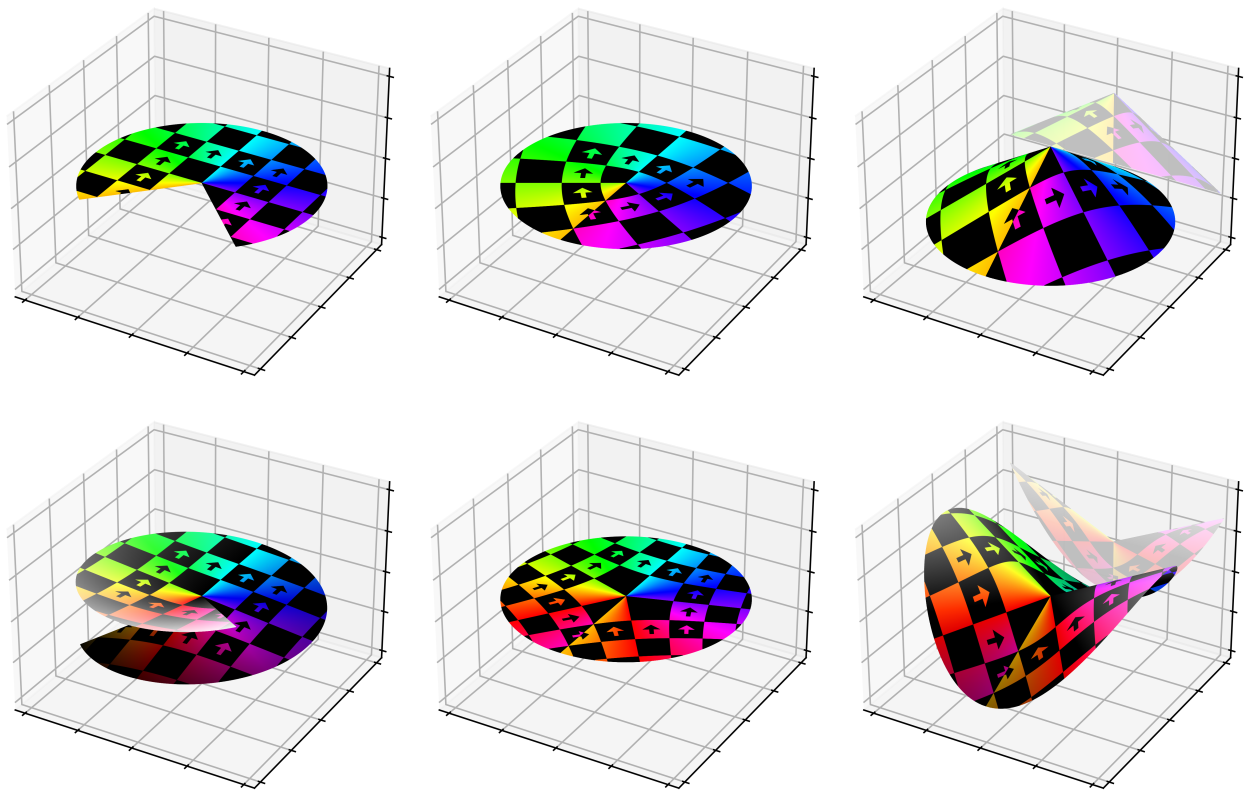


FIGURE 3.11. Visualization of the relationship between singularities and surface curvature. In both rows, a portion of the unit disc (left) can either be deformed into a flat disc (center) or instead *folded* into a surface embedded in 3D. Interestingly, no points other than the central singularities demonstrate any curvature.

Figure 3.11 shows how singularities can be thought of as indicating curvature. Instead of simply deforming the partial disc into a complete disc, slightly distorting the space to close up the disc, the partial disc can be bent into a cone (or saddle) in a way that does not deform the original space.

This somewhat paradoxical concept of a planar surface with curvature is related to the size function that we are attempting to approximate and will be investigated further in Section 3.2.5. However, as can be seen in Figure 3.9, there is an apparent increase in the size of quads when moving away from a degree-2 or degree-3 singularity, and similarly, a decrease in the size of quads moving away from a degree-5 or degree-6 singularity, with the size change being more drastic with degrees further away from 4, or with increasing holonomy (and consequently curvature). This connection between quad size and curvature is how we use approaches based on surface curvature in order to achieve our target size function.

3.1.8. Separatrices.

DEFINITION 3.1.10 (Separatrix). *A separatrix is a field line that begins or ends at a singularity. Every non-terminal point along the separatrix is non-singular; a separatrix cannot have a singular point along its path. Instead two distinct separatrices would meet at that singularity.*

In this context, we consider separatrices that pass through even-degree singularities to be distinct separatrices that meet at the singularity; a valence-6 singularity is adjacent to six separatrices but only three field lines. Because we are focusing on cross fields, and the field lines are bidirectional, we take the perspective that separatrices originate at a singularity and are directed outward. While it is possible for a separatrix to travel between two singularities, and as we will see in Chapter 5 it is even beneficial, it is highly unlikely for the separatrices to exactly align, so we ignore that case for now.

All of the separatrices can be combined to form the *separatrix graph*. The vertices in this graph include all of the singularities in the field, simply because any missing singularity would mean that a separatrix has been omitted, as well as any point where two separatrices cross. These crossing points, being in the interior of a separatrix, must be regular points, so in our case of a *4-RoSy* field, all separatrices cross at right-angles. The separatrix graph divides the space into singularity-free regions, because all singularities are already vertices in the graph, and thus are restricted to region boundaries. These regions are all four-sided, as a lack of interior singularities means that the holonomy of the region is zero, and any other number of sides would indicate a different holonomy. We call these regions *simple tiles*, which we will revisit in Section 5.0.1.2.

DEFINITION 3.1.11 (Simple Tile). *A simple tile is a region bounded on all sides by field lines, with no singularities in the interior of the region.*

This construction allows us to complete our definition of parallel. Considering a simple definition of parallel lines to be that they never intersect, our definition is based on how strongly we can guarantee that they do not intersect. An intersection between two field lines that are mutually perpendicular to a third implies the creation of a three-sided field-aligned region, which implies a nonzero holonomy and thus the existence of a singularity somewhere between the three field lines. However, if we know that a region is singularity-free, such as a simple cell in the separatrix graph, it would not be possible for the portions of the two field lines within the region to be perpendicular to a third and also intersect. We call this *locally parallel*, where only the portions of the field lines within the singularity-free region are guaranteed to not intersect.

Any field line entering one face of a simple tile *must* leave via the opposing face; changing direction enough to collide with either of the other two faces at a right angle would imply a non-zero holonomy and therefore a singularity on the interior of the region, while colliding with a zero angle implies the field line *is* one of the bounding separatrices, and no other intersection angles are possible as both the region boundaries and field lines are strictly field-aligned.

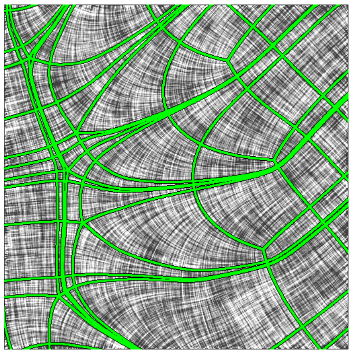


FIGURE 3.12. Separatrix graph for a region with right-to-left growth, with valence-5 singularities on the left and valence-3 on the right.

Given the full separatrix graph, we can also define a third, even stronger definition of parallel. Two field lines, not just segments of them, are *globally parallel* if they are locally parallel somewhere and no singularities are ever present between them. In a simple tile in the separatrix graph, there are two sets of mutually-orthogonal field lines, with each set entering and leaving the tile through a different pair of opposed edges. All of the field lines in one of these sets are locally parallel, and cannot be separated by a singularity; otherwise, one of that singularity's separatrices would separate the two field lines and would have subdivided the simple tile in the separatrix graph into two smaller simple tiles. Therefore, every simple tile in the separatrix graph contains two orthogonal sets of globally-parallel field lines. In this sense, the separatrix graph completely defines the global behavior of the direction field.

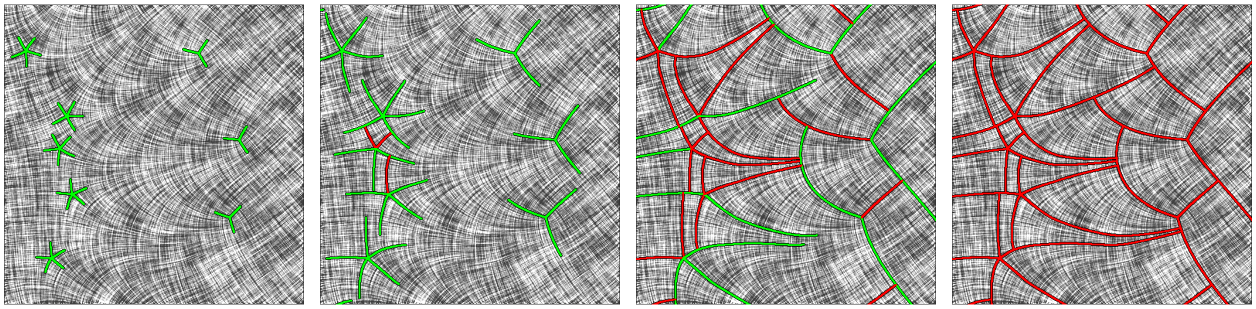


FIGURE 3.13. A series showing the evolution of the motorcycle graph for the same field as shown in Figure 3.12. Green paths indicate live motorcycles, while red paths indicate crashed motorcycles and therefore complete regions of the graph.

3.1.9. The Motorcycle Graph. An alternative to the separatrix graph is the *motorcycle graph* [**EGKT08**], named after its similarity to the Light Cycle game from the 1982 movie *Tron*. Instead of separatrices being traced out in their entirety, "motorcycles" are driven out from each singularity along each separatrix path. Their paths are traced out simultaneously, and whenever one crosses an existing path, it explodes, leaving a

T-junction in the graph; the path for this motorcycle terminates at the point of collision instead of continuing along the surface. Figure 3.13 shows this process, with green paths indicating "live" motorcycles and red paths indicating the associated motorcycle has crashed so the segment is complete. Figure 3.12 shows that the separatrix graph for the same area is much more complicated. The motorcycle graph similarly divides the region into rectangular, singularity-free regions, but the bundles of field lines within each region are only guaranteed to be locally parallel. Within each region, the T-junctions are explicitly marked to differentiate those intersections from a degenerate vertex with a $180°$ opening, such as the one at the center of Figure 3.9a that only has two incident edges. Each rectangular region will only have four corners (with $90°$ internal angles) but may additionally have vertices corresponding to outward T-junctions along the edges (with $180°$ internal angles).

The benefit is that the motorcycle graph is typically drastically simpler than the separatrix graph, with a separatrix only being "live" immediately around its originating singularity. This simplifies the separatrix graph while preserving the property that the region has been subdivided into *simple tiles*, and for this reason the motorcycle graph is used both for the basis of mesh extraction in Chapter 5, as well as for visualization in this chapter.

3.1.10. Path Tracing. There are performance and accuracy concerns involved with the process of tracing field lines, especially around singularities. It is sufficient to move in *steps*, sampling the direction at a point and moving a fixed distance in the sampled direction. Taking shorter steps increases the accuracy of the resulting field lines; short steps are most helpful in areas of high curvature, so we make sure to reduce our step size whenever near a singularity.

The more pressing issue, however, is that an excessively long step length might incorrectly align with one of the orthogonal directions instead. Directions in an *n-RoSy* field that deviate by $\frac{2\pi}{n}$ are considered equivalent, so there is no way to distinguish between $+\frac{\pi}{n}$ and $-\frac{\pi}{n}$ rotations. For example, taking two steps in an *4-RoSy* field that turn to the left $\frac{\pi}{6}$ is safe, but when taken in a single step instead, the direction would "overflow" and appear to be a *rightward* $\frac{\pi}{6}$ turn.

Singularities represent the most drastic change in field direction, so we try to avoid sampling close to a singularity whenever possible.

3.1.11. Line Integral Convolution. The *Line Integral Convolution* is a visualization technique that is used to produce intuitive renderings of direction fields, with streaked patterns that are aligned with the direction field [CL93]. The image is constructed by sampling a source image of random noise. For each pixel in the output image, n field lines are traced (along the *n-RoSy* field) outward from the pixel. The random noise is sampled along these field lines, and the value for the pixel is the average of these samples. For our cross fields, 4 field lines are traced from each sample pixel, generating the characteristic cross-hatch pattern that shows the two orthogonal directions at each point.

To explain the pattern in the LIC image, it is helpful to look at the neighborhood around any pixel in the image. Figure 3.14 shows two pairs of close sample points, with the first pair sitting on a common field line, shown in red. While field lines have zero width, the noise image is being sampled along the paths, giving them an effective width of one pixel. If a neighboring pixel sits along a common field line with the source pixel, we would expect them to be similar, as they will both have sampled along this same path; other than a few samples at the far ends of the field line, both points will have sampled the same values along the path, so the only variance between the two points comes from the contents of the two disjoint green paths. If the two pixels do not sit along the same field line, then they should differ drastically, as they are sampling along different paths, with only two short regions of overlap between them. By roughly doubling the number of independent random samples, the unaligned pixels will have approximately double the variance in their sampled values. It is this difference in variance that produces the pattern that is visibly always aligned with the field. Even if the pixels are not perfectly aligned with the field line, their similarity will be proportional to the overlap between the two parallel field lines between them.

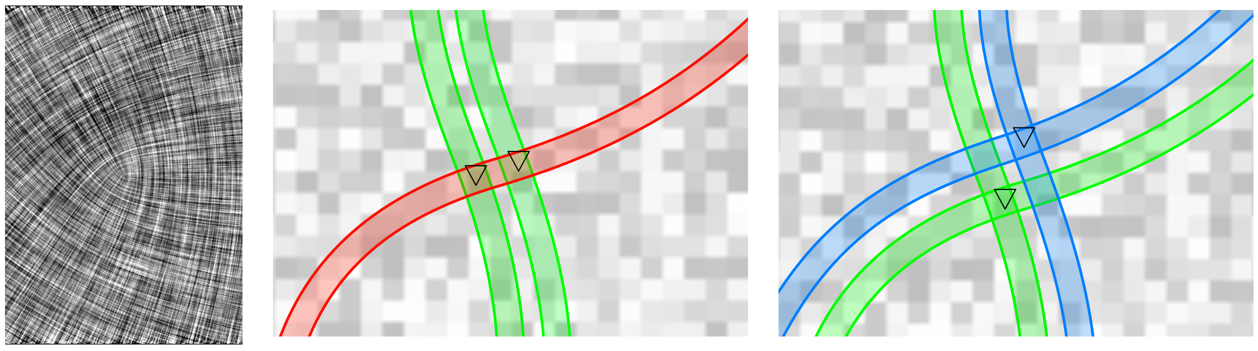

(a) Sample LIC Image (b) Aligned Sample Points (c) Unaligned Sample Points

FIGURE 3.14. LIC image construction. By sampling along field lines, neighboring points (b) that are on a common field line (red), will tend to be similar in the output image, as half of their sampled values are identical. Unaligned points that do not share a field line (c) differ drastically in the output image, as they only share two small regions of overlap between otherwise disjoint paths.

With random samples in the range $[0, 1)$, the expected value of any of pixel is 0.5. Producing LIC images by using these values directly as pixel brightness results in muddy images that are difficult to parse, as the vast majority of pixels fall within a fairly narrow band. We instead rescale the values to better observe the variance in the sums. Scaling according to the minimum and maximum values over the image can introduce noticeable brightness and contrast changes between images as the random sampling process will produce highly variable extrema. For consistency we scale to the fixed range $[\frac{4}{9}, \frac{5}{9}]$ for all LIC images seen here, clamping values outside the range.

There is no special treatment for singularities when generating a LIC image; they are apparent to observers simply because we notice the abnormal behavior of the field lines around the singularity.

3.1.12. Motorcycle Graph Numerical Stability. In an effort to avoid problems that occur when sampling near a singularity, which is precisely the case for the initial portion of every motorcycle's path when generating the Motorcycle Graph, we start this process by tracing *inward* toward each singularity. Starting along the perimeter of a disc centered around the singularity, we search along the perimeter in order to find the inward field line that comes closest to approximating each separatrix, which is simply the field line that passes closest to the singularity. From a point on the disc boundary we can trace inwards, following the field line until it passes the singularity. We then perform an orientation test, to see which side of the singularity the field line passed. This allows us to perform a binary search, keeping the two paths on either side of the singularity and choosing the midpoint on the arc connecting the seed points as the starting point for the next field line. After a fixed number of iterations, the next midpoint generates a field line that is saved as the initial portion of the separatrix, and a motorcycle is saved at the starting point on the disc, except facing outward, ready to begin the motorcycle graph construction.

Generally, motorcycles all move at the same rate; biasing motorcycles, such as having them move at different speeds, would simply change the precedence of which motorcycle "lives" through any collision in the separatrix graph, when the most important information is that the intersection happened in the first place. Keeping all motorcycles at the same speed ensures that each motorcycle is likely to be active only in the areas where it is most important, specifically in areas near its originating singularity.

In areas of high curvature, an oversized step may induce an erroneous $90°$ turn in the motorcycle's path from the actual path, which will eventually lead to a 3- or 5-sided region in the motorcycle graph. These irregular tiles may be handled directly, which we discuss in Section 5.1.6, but we prefer to prevent those regions from forming in the first place. Reducing the size of steps that the motorcycles take in areas of high curvature is one possible solution, but it is not sufficient to only reduce the step size near the originating singularity by, *e.g.*, starting the motorcycles off with a reduced step size that increases as it travels; motorcycle paths also need to be accurate where they terminate, especially if they terminate near another singularity. A partial solution is to *oversample* motorcycle paths, keeping a uniform step size and some large fraction of the samples, so the resolution of the resulting polyline is much coarser than the steps that were taken to produce it.

It is possible for a motorcycle M_1 from a singularity S_1 to come arbitrarily close to any other singularity S_2, but for that to happen, it must be traveling locally parallel to a separatrix created by some motorcycle M_2 from S_2. In this case, the path of M_1 is not as important as ensuring that it impacts the correct separatrix; the step most likely to cause a "wrong turn" (switching to a different phase of the cross field) due to curvature will be near S_2, and therefore likely to be over an existing separatrix from S_2. In this case, testing to ensure motorcycles collide with existing field lines at right angles can serve as good warning, as a motorcycle colliding with a parallel field line is an indication that the step size is too large.

3.2. Size

Now that we have covered the basics of cross fields and have explored the parallels between cross fields and quad meshes, we will now investigate size: how cross fields and quad meshes both imply metrics, and how a metric causes shortest-length paths to curve. Our primary contribution in this chapter is the realization that the curvature of these paths resembles surface curvature when computing cross fields on manifold surfaces embedded in 3D, which allows us to take advantage of surface meshing techniques to achieve size-driven meshes. In particular, we build off of the approach from *Globally Optimal Direction Fields* [**KCPS13**], which generates cross fields from the curvature of manifold surfaces and will be explored further in Section 3.3.

3.2.1. Size-Aware Meshing.

DEFINITION 3.2.1 (Size Field). *A size field σ is a scalar function that defines a real-valued size σ_p at every point p of the domain, which will typically be the Euclidean plane, but could be a closed surface $\mathbb{S}$: $\sigma : p \in \mathbb{S} \mapsto \sigma_p \in \mathbb{R}$. Size is a one-dimensional quantity, in our case corresponding to the desired edge length in the produced quad mesh. We say a mesh is compatible with a given size field if the length of each edge is within some bounds of the size values sampled along it.*

Edge length serves as a one-dimensional metric used to express the relative size of nearby elements in the mesh. This metric is not precise as its primary purpose is comparing *grading*, or how the relative sizes of elements change over the mesh, which will indicate relative element densities; ideal element densities depend on the function that the mesh is being created to simulate. This metric translates nicely across a variety of environments; it is useful in both two and three dimensions, as it follows the square root of quadrilateral area and the cube root of hexahedron volume for ideal, nearly-square elements. It conveys the scale of mixed-element meshes without needing to consider the differences between the shapes, areas, or volumes of the different elements. A strict measure of edge lengths, such as taking the average lengths of the edges within a quad, would have discontinuities moving from quad to quad, and is not particularly helpful. Instead, edge lengths serve as a mental model for assessing whether a mesh is compatible with a requested sizing field, as well as how the size of elements change over a mesh.

DEFINITION 3.2.2 (Grading). *Also referred to as telescoping, grading is the intentional change in element size over a region of a mesh. It is often used to increase mesh resolution in areas of interest, in effect prioritizing those areas at the expense of others where the decreased resolution is acceptable. Grading may be necessary when running a simulation with a sufficiently-dense constant-size mesh would be prohibitive due to time or memory constraints.*

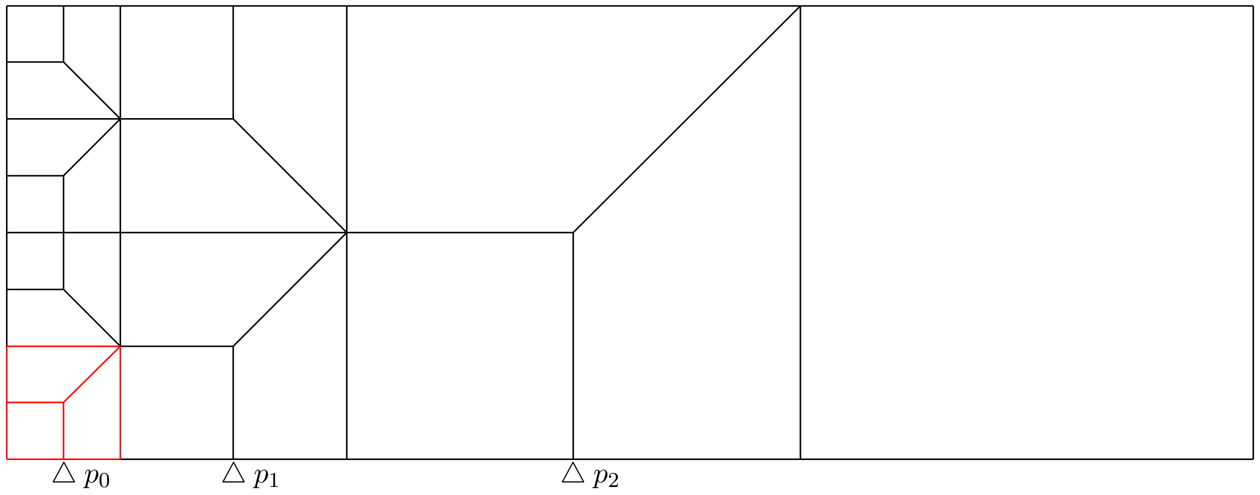


FIGURE 3.15. A copy of Figure 1.2, showing the grading of a common growth template. All three marked points have three incident edges of the same length, giving a consistent value for the edge-length metric at these points. Note that the distance from p_0 to p_1 is three times the edge length at p_0, and similarly the distance from p_1 to p_2 is three edge-sized steps from p_1. While the lengths of the edges are doubling, so are the distances between sample points (the edges themselves), showing that the edge lengths are growing linearly, or with a constant grading.

If mesh elements are required to be nicely-shaped, the grading within a mesh is at most linear; the change in observed edge length is proportional to the distance traveled. This implies that the size functions we target should be Lipschitz-continuous, with the Lipschitz constant chosen to balance between growth rate and element quality. Intuitively, the distance between opposing edges within a quad varies linearly across the quad, effectively being a linear interpolation of the other two edges. By imposing quality metrics on the quads, and avoiding extreme internal angles and aspect ratios, we expect any sequence of quads to be roughly self-similar. Growth can be achieved by repeating patterns of quads, such as the example shown in Figure 3.15, where the edge length doubles with each repetition of the template. The template itself has singularities of degree 3 and 5, but the repeated application of the template can create higher degree singularities, such as the degree-6 singularities seen here.

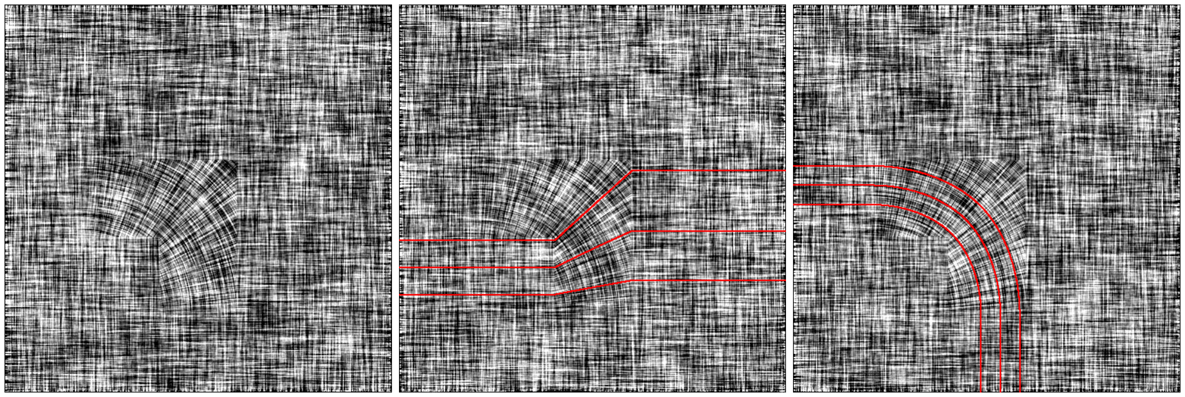

(a) 3-5 growth pattern (b) Diverging field lines (c) Redirected field lines

FIGURE 3.16. A LIC rendering of a field with two singularities (left), corresponding to the common growth pattern shown in Figure 3.15. The growth is apparent in the divergence of field lines (center), which depends on other field lines being diverted out of the way (right).

Grading can also be achieved without singularities by *divergence* in the field lines, where relative growth in the resulting edge lengths is proportional to the relative increase in distance between the diverging field lines. This can be seen in Figure 3.21. Our goal is to utilize both sources of grading to produce a quad mesh whose edge lengths are compatible with an input size field σ.

While there is no inherent scale to a cross field, each pair of parallel field lines represents a potential chord of quads, so if the field lines diverge, the potential edges across the chord must lengthen as well. Ideally, two globally-parallel field lines should always be the same distance apart according to σ, so sampling the distance between them and dividing by σ at the sample location should result in a constant value; the more consistent those values are, the better the cross field matches the desired grading from the sizing field.

Exploring the relationship between a given cross field and the set of compatible size fields is the topic of Chapter 4.

For example, in Figure 3.16b, the red field lines are globally parallel; while the height of the bundle obviously differs from one side of the image to the other, the bundle should always have the same intrinsic height, when measured in terms of the size field. Considering this field was derived from the template seen in Figure 3.15, where the field lines travel between two sets of quads that differ in size by a factor of two, the fact that the height of the bundle doubles over that same span implies that the cross field in Figure 3.16 is compatible with the size field implied by Figure 3.15.

In Section 3.4, we use this technique of measuring the distance between globally-parallel field lines to evaluate how well a cross field matches the desired growth in a size field.

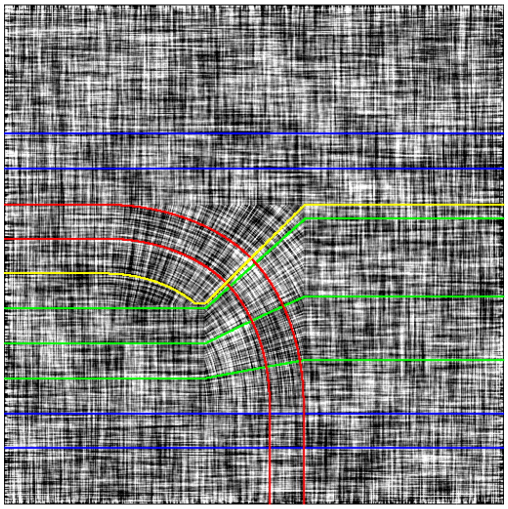


FIGURE 3.17. Equally-spaced field lines moving through the same cross field as Figure 3.16. Blue field lines are not deflected. Red field lines are redirected to allow the green lines to diverge. The yellow field line is an artifact of the synthetic field, and is globally parallel with the green lines.

Diverging chords pose an interesting problem that is illustrated in Figure 3.16, which shows a synthetic cross field designed to mimic the behavior seen in Figure 3.15. As diverging chords get wider, it takes fewer of them to cover an area. However, as is often the case, the chords are prevented from fanning out by constraints, in this case by the regions of the field that are perfectly horizontal along the top and bottom of the region. So, in order for some chords to grow (Figure 3.16b), other chords must be diverted (Figure 3.16c) to make space. Singularities are the mechanism by which bundles of field lines are redirected, pulling chords over each other (for positive holonomy singularities; degree < 4), or apart from each other (with negative holonomies; degree > 4), leaving room for other bundles to diverge and expand to fill the space. All of these paths are shown simultaneously in Figure 3.17, with the yellow path being particularly instructive because it represents the transition between the red and green field lines; it is clearly near a separatrix (not shown) for the degree-3 singularity, but there is also some distortion due to the fact that this is a synthetic field designed to mimic the growth of Figure 3.15.

As we noted in Figure 3.9, there is a correlation between the sign of the singularity and the sizes of the quads. Singularities with degree less than 4 have relatively small quads near the singularity, increasing in size in proportion to the distance to the singularity, with singularities with degree greater than 4 showing the reverse relationship. Near any one singularity, the growth rate is uniform in all directions; of particular importance is combining *opposite* singularities in order to control the direction of the growth, as seen in Figure 3.15 where degree 3 and 5 singularities are arranged in a pattern to force a left-to-right growth rate. As seen in Figure 3.9f, singularities can be combined to achieve arbitrary growth rates, with denser clusters of singularities effecting faster growth. However, as the singularities themselves will become vertices in any resulting mesh that follows the cross field, this imposes a limit on the resulting edge lengths, which cannot be longer than the distance between singularities. This implies that producing sparse meshes is a harder problem than producing dense meshes, as dense meshes allow for dense clusters of singularities, allowing for stronger grading, which in turn gives better control over sizing overall.

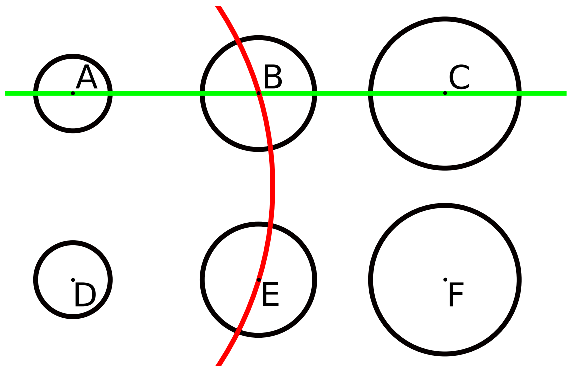


FIGURE 3.18. Six points, each surrounded by a unit circle in the metric implied by a varying size function that increases to the right. Two geodesics are shown: the geodesic in green (between B and C) is parallel to the gradient of the size function, and remains straight, while the geodesic in red (between B and E) is warped by the size function, bowing outward in the direction of increased size.

3.2.2. Geodesics. Figure 3.18 illustrates the relationship between a non-uniform size function and the induced curvature that we take advantage of. There are six points that are grid-aligned in *paper-space*, meaning they are equidistant when measured on the page, but there is an associated metric on the surface that distorts distances, so $|AD| > |BE| > |CF|$. This distortion changes the shape of geodesic curves; because space is "bigger" on the right, a geodesic will bend rightward because it can make more progress in the larger metric. The red geodesic is strongly distorted, while the distortion has no effect on the green geodesic because it is parallel to the size function gradient. This bending of space appears to behave just like geodesics on a curved surface like a globe, (although, on a globe, the metric is constant): two points with equal latitudes will share a geodesic that bends away from the equator, while two points with equal longitudes will have a "straight" geodesic along the longitude. That is, the curved space with a constant metric has similar geodesics as our planar system with a varying metric.

Note that we have been seeing geodesic curves throughout this chapter, as Figures 3.2, 3.3, 3.5 and 3.9 all show quads with distinctly curved sides after they have undergone some transformation. In these cases, these edges are all geodesics, as they travel along the shortest path in the new metric induced by the transformation. As the new metric increases or decreases outward from the center of each example, the lines *along* the gradient (through the center of each image) all remain straight, while all other lines appear to bend in proportion to their alignment with the gradient, demonstrating the same pattern as shown in Figure 3.18.

3.2.3. Riemannian Surfaces.

DEFINITION 3.2.3 (Riemannian Surface). *A Riemannian Surface is a surface with an associated metric g that is used to measure lengths on the surface. When viewed as an extension of the inner product, the standard inner product $a^T b$ becomes $a^T gb$, with g taking the form of a $n \times n$ symmetric positive definite matrix called the metric tensor.*

Given that we already have a desired size function σ defined over our triangulated mesh M, we can define the metric tensor as:

$$g = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}$$

We can see that this scales the length of any vector $v = (x, y)$ by σ as expected:

$$|v| = \sqrt{v \cdot v} = \sqrt{\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}} = \sqrt{\sigma^2 x^2 + \sigma^2 y^2} = \sigma \sqrt{x^2 + y^2}$$

3.2.4. Einstein Notation and Covariance. The following sections rely on *Einstein Notation* for some equations, which is a technique for succinctly expressing tensor operations developed for the Theory of Relativity. An in-depth exploration of the topic is beyond the scope of this section (and dissertation!); we refer readers to Benjamin Crowell's Special Relativity [**Cro21**] for as approachable of an introduction as the topic permits. But, we will attempt a summary that will be sufficient for the material that follows.

3.2.4.1. *Explicit Index Manipulation.* The core of Einstein Notation is specifying interactions between vectors, matrices, and multidimensional arrays, using subscripts and superscripts to indicate which indices interact.

In linear algebra, operations between vectors and matrices are implied by their ordering within an equation. Einstein Notation makes these pairings explicit, by using free variables as subscripts and superscripts (we will discuss which is used when momentarily) to mark which indices are iterated over to perform an operation.

Taking a vector rotation as an example, where the vector v is multiplied by the rotation matrix R, resulting in $u = Rv$, which actually implies:

$$u_1 = R_{11}v_1 + R_{12}v_2$$
$$u_2 = R_{21}v_1 + R_{22}v_2$$

This would be expressed in Einstein Notation as:

$$u^i = R^i{}_j v^j$$

we do not need to explicitly state that u and v are column-vectors as the columns (the second index) of R are explicitly tied to v by sharing the same index variable j.

This is particularly helpful when working with arrays of 3 or more dimensions, such as the 2×2×2 array Γ which we will encounter shortly.

3.2.4.2. *Covariant and Contravariant Values.* Tensors are a particularly important class of arrays. A defining property of tensors are their *transformation laws*, which define how a tensor must change in response to a change of basis in order to represent the same physical quantity. In particular, we want the results of certain operations to be invariant to coordinate changes, and this is achieved by defining how the arguments of those operations change with respect to a change in coordinate system.

Contravariant vectors react to a change in basis by undergoing an opposite change. 2D displacement is a good example: the vector $(1, 2)$ measured in meters becomes the vector $(0.001, 0.002)$ when measured in kilometers. By changing the spatial scale, the spatial basis vectors $(1, 0)$ and $(0, 1)$ are essentially being multiplied by one thousand, while contravariant values expressed in that basis get divided by one thousand in order to retain the same physical meaning.

Covariant vectors, on the other hand, react to a change in basis by undergoing the same change. What is the work $W = F \cdot d$ performed by a 10-Newton force in the $+x$ direction over the example displacement $d = (1, 2)$? If we represent the force $F = (10, 0)$, a simple dot product gives us $F \cdot d = (10, 0) \cdot (1, 2) = 10$. Work is invariant to the coordinate system, so if we change the coordinate system from meters to kilometers, we can work backwards from $10 = F \cdot d = F \cdot (0.001, 0.002)$ to show that F needs to be $(10000, 0)$. We're not changing the force F, as the units now are $Kg\ mm\ s^{-2}$, but it is perhaps more helpful to think of the operation $F\cdot$, which takes a vector quantity (displacement) and returns a scalar quantity (work), as having transformed to preserve the resulting scalar as a basis-invariant quantity. Thus, the operation $F\cdot$ is covariant, transforming *with* a change in basis, to counteract the opposite change that the displacement d undergoes.

Contravariant vectors cover most of the "standard" vector quantities we're used to in Linear Algebra, so the contravariant label is often omitted. Covariant vectors are often shortened to covectors. Einstein Notation uses the *placement* of indices to indicate the basis for that index: superscripts denote contravariant indices, and subscripts denote covariant indices.

The *metric tensor*, introduced in Section 3.2.3, is a covariant quantity. However, if we want to maintain the invariant of the metric under a change of basis, The metric tensor changes with the *square* of the basis change, because $g_{ij}v^iv^j$ is the *squared* length of v. Recall that the length of a vector v in a metric g is $|v| = \sqrt{v^T gv}$, so measuring the length of a vector $v' = (\frac{x}{c}, \frac{y}{c})$ after a basis change c yields the original length, unchanged:

$$|v'| = \sqrt{v' \cdot v'} = \sqrt{v'^T g' v'} = \sqrt{\begin{bmatrix} \frac{y}{c} & \frac{x}{c} \end{bmatrix} \begin{bmatrix} c^2\sigma^2 & 0 \\ 0 & c^2\sigma^2 \end{bmatrix} \begin{bmatrix} \frac{x}{c} \\ \frac{y}{c} \end{bmatrix}} = \sqrt{c^2\sigma^2 \frac{x^2}{c^2} + c^2\sigma^2 \frac{y^2}{c^2}} = \sigma\sqrt{x^2 + y^2}$$

Additionally, the metric tensor gives us a method of converting between covariant and contravariant bases:

$$v_i = g_{ij}v^j$$
$$v^i = g^{ij}v_j \text{ where } g^{ij} = (g^{-1})_{ij}$$

3.2.5. Parallel Transport Across Mesh Edges. On Earth, we are accustomed to the fact that the shortest path between any two points is along a *geodesic*, which in the context of global navigation is an arc of a *great circle* around the globe. When plotted on a map, the geodesic will likely appear to curve, not because the path turns but because the directions themselves are changing as progress is made along the path, due to the distortion incurred by squishing a curved surface onto a flat map. Nonetheless, when navigating on the globe or the map, care must be taken to distinguish between whether the surface or the path is curving.

Parallel transport is the mathematical formalization of the idea of removing the influence of surface curvature, so we can determine how much the path itself curves. Geodesics are the "straight" rulers against which paths are measured and are also the means of continuing a path along an arbitrarily curved surface.

If we take the size function given by the metric tensor, changes in the metric induce curvature in the Riemannian surface, which then induces curvature in the geodesic curves. We will now explore the paths of the geodesics resulting from the varying metric.

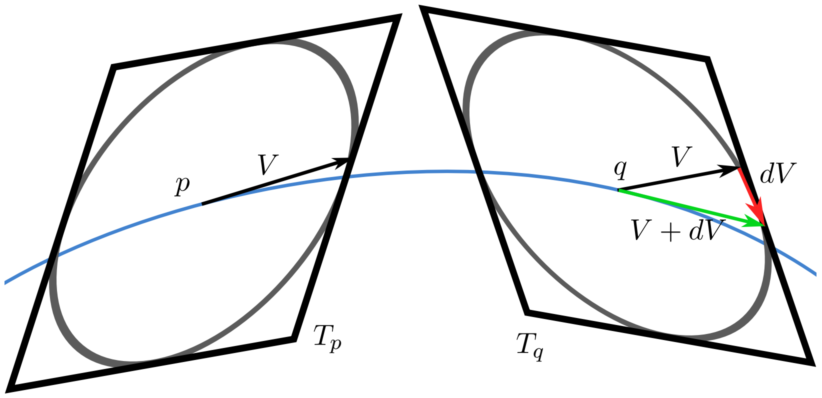


FIGURE 3.19. Parallel transport of a vector V from a point p to q. V is moved from the tangent plane T_p to T_q by expressing it in the basis vectors of T_q and then the vector dV is added to form $V + dV$, the transported vector. A critical property is that $V \in T_p$ and $V + dV \in T_q$ will share the same angle with the geodesic between p and q; if V at p is tangent to the geodesic between p and q (blue), then $V + dV$ will be tangent to it at q. So, dV is a measure of how much the geodesic itself curves from p to q.

Figure 3.19 shows how parallel transport is used to calculate the change dV in an arbitrary vector V when moving from point p to q on a surface, making a step $du = q - p$. We use the identity (*e.g.* [**LR89**]):

$$dV^k = -\Gamma^k{}_{ij} V^i du^j \tag{3.1}$$

Ignoring for a moment what each variable is, the Einstein Notation used here tells us several things: dV, V, and du are all $1 \times n$ (contravariant) vectors, and Γ is an $n \times n \times n$ array. As illustrated in Figure 3.19, this identity establishes the change dV in a vector V, when moving along a step du over a surface with *connection* Γ.

The *Christoffel Symbol* $\Gamma^k{}_{ij}$ determines the *connection*, or how vectors change as we move from one point to an infinitesimally close neighbor, due to the curvature of the surface. The connection is not uniquely defined in general, but we use the Levi-Civita connection, which is the unique *torsion-free* connection. Torsion is induced rotation of the tangent plane when moving along a curve on the surface, so the Levi-Civita connection ensures the only rotation of tangent vectors is to correct for surface curvature. The Levi-Civita connection defines[3] the $2 \times 2 \times 2$ array $\Gamma^k{}_{ij}$ as:

[3] [**LR89**], Equation 5.14

$$\Gamma^k{}_{ij} = \frac{1}{2} g^{km} \left(\frac{\partial g_{mi}}{\partial u^j} + \frac{\partial g_{mj}}{\partial u^i} - \frac{\partial g_{ij}}{\partial u^m} \right) \tag{3.2}$$

In an attempt to supplement reader (and author) unfamiliarity with Einstein Notation, we express the $2 \times 2 \times 2$ array $\Gamma^k{}_{ij}$ in the following matrix-esque notation, with the front and back 2×2 arrays (the two different values for index i) separated by a bar:

$$\Gamma^k{}_{ij} = \left[\begin{array}{cc|cc} \Gamma^1{}_{11} & \Gamma^1{}_{12} & \Gamma^1{}_{21} & \Gamma^1{}_{22} \\ \Gamma^2{}_{11} & \Gamma^2{}_{12} & \Gamma^2{}_{21} & \Gamma^2{}_{22} \end{array} \right]$$

This appears cumbersome at first, but looking at $\Gamma^k{}_{ij} V^i$ in isolation, we can see that this is a $2 \times 2 \times 2$ array. If we contract it with a 2×1 column vector V^i, we get a 2×2 array that we can display in standard matrix notation. Formally, this temporary matrix $T^k{}_j = \Gamma^k{}_{ij} V^i$ maintains the indices j and k, while i has been removed by contracting with V. We order the entries of $\Gamma^k{}_{ij}$ in this pseudo-matrix notation so that the front-and-back matrices collapse into the 2×2 temporary matrix T, which is then correctly aligned to have the j index cancel out when multiplied with du^j.

$$\left[\begin{array}{cc|cc} \Gamma^1{}_{11} & \Gamma^1{}_{12} & \Gamma^1{}_{21} & \Gamma^1{}_{22} \\ \Gamma^2{}_{11} & \Gamma^2{}_{12} & \Gamma^2{}_{21} & \Gamma^2{}_{22} \end{array} \right] \begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \Gamma^1{}_{11} V_x + \Gamma^1{}_{21} V_y & \Gamma^1{}_{12} V_x + \Gamma^1{}_{22} V_y \\ \Gamma^2{}_{11} V_x + \Gamma^2{}_{21} V_y & \Gamma^2{}_{12} V_x + \Gamma^2{}_{22} V_y \end{bmatrix} = \begin{bmatrix} T^1{}_1 & T^1{}_2 \\ T^2{}_1 & T^2{}_2 \end{bmatrix}$$

We can re-write Equation 3.2 for our specific conformal metric tensor g in this more cumbersome but familiar notation as:

$$\Gamma = \frac{1}{\sigma} \left[\begin{array}{cc|cc} \partial\sigma/\partial x & \partial\sigma/\partial y & -\partial\sigma/\partial y & \partial\sigma/\partial x \\ \partial\sigma/\partial y & -\partial\sigma/\partial x & \partial\sigma/\partial x & \partial\sigma/\partial y \end{array} \right] \tag{3.3}$$

3.2.5.1. *A Discrete Example.* We will now look at an example to ensure the behavior of our derived equations matches the intuition we established in Section 3.2.2.

Parallel transport on a Riemannian surface gives us the change dV observed in a tangent vector V when moving along the surface as the metric g varies. Instead of the infinitesimal step du given in Equation 3.1, let us consider a discrete step (S_x, S_y), as we will eventually be stepping over edges in the background mesh, where $S = v_j - v_i$. We will also consider the change induced over the step to be a rotation θ instead of vector dV,

$$\theta = \text{angle}(V + dV) - \text{angle}(V)$$

with the function angle$(V) = \text{atan2}(V_y, V_x)$ returning the signed (counter-clockwise) rotation from the $+x$ direction. This works for our triangles in the plane where the $+x$ can serve as a basis vector as it lies in the tangent plane of all triangles. On a curved surface, the angles would need to be properly adjusted to reflect the individual basis vectors for each triangle.

We want to ensure the behavior of θ matches the intuition from Figure 3.18, in particular that we see no rotation along the size gradient, and bending toward the region of higher size when moving perpendicular to the gradient. We will consider the rotation induced by parallel transport in the scenario where we have the local size factor $\sigma = 1$ and where the gradient is in the $+x$ direction; $\partial\sigma/\partial x = 1$, and $\partial\sigma/\partial y = 0$. When applied to Equation 3.3 we get:

$$\Gamma = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 \end{array}\right]$$

which can be applied to Equation 3.1:

$$dV = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 \end{array}\right] \begin{bmatrix} V_x \\ V_y \end{bmatrix} \begin{bmatrix} \text{Step}_x \\ \text{Step}_y \end{bmatrix}$$

$$dV = \begin{bmatrix} V_x & V_y \\ V_y & -V_x \end{bmatrix} \begin{bmatrix} \text{Step}_x \\ \text{Step}_y \end{bmatrix}$$

If we take a unit step in the $+x$ direction, which is along the gradient of the size field,

$$dV = \begin{bmatrix} V_x & V_y \\ V_y & -V_x \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} V_x \\ V_y \end{bmatrix} = V$$

so $\theta = \text{angle}(V + dV) - \text{angle}(V) = \text{angle}(2V) - \text{angle}(V) = 0$.

This demonstrates that moving directly along (or against) the gradient does not induce a rotation along the edge.

On the other hand, taking a unit step in the $+y$ direction, perpendicular to the gradient of the size field,

$$dV = \begin{bmatrix} V_x & V_y \\ V_y & -V_x \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} V_y \\ -V_x \end{bmatrix}$$

so $\theta = \text{angle}(V + dV) - \text{angle}(V) = \text{angle}(V + \text{rotate}_{-90^\circ} V) - \text{angle}(V) = \text{angle}(V) - 45^\circ - \text{angle}(V) = -45^\circ$.

Specifically, moving *across* the gradient induces the greatest rotation along the edge. This is a negative (clockwise) rotation, so an upward V will be rotated *toward* the direction of increasing size. This matches the intuition from Figure 3.18. Note that the direction of V had no effect on the outcome; the rotation depends only on the direction of the step taken.

This works as an approximation if each step size du, and thus the lengths of the edges in the background mesh, is small enough for dV to be reliably represented as a rotation θ. This is taken into consideration when we generate the background mesh; as the size function (and therefore its gradient as well) are known during mesh generation, this *becomes* the predicate for continued mesh refinement: if any pair of vertices of an element would permit a rotation greater than some threshold θ_{max}, it must be subdivided.

3.2.5.2. *Calculation.* Note that the value of the initial vector V being transported does not matter for this calculation. We can rearrange Equation 3.1 into $dV^k = (-\Gamma^k{}_{ij}du^j)V^i$, where it becomes apparent that the matrix applied to V is independent of V. As we just want the induced rotation θ, we set $V = (1, 0)$. This simplifies to $\theta = \text{angle}(V + dV) + \text{angle}(V) = \text{angle}(V + dV) + 0 = \text{atan2}(dV_y, dV_x + 1.0)$

$$dV = \left[\begin{array}{cc|cc} \partial\sigma/\partial x & \partial\sigma/\partial y & -\partial\sigma/\partial y & \partial\sigma/\partial x \\ \partial\sigma/\partial y & -\partial\sigma/\partial x & \partial\sigma/\partial x & \partial\sigma/\partial y \end{array}\right] \begin{bmatrix} V_x \to 1 \\ V_y \to 0 \end{bmatrix} \begin{bmatrix} S_x \\ S_y \end{bmatrix}$$

$$= \frac{1}{\sigma} \begin{bmatrix} \partial\sigma/\partial x & \partial\sigma/\partial y \\ \partial\sigma/\partial y & -\partial\sigma/\partial x \end{bmatrix} \begin{bmatrix} S_x \\ S_y \end{bmatrix}$$

So, taking $(\partial_x\sigma, \partial_y\sigma) = (\partial\sigma/\partial x, \partial\sigma/\partial y)$ to be the *average* of the gradients at vertices i and j, and similarly σ as the average of the size function at both vertices:

$$dV = \frac{1}{\sigma} \begin{bmatrix} \partial_x\sigma & \partial_y\sigma \\ \partial_y\sigma & -\partial_x\sigma \end{bmatrix} \begin{bmatrix} S_x \\ S_y \end{bmatrix}$$

Splitting this into components,

$$\begin{aligned} dV_x &= (\partial_x\sigma S_x + \partial_y\sigma S_y)/\sigma \\ dV_y &= (\partial_y\sigma S_x - \partial_x\sigma S_y)/\sigma \end{aligned} \tag{3.4}$$

we can use the components to calculate the angle of rotation observed over the step S:

$$\theta = \text{atan2}((\partial_y\sigma S_x - \partial_x\sigma S_y)/\sigma, 1.0 + (\partial_x\sigma S_x + \partial_y\sigma S_y)/\sigma) \tag{3.5}$$

3.3. *Globally Optimal Direction Fields*

Now that we know the curvature induced by a varying metric, we can integrate this into the *GODF* approach in order to produce size-driven cross fields. Instead of immediately presenting the solve performed by *GODF*, we build intuition by working through the problem formulations that lead up to the actual solve. We start with their definition of a smooth field, introduce the Helmholtz equation that allows us to solve for the smoothest field as an eigenvalue problem, and then use a Finite-Element Method formulation that discretizes the continuous eigenvalue problem into a piecewise linear function over the background triangle mesh.

For an introduction to Finite-Element Methods, we direct the reader to Francisco-Javier Sayas' *A gentle introduction to the Finite Element Method* [**Say15**], and we will try to match their notation, as well as the notation in *GODF*, whenever possible.

GODF [**KCPS13**] describes a method to produce *N-RoSy* direction fields on manifold surfaces that balance being as *smooth* as possible while optionally as aligned as possible with a guidance direction field, such as *e.g.* the curvature of the underlying surface. Without a guidance field, their smoothness constraint optimizes toward straight (geodesic) paths, which only curve in the direction normal to the surface. Formally, logically straight lines on the surface are the ones that follow the the Levi-Civita connection (described in Section 3.2.5) on the surface. Our (varying) size function over our (flat) surface is a Riemannian metric, so also defines a Levi-Civita connection over the surface; in our case the geodesics do curve in the surface plane, but they are true shortest-path curves in the metric.

Smoothness here refers to the Dirichlet energy E_D of a field ψ over surface Ω (which is our mesh M, once discretized):

$$E_D = \frac{1}{2}\int_M ||\nabla\psi||^2 \, dA$$

We will minimize the Dirichlet energy by way of a related problem, the Helmholtz equation:

$$-\Delta f = \lambda f$$

In Euclidean geometry, Δ is the *Laplacian* operator, which is the divergence $(\nabla\cdot)$ of the gradient (∇) of the function f. As we're dealing with a Riemannian surface, Δ indicates the *Laplace-Beltrami operator*, a generalization of the Laplacian operator that takes into account parallel transport on surfaces, in our case, curved by a size function. However, for the time being, we will describe the math in terms of the Euclidean

Laplacian operator, to build intuition for the problem, and refer the reader to the appendices of [**KCPS13**] for derivations involving the Laplace-Beltrami operator.

Harmonic functions, where $\Delta f(p) = 0$ for all $p \in \Omega$, represent a special class of solutions to the Helmholtz equation. They are not guaranteed to exist, but when they do, they are associated with an eigenvalue of $\lambda = 0$. We will revisit harmonic functions in Section 4.5, where we investigate the consequences of harmonic fields on their associated size functions.

The trivial solution $f = 0$ is harmonic but is not a useful result, so we enforce unit norm over f to avoid the trivial solution. While there may or may exist an eigenvalue $\lambda = 0$, we consider the smallest eigenvalue λ_1 and its associated eigenfunction f_1 which have the useful "max-min" property:

$$\lambda_1 = \inf_{f \neq 0} \frac{\int_\Omega ||\nabla f||^2 \, dA}{\int_\Omega ||f||^2 \, dA}$$

With the enforced unit norm on f this simplifies to:

$$\lambda_1 = \inf_{f \neq 0} \int_\Omega ||\nabla f||^2$$

If there is a nontrivial harmonic solution, it will be found as f_1 with $\lambda_1 = 0$. However, in most cases, there will only be a trivial harmonic solution, so f_1 will be the minimum-energy function among those with unit norm, but not a minimum-energy function in general, so will not be harmonic.

We can add the homogeneous Neumann boundary condition, and formulate the eigenvalue problem as a boundary-value problem:

$$\text{find value } \lambda \text{ and function } u \neq 0 \text{ such that } \left[\begin{array}{ll} -\Delta u = \lambda u & \text{in } \Omega \\ \frac{\partial u}{\partial \mathbf{n}} = 0 & \text{on } \Gamma \end{array} \right.$$

In order to solve this, we want to transform this continuous eigenvector problem into a discrete eigenvector problem. We start with Green's Theorem

$$\int_\Omega (\Delta u) v \, dA + \int_\Omega \nabla u \cdot \nabla v \, dA = \int_\Gamma (\frac{\partial u}{\partial \mathbf{n}}) v \, ds$$

which introduces the *virtual function* v, also called the "test function". This allows a relation to be made between area integrals over Ω on the left side of the equation and line integrals over Γ on the right. While u is unchanged, and is still the function we are trying to solve for, v can be any function so long as it takes the value zero on the boundary Γ_D. Any virtual function v provides a constraint on u.

Applying Green's Theorem to the eigenvalue formulation of the boundary value problem yields the *weak* or *variational* form of the eigenvector problem:

$$\text{find value } \lambda \text{ and function } u \neq 0 \text{ such that } \left[\begin{array}{ll} \frac{\partial u}{\partial \mathbf{n}} = 0 & \text{on } \Gamma \\ \int_\Omega \nabla u \cdot \nabla v \; dA = \lambda \int_\Omega uv \; dA & \text{in } \Omega, \text{for all } v \\ & \quad \text{such that } v = 0 \text{ on } \Gamma \end{array}\right.$$

3.3.0.1. *The Discrete Eigenvalue Problem.* The formulation above provides a simple representation of an eigenvalue finite-element problem, but it needs to be discretized to a finite-dimensional form where we can compute a solution. This section is effectively the solution to Exercise 3 from Chapter 4 in [Say15].

We first discretize the problem over our background triangle mesh, so our solution u will be piecewise-linear. We apply the decomposition:

$$u = \sum_i \mathbf{u}_i \phi_i$$

which encodes the piecewise-linear function u as a vector $\mathbf{u}$, where each element $\mathbf{u}_i$ is a scalar value associated with vertex v_i. The function ϕ_i is the hat function, a linear basis function such that $\phi_i = 1$ at v_i and $\phi_i = 0$ at all other vertices, and is linearly interpolated everywhere else. Within some arbitrary triangle t_{jkl}, there will only be three nonzero basis functions: ϕ_j, ϕ_k, and ϕ_l. Observe that the gradient of u is the gradient of the basis functions scaled by $\mathbf{u}$:

$$\nabla u = \sum_i \mathbf{u}_i \nabla \phi_i$$

Starting with our formulation from Green's theorem, noting that the right-hand side $\int_\Gamma (\frac{\partial u}{\partial \mathbf{n}}) v \; ds$ is always zero due to the Neumann boundary condition:

$$\text{Find } u \neq 0, \lambda \text{ such that } \int_\Omega \nabla u \nabla v \; dA = \lambda \int_\Omega uv \; dA \text{ for all functions } v \tag{3.6}$$

we can derive the entries of the Mass and Energy (also known as Stiffness) matrices. Since $\mathbf{u}$ has one component per vertex, we need one virtual function per vertex to set up a solvable system of equations. As this is valid for any piecewise-linear function v, we consider each of our linear basis functions ϕ_i to be one of the virtual functions v that must be satisfied.

The left-hand side of Equation 3.6,

$$\int_\Omega \nabla u \nabla v \; dA = \sum_{i,j} u_i \nabla \phi_i \nabla \phi_j = (\nabla \phi_i \cdot \nabla \phi_j) \cdot \mathbf{u} = W\mathbf{u}$$

yields a system of n equations, where each vertex j evaluates a separate $v = \phi_j$, so ∇v becomes $\nabla \phi_j$. This is the *Stiffness Matrix* W in [**Say15**] and is referred to as the *Energy Matrix* A in [**KCPS13**].

In dealing with the right-hand side of Equation 3.6, we split the integral over the surface Ω into integrals over each triangle t_{ijk}:

$$\lambda \int_\Omega uv \; dA = \lambda \sum_{j \in t_{ijk}} \int_{t_{ijk}} uv \; dA$$

For any point p in triangle t_{ijk}, we have the barycentric coordinates b_i, b_j, and b_k, where $p = V_i b_i + V_j b_j + V_k b_k$, where V_i is vertex i, not to be confused with the virtual basis hat function v_i. So, u evaluated at any interior point p is $u(p)$, and can be calculated as $u(p) = \mathbf{u}_i b_i + \mathbf{u}_j b_j + \mathbf{u}_k b_k$. Because each equation $v = \phi_i$ singles out a particular i, we end up with the surprisingly simple $v = b_i$, resulting in:

$$\int_{t_{ijk}} uv = \int_{t_{ijk}} (\mathbf{u}_i b_i + \mathbf{u}_j b_j + \mathbf{u}_k b_k) b_i$$

This evaluates to:

$$\int_{t_{ijk}} uv = 2|t_{ijk}| \left[\frac{1}{12}\mathbf{u}_i + \frac{1}{24}\mathbf{u}_j + \frac{1}{24}\mathbf{u}_k \right]$$

These become entries in the $n \times n$ mass matrix M, where each entry is built up by iterating over all triangles such that:

$$M_{ii} \leftarrow \frac{1}{6}|t_{ijk}|$$

$$M_{jk} \leftarrow \frac{1}{12}|t_{ijk}|$$

Finally yielding

$$\lambda \int_\Omega uv = \lambda M \cdot \mathbf{u}$$

This formulation of the Mass matrix matches the usage seen in [**KCPS13**] (on the top of the left column on page 7). M_{ii} matches exactly and they note that their calculation for M_{ij} approaches this expected value as the triangle curvature approaches zero.

Note that each triangle t_{ijk} will be evaluated three times - once for each choice of j. So, the given entries for M_{ii} also apply to M_{jj} and M_{kk}, and M_{jk} applies to all indices jk where $j \neq k$, including *e.g.* M_{kj}.

Finally, with the definitions for the Mass and Energy matrices, finding a function u consistent with Equation 3.6 involves solving the generalized eigenvector problem:

$$W\mathbf{u} = \lambda M\mathbf{u}$$

3.3.0.2. *Connection, Encoded as Matrix.* In the previous section, we covered a simple form of the finite-element method. The construction we employ from *GODF* to produce a cross field from an input size function differs in a few key ways. Primarily, instead of solving for a scalar function u over the region, u is now a complex-valued function, and accordingly the symmetric matrices M and W above become the Hermitian matrices M and A; while [**Say15**] used W for the energy matrix, we will now use A to match the usage in [**KCPS13**].

Additionally, *GODF* uses different computations for the mass and energy matrices than we have presented, because we did not take surface curvature into account, for clarity. As we called out, our calculation for the entries of the mass matrix match their approach for flat triangles. Appendix D in *GODF* includes the derivation of these updated functions, and high-accuracy implementations are included in the supplemental materials.

The mass and energy matrices are built up by summing over each triangle t_{ijk} in the mesh. Repeated indices refer to diagonal entries for a single vertex, such as M_{ii}. Entries M_{jk} also imply corresponding operations for M_{ij} and M_{ki}. As both matrices are Hermitian, the off-diagonal matrix entries will be complex conjugates: $M_{ij} = \overline{M_{ji}}$.

There are a number of per-triangle or per-edge intermediate values that are computed in building the matrices.

The *transport coefficient* r_{pq} is the complex value representing the induced rotation when traveling from vertex p to vertex q. Given a geodesic connecting the two mesh vertices, [**KCPS13**] defines ρ_p as the rotation that aligns the local coordinate system at p with the geodesic, and $\rho_{pq} = \rho_q - \rho_p$ as the rotation in radians between the two coordinate systems; a vector in the coordinate space of vertex v_p will be rotated by ρ_{pq} when represented in the tangent space of vertex v_q. In our case in 2D, an alignment vector used in [**KCPS13**] is not necessary to orient each tangent space, simplifying the calculation of ρ_{pq} below.

The geodesic along a step $[S_x, S_y]$ between vertices p and q does not need to be calculated explicitly; we can compute the angle of deviation by using Equation 3.5:

$$\theta = \text{atan2}((\partial_y \sigma S_x - \partial_x \sigma S_y)/\sigma, 1.0 + (\partial_x \sigma S_x + \partial_y \sigma S_y)/\sigma) \qquad \text{(From 3.2.5.2)}$$

Combined with the rotation angle (multiplied by the field symmetry; equivalent to raising the resulting complex rotation to the n^{th} power):

$$\rho_{pq} := n\theta_{pq} \qquad \text{([KCPS13] Eqn. 4)}$$

and the definition of the transport coefficient

$$r_{pq} := e^{i\rho_{pq}} \qquad \text{([KCPS13] Unnumbered)}$$

we are able to directly compute the complex value:

$$r_{pq} = e^{4i\cdot\text{atan2}((\partial_y \sigma S_x - \partial_x \sigma S_y)/\sigma, 1.0 + (\partial_x \sigma S_x + \partial_y \sigma S_y)/\sigma)}$$

The holonomy Ω_{ijk} over a triangle is defined as

$$e^{i\Omega_{ijk}} := r_{ij} r_{jk} r_{ki} \qquad \text{([KCPS13] Eqn. 13)}$$

so we evaluate $\Omega_{ijk} = \text{arg}(r_{ij} r_{jk} r_{ki})$ where the arg function yields the phase of its complex argument; it may be defined as:

$$\text{arg}(x + iy) = atan2(y, x)$$

The triangle area $|t_{ijk}|$ and edges $p_{st} = p_t - p_s$ are computed directly from the vertices p_i, p_j and p_k.

In ancillary materials, [KCPS13] provide high-accuracy implementations of the helper functions **MassIJ** and **DirichletIJ**, as these functions are likely to suffer numerical precision issues if naively implemented. These functions are the per-triangle integrals necessary to build the mass and energy matrices, but here are being computed over curved surfaces, with the assumption that each triangle has constant curvature from the holonomy Ω_{ijk} over the triangle t_{ijk}. We use these functions in our implementation.

Substituting in the helper functions into the matrix summation terms yields:

$$\begin{aligned} M_{ii} &= M_{jj} = M_{kk} = \tfrac{1}{6}|t_{ijk}| \\ M_{jk} &= \overline{r}_{jk}|t_{ijk}|\textbf{MassIJ}(\Omega_{ijk}) \end{aligned} \qquad \text{([KCPS13] Eqn. 17)}$$

$$\begin{aligned} A_{ii} &= \Delta_{ii} - s\tfrac{\Omega_{ijk}}{|t_{ijk}|}M_{ii} \\ A_{jk} &= \Delta_{jk} - s\left(\tfrac{\Omega_{ijk}}{|t_{ijk}|}M_{jk} - \epsilon_{jk}\tfrac{i\overline{r}_{jk}}{2}\right) \end{aligned} \qquad \text{([KCPS13] Eqn. 18)}$$

$$\begin{aligned} \Delta_{ii} &= \tfrac{1}{4|t_{ijk}|}\left[|p_{jk}|^2 + \Omega_{ijk}^2\tfrac{|p_{ij}|^2+\langle p_{ij},p_{ik}\rangle+|p_{ki}|^2}{90}\right] \\ \Delta_{ij} &= \overline{r}_{ij}\tfrac{1}{|t_{ijk}|}\textbf{DirichletIJ}(\Omega_{ijk}, |p_{ki}|^2, -\langle p_{ki}, p_{jk}\rangle, |p_{jk}|^2) \end{aligned} \qquad \text{([KCPS13] Unnumbered)}$$

In addition to the Δ_{ij} energy given above, the complex conjugate is also used (note the leading r_{ij}, instead of the $\overline{r}_{ij}$ in the previous equation):

$$\Delta_{ji} = r_{ij}\frac{1}{|t_{ijk}|}\textbf{DirichletIJ}(\Omega_{ijk}, |p_{ki}|^2, -\langle p_{ki}, p_{jk}\rangle, |p_{jk}|^2)$$

Currently, we're working with $s = 0$, representing what Knoppel et al. call the "Dirichlet energy" formulation, where the boundary conditions are assumed to have a derivative of zero. The Dirichlet energy has increased singularity count and straighter field lines, while the alternate energy formulation in [KCPS13] splits the Dirichlet energy into "holomorphic" and "anti-holomorphic" components, offering fewer singularities and a balance of the two, respectively. Because we're using the Dirichlet energy, $A = \Delta$.

Now that the mass and energy matrices have been built, it is possible to solve the generalized eigenvector problem $A\mathbf{u} = \lambda M\mathbf{u}$, where our goal is the vector $\mathbf{u}$ associated with the smallest nonzero eigenvalue. Each entry u_i in the vector holds the fourth power of a complex number, representing a rotation in the tangent plane at v_i.

We leverage the SciPy software suite to perform the solve, and take advantage of their sparse matrix formats to efficiently construct M and A. As the shape of the matrix is $|V| \times |V|$, but we only store values along the diagonal and along entries corresponding to mesh edges, both of which are only $O(|V|)$, using the sparse matrix linear algebra functions is a necessary performance optimization. We start with the SciPy *Dictionary-Of-Keys* matrix, **scipy.sparse.dok_matrix**, which simply stores index-value pairs, avoiding the restructuring required for other formats when new entries are added to the matrix. Then, we convert these matrices to the *Compressed Sparse Column* format, **scipy.sparse.csc_matrix**, which is a format that is much more efficient for the following solve. We use the **scipy.sparse.linalg.eigs** solver in shift-invert mode by specifying the optional parameter *sigma*$= 0$. The LAPACK solvers that SciPy is built on top of are better

Algorithm 1 Globally-Optimal Direction Field Generation

```
1: procedure CALCULATE TRANSPORT COEFFICIENTS
2:     for edge $e_{ij} \in$ edges do
3:         $\sigma = (\sigma_i + \sigma_j)/2$
4:         $g = (g_i + g_j)/2$
5:         $S = v_j - v_i$
6:         $dV_x = (\partial_x\sigma \cdot S_x + \partial_y\sigma \cdot S_y)/\sigma$
7:         $dV_y = (\partial_y\sigma \cdot S_x - \partial_x\sigma \cdot S_y)/\sigma$
8:         $\theta_{ij} = \text{atan2}(dV_y, 1 + dV_x)$
9:         $\rho_{ij} = -n * \theta_{ij}$
10:        $r_{ij} = \cos(\rho_{ij}) + i \cdot \sin(\rho_{ij})$                ▷ $r_{ij} = e^{i\rho_{ij}}$
11:        $r_{ji} = \cos(\rho_{ji}) - i \cdot \sin(\rho_{ji})$                ▷ $r_{ij} = \overline{r}_{ji}$
12: procedure CALCULATE MASS AND ENERGY MATRICES
13:     $M = [0]$                ▷ Entries in $M$ and $A$ are accumulated over all triangles
14:     $A = [0]$
15:     for triangle $t_{ijk} \in$ triangles do
16:         $M_{ii} \leftarrow \frac{|t_{ijk}|}{6}; M_{jj} \leftarrow \frac{|t_{ijk}|}{6}; M_{kk} \leftarrow \frac{|t_{ijk}|}{6}$
17:         $\Omega_{ijk} = \arg(r_{ij} \cdot r_{jk} \cdot r_{ki})$
18:         $M_{ij} \leftarrow \overline{r}_{ij} \cdot |t_{ijk}| \cdot \textbf{MassIJ}(\Omega_{ijk})$
19:         $M_{jk} \leftarrow \overline{r}_{jk} \cdot |t_{ijk}| \cdot \textbf{MassIJ}(\Omega_{ijk})$
20:         $M_{ki} \leftarrow \overline{r}_{ki} \cdot |t_{ijk}| \cdot \textbf{MassIJ}(\Omega_{ijk})$                ▷ $M$ is Hermitian, so $M_{ij} = \overline{M}_{ji}$
21:         $M_{ji} \leftarrow r_{ji} \cdot |t_{ijk}| \cdot \textbf{MassIJ}(\Omega_{ijk})$
22:         $M_{kj} \leftarrow r_{kj} \cdot |t_{ijk}| \cdot \textbf{MassIJ}(\Omega_{ijk})$
23:         $M_{ik} \leftarrow r_{ik} \cdot |t_{ijk}| \cdot \textbf{MassIJ}(\Omega_{ijk})$
24:         $A_{ii} \leftarrow \frac{1}{4 \cdot |t_{ijk}|} \cdot (|p_{jk}|^2 + \Omega_{ijk}^2 \cdot \frac{(|p_{ij}|^2 - \langle p_{ij}, p_{ki} \rangle) + |p_{ki}|^2)}{90})$                ▷ edge $p_{ij} = p_j - p_i$
25:         $A_{jj} \leftarrow \frac{1}{4 \cdot |t_{ijk}|} \cdot (|p_{ki}|^2 + \Omega_{ijk}^2 \cdot \frac{(|p_{jk}|^2 - \langle p_{jk}, p_{ij} \rangle) + |p_{ij}|^2)}{90})$
26:         $A_{kk} \leftarrow \frac{1}{4 \cdot |t_{ijk}|} \cdot (|p_{ij}|^2 + \Omega_{ijk}^2 \cdot \frac{(|p_{ki}|^2 - \langle p_{ki}, p_{jk} \rangle) + |p_{jk}|^2)}{90})$
27:         $A_{ij} \leftarrow \overline{r}_{ij} \frac{1}{|t_{ijk}|} \cdot \textbf{DirichletIJ}(\Omega_{ijk}, |p_{ki}|^2, \langle p_{ki}, p_{jk} \rangle, |p_{jk}|^2)$
28:         $A_{jk} \leftarrow \overline{r}_{jk} \frac{1}{|t_{ijk}|} \cdot \textbf{DirichletIJ}(\Omega_{ijk}, |p_{ij}|^2, \langle p_{ij}, p_{ki} \rangle, |p_{ki}|^2)$
29:         $A_{ki} \leftarrow \overline{r}_{ki} \frac{1}{|t_{ijk}|} \cdot \textbf{DirichletIJ}(\Omega_{ijk}, |p_{jk}|^2, \langle p_{jk}, p_{ij} \rangle, |p_{ij}|^2)$                ▷ $A$ is Hermitian, so $A_{ij} = \overline{A}_{ji}$
30:         $A_{ji} \leftarrow r_{ij} \frac{1}{|t_{ijk}|} \cdot \textbf{DirichletIJ}(\Omega_{ijk}, |p_{ki}|^2, \langle p_{ki}, p_{jk} \rangle, |p_{jk}|^2)$
31:         $A_{kj} \leftarrow r_{jk} \frac{1}{|t_{ijk}|} \cdot \textbf{DirichletIJ}(\Omega_{ijk}, |p_{ij}|^2, \langle p_{ij}, p_{ki} \rangle, |p_{ki}|^2)$
32:         $A_{ik} \leftarrow r_{ki} \frac{1}{|t_{ijk}|} \cdot \textbf{DirichletIJ}(\Omega_{ijk}, |p_{jk}|^2, \langle p_{jk}, p_{ij} \rangle, |p_{ij}|^2)$
33:     $A \leftarrow A + 10^{-8} \cdot M$                ▷ Avoid factorization issues, as noted in Section 7 of [KCPS13].
```

Algorithm 1 Globally-Optimal Direction Field Generation, Continued

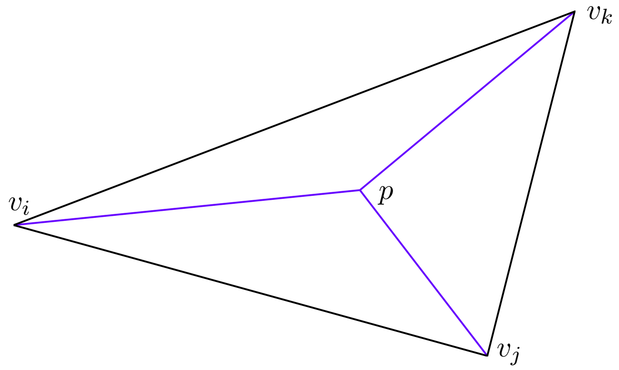
```
1: procedure CALCULATE DIRECTION FIELD
2:     Solve $Au = \lambda Mu$                ▷ Solved via scipy.sparse.linalg.eigs()
3:     for point $p \in$ triangle $t_{ijk}$ do
4:         $\alpha, \beta, \gamma = \text{barycentric\_weights}(p, t_{ijk})$
5:         $\varphi_p = \arg(\sqrt[n]{\alpha u_i + \beta u_j + \gamma u_k}) + c$                ▷ $c \in [0, 2\pi)$ constant over the entire mesh.
```

suited to finding large eigenvalues, so shift-invert mode allows SciPy to transform the smallest-eigenvalue problem into a largest-eigenvalue problem by inverting the input matrix. [VGO+20]

3.3.1. Direction Field Conversion. The vector u now stores the directions that we want to interpolate over the mesh to form the full direction field. Calculating the direction of the field at any point p requires finding the triangle t_{ijk} containing p and then interpolating the values stored at the vertices to find φ_p. Figure 3.20 illustrates the barycentric interpolation used to calculate φ_p. The barycentric coordinates α, β, and γ are the fraction of the area of t_{ijk} covered by the triangle opposite the vertex the weight is assigned to, *e.g.* the weight α is assigned to v_i and is the relative area of the triangle formed by p, v_j, and v_k. The interpolated value $\alpha \cdot u_i + \beta \cdot u_j + \gamma \cdot u_k$, like the values in u, are all complex numbers to the n^{th} power. It is important to take the n^{th} root of the angle *after* interpolation, in order to avoid the issues with angle interpolation described in Section 3.1.2.2.

To find φ_p, in our *4-RoSy* field, we calculate:

$$\varphi_p = \text{arg}(\sqrt[4]{\alpha \cdot u_i + \beta \cdot u_j + \gamma \cdot u_k})$$



(a) Splitting t_{ijk} at p, forming three triangles, the areas of which are the weights of the barycentric coordinates.

$$\begin{aligned}\alpha &= \text{area}(\ p, v_j, v_k) \ / \ \text{area}(v_i, v_j, v_k)\\ \beta &= \text{area}(v_i,\ p, v_k) \ / \ \text{area}(v_i, v_j, v_k)\\ \gamma &= \text{area}(v_i, v_j,\ p) \ / \ \text{area}(v_i, v_j, v_k)\end{aligned}$$

$$q_p = \alpha \cdot q_i + \beta \cdot q_j + \gamma \cdot q_k$$

(b) Calculating barycentric coordinates, and applying them to find some quantity q_p, interpolated from values stored at the vertices q_i, q_j, and q_k.

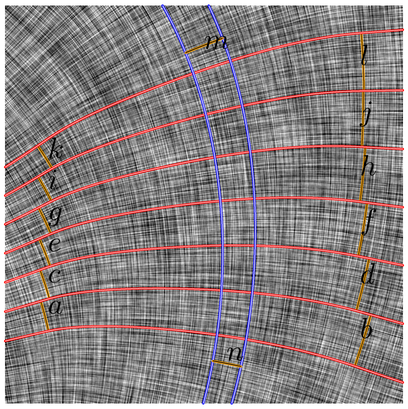
FIGURE 3.20. Calculating barycentric coordinates for a point p in triangle t_{ijk}

Interpolating the values for u over the mesh M results in a piecewise-linear function, linear within each triangle t_{ijk} and continuous over the entire region, with discontinuities in ∇u along each edge. However, the final field φ is no longer linear due to the fourth root and implicit normalization when converting from the complex vector to an angle.

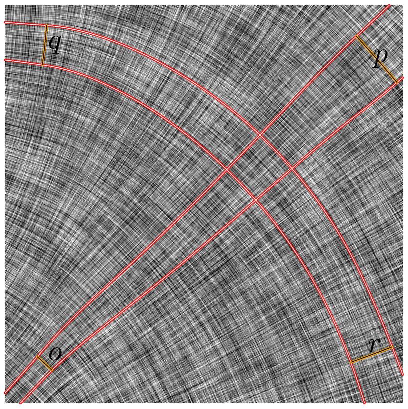
The values u are derived from deltas over each edge in the background mesh. It is as if we are given given $u_j - u_i$ for each edge e_{ij} and have solved for some valid u, or in a continuous sense, given $\nabla\varphi$ and are integrating over the region to solve for the field φ, This means there is an untapped degree of freedom here, where adding some offset ϕ to some vertex k, giving $\hat{u}_k = u_k + \phi$ ends up with the same solution except that ϕ has been added to *all* vertices $\hat{u}_i = u_i + \phi\ \forall i$; the resulting $\hat{u}$ is valid for any choice of ϕ. Another way to consider this is that the resulting field φ, which is represented by an angle function α interpolated over triangles from u, is equally valid for any constant offset ϕ: $\varphi = \alpha + \phi$. We will refer to ϕ as the *phase* of the field φ, and we will see that phase plays an important role in the meshes we generate in Section 3.4.3.

3.4. Evaluation

In this section we evaluate some cross fields produced by the method we've presented. Chapter 4 will explore size computation in more rigor, but for now we will use the relative distance between chosen field lines throughout the region as a proxy for growth. While we look at two different phases of the first example dataset to measure field growth in different directions, we observe that the chosen phase of the cross field has an impact on the resulting size functions; we will explore this more in Section 4.13.



(a) Field aligned with desired size gradient



(b) Field aligned with observed size gradient

Location + Length				Ratio
a	7.75	b	13.55	1.75
c	7.61	d	13.55	1.78
e	7.44	f	13.65	1.83
g	7.26	h	13.83	1.91
i	7.02	j	14.08	2.01
k	6.78	l	14.59	2.01
m	10.01	n	7.20	0.72
o	5.02	p	15.47	3.08
q	10.01	r	11.09	1.11

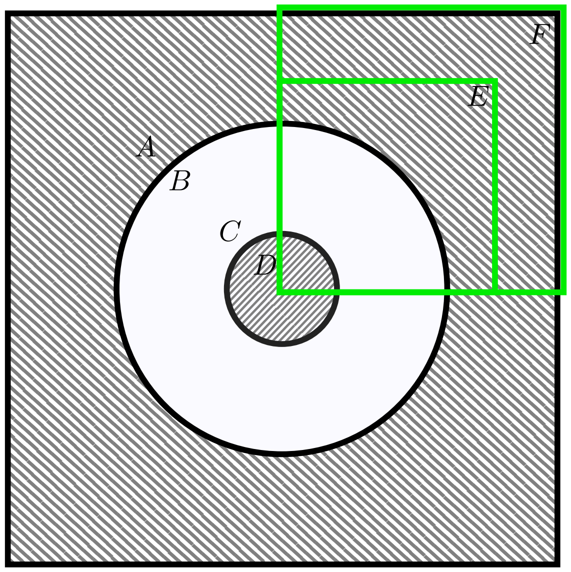
(c) Distance measurements between parallel field lines, with growth ratios.

FIGURE 3.21. Two alignments of a generated direction field aiming for a doubling of the size moving from left to right.

3.4.1. Singularity-Free Grading. Figure 3.21 shows a case where the desired grading is sufficiently simple and gradual enough to be accomplished without singularities. The input sizing function doubles from the left of the image to the right, and is constant along any vertical path.

Figure 3.21a shows the field roughly aligned with the desired size gradient. The red field lines are moving along the desired gradient, so the distances between them should increase along with the size function, which is seen by all of them roughly doubling. The blue field lines are moving perpendicular to the desired gradient direction, so they should remain a fixed distance apart; but there is an observed decrease in distance of 28%. Figure 3.21b shows the same cross field, but with an phase ϕ that is rotated $45°$ with respect to the direction of the desired gradient. The two pairs of field lines behave very differently, appearing to be radii and perimeters of circles centered just past the bottom-left corner of the image. The field lines from the bottom left, the apparent radii, show a tripling of distance over the image, while the other pair of lines remain relatively equidistant. This second set of field lines matches the intuition we established in Figure 3.18, where the field lines following the gradient are straight and those across the gradient are curved as if they are being pulled in the direction of growth. It also serves to extend our intuition: the straight field lines that follow the gradient must also get further apart as the implied size function increases, suggesting that the produced gradient is actually radial, radiating outward from some point off to the bottom-right of the image. The target configuration, therefore, is unachievable; there cannot be an exact solution to a constant non-zero size gradient, as the straight field lines that follow the gradient need to diverge proportionally to the gradient. As seen here, the optimization process still produces a smooth field that approximates the desired behavior.

This example demonstrates the limitation of relying entirely on field-line divergence to achieve growth, by not including any singularities. The best approximation of a constant gradient was the observed radial gradient. The resulting growth is not aligned with the desired gradient direction but instead with the direction that minimizes the error in growth over the entire image. So, the excessive growth seen in the radii is what, when averaged over the image, causes the properly-aligned paths to hit their marks. This misalignment is also what causes the blue path to deviate from constant-width.



A	Constant Minimum Size
B	Minimum Size
C	Maximum Size
D	Constant Maximum Size
E	Crop area for figures in this section
F	Crop area for background meshes in this section

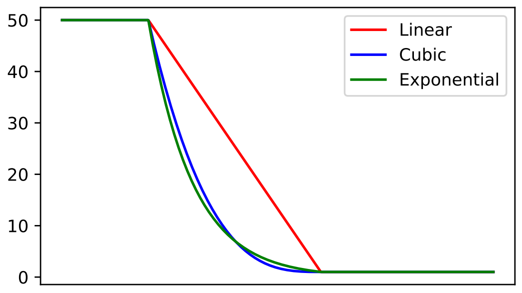
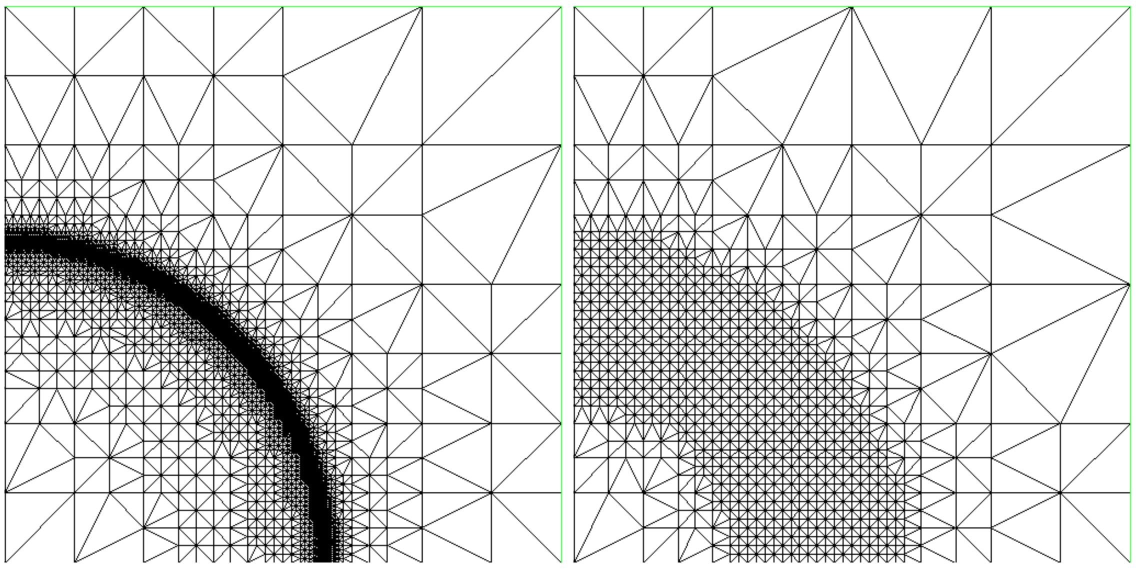


FIGURE 3.22. Synthetic example size function layout. Fields are calculated for the entire region, but exhibit enough symmetry where one quadrant effectively conveys the behavior, so the images are cropped to show more detail. Three different easing functions are used to transition sizes along the unshaded region.

3.4.2. High-Growth Synthetic Example. In this example, we consider a region defined by two concentric circles, where the size function increases from its minimum at the edge of the outer circle, to its maximum at the edge of the inner circle. These images are symmetric, so images will be zoomed into the upper-right quadrant. We purposely include a thick border region around the entire extent with a constant size, so that the resulting field is forced to include all relevant singularities in the region of interest; this is to avoid confounding issues such as seen in the example from Section 3.4.1: is the region really singularity-free, or is the observed radial pattern the result of a degree-0 singularity just off-image?

As noted when discussing element growth rates in Section 3.2.1, we expect quad sizes to change linearly over the region. However, in Section 3.2.5.2, we observed that the strength of the deflection of the field is proportional to the change in the size function and *inversely* proportional to the size itself; this means that linear growth in the size function will induce the strongest curvature in areas where the size is the smallest.

This can cause problems when combined with a fixed-density background mesh, as regions with a small-but-growing size function can have a large enough curvature to cause it to overflow beyond $\frac{\pi}{2}$ and wraparound to $-\frac{\pi}{2}$, or vice-versa. An adaptive background mesh proportional to target size would not be efficient, as zero-growth areas with small size do not benefit from high mesh resolution. A better solution is to use an adaptive background mesh that estimates the desired curvature over each cell in the background mesh, and then refines each cell where the curvature exceeds a given threshold. This gives the benefit that regions with any constant size are also sparsely meshed.



(a) Adaptive background mesh for linear growth (b) Adaptive background mesh for exponential growth

FIGURE 3.23. An exponentially growing size function distributes the curvature evenly over the region, so the area where the size grows has roughly constant density in the adaptive background mesh.

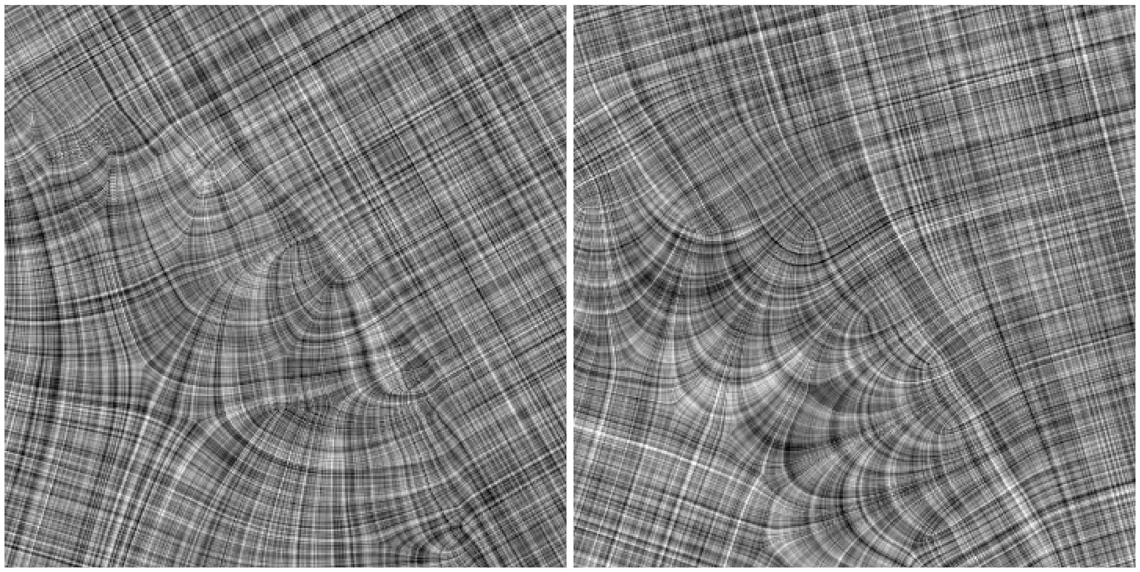

(a) LIC render for linear growth (b) LIC render for exponential growth

FIGURE 3.24. The LIC renders show the difference between linear and exponential growth in the target size function, where exponential growth has more uniform distortion over the entire growth regions, while linear growth biases the distortion toward the areas of smaller size.

Even when using an adaptive background mesh, aggressive growth rates that grow linearly will result in high-curvature regions which are very densely meshed. We will explore the relationship between curvature and size in Chapter 4, particularly in the derivation of Equation 4.3, but for now we will simply assert that the relationship between the field angle α, in particular the curvature $|\nabla\alpha|$, and the compatible size function σ is $|\nabla\alpha| \propto |\sigma^{-1}\nabla\sigma|$. A constant $\nabla\alpha$ gives uniform element sizes in the adaptive background mesh, so we solve for a σ that is proportional to its own gradient, which gives an exponential function for σ. In this circular example, using a σ with exponential growth has the effect of allowing singularities to migrate inward from the outermost edge, and they end up more evenly distributed over the region. This effect is quite pronounced, so we also consider using a cubic easing function as well, offering a balance in terms of mesh quality and singularity placement between linear and exponential growth.

While the term "easing" is used for the non-linear growth rates, taken from the animation technique of applying a one-dimensional function to an interpolant, the goal is not to remove discontinuities. In fact, none of the functions used here are C^1 smooth due to the abrupt stop in growth when transitioning to the constant (maximum) size at the inner radius, (C) in Figure 3.22. We tested easing functions with smoothed transitions to the minimum and maximum value, and did not notice a significant difference in outcomes.

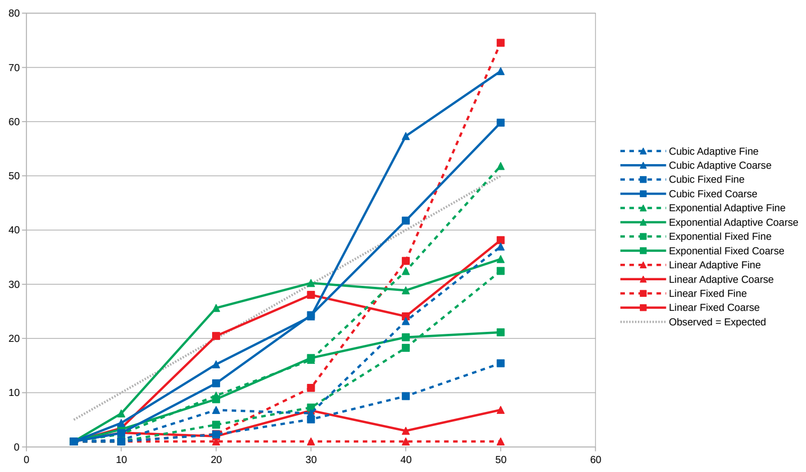
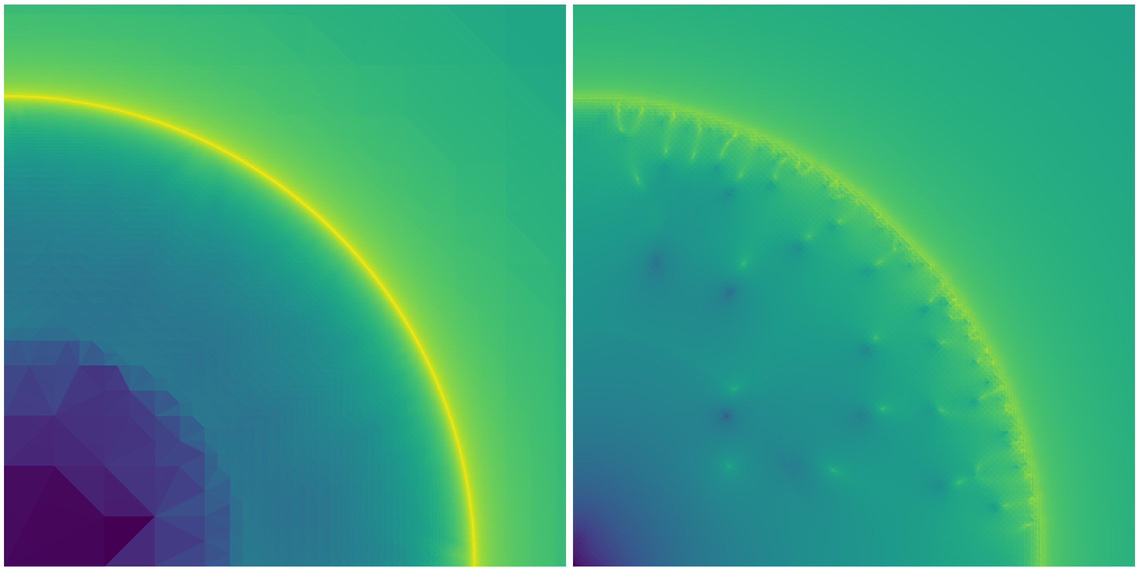


FIGURE 3.25. Observed growth rates for different background meshes and easing functions applied to the target size growth rate.

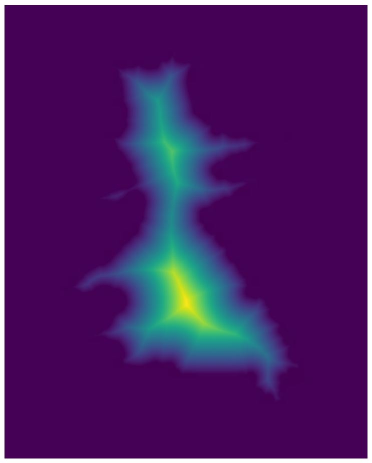


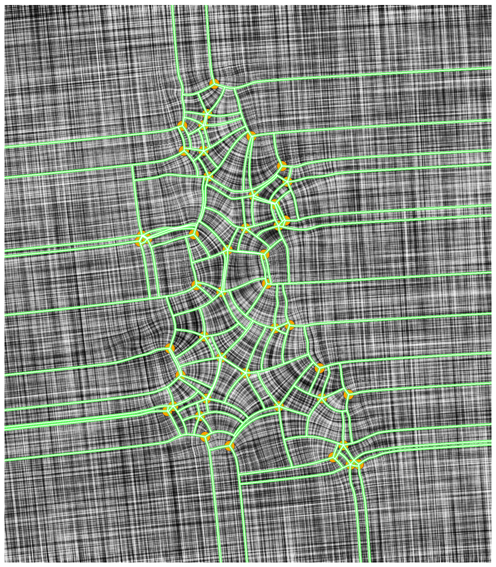
(a) Log-scale energy for linear growth with fine adaptive base mesh

(b) Log-scale energy for linear growth with fine fixed base mesh

FIGURE 3.26. Minimum-energy configurations for highest- and lowest-growth entries from Figure 3.25. Note that the adaptive background mesh case (left) corresponds to zero overall growth, and the energy is concentrated in the band where the highest growth should have occurred, while the fixed background mesh case (right) still has the (considerably weaker) high-energy outer band as well as spikes in energy corresponding to singularities in the resulting cross field.

As seen in Figure 3.25, changes to the background mesh and size function have a drastic impact on the final observed growth rate in the output cross field. In each of these cases, we measure the growth rate by taking the geometric average of many sampled growth rates, following a pair of streamlines from the perimeter of the region to the center. Chapter 4 considers better ways of measuring the growth rate associated with a cross field. The subtle differences in how a background mesh or easing function ends up distributing the curvature over the triangles in the background mesh can have effects that are difficult to predict. As seen in Figure 3.26, linear growth represents the most difficult problem to solve, as most of the relative growth is relegated to the edge of the growth region. In some cases, this causes the minimum-energy cross field to exhibit no growth at all; it is easier for the optimizer to "ignore" the growth from a relatively small group of high-curvature triangles than it is for it to distort the entire region to achieve the desired growth rate. However, because the curvature tends to be highly concentrated (in areas where the size σ is the smallest), the fine fixed-size linear base mesh can produce very strong growth, sometimes overshooting the desired growth target. The exponential growth functions tend to perform the most consistently, although not enough to use it exclusively; we often test with several easing functions for any given dataset, keeping one or a selection of the best performing parameters. The cubic easing function differs surprisingly from the exponential, given how similar they are in Figure 3.22; cubic is more consistent than the linear, and occasionally more able to match the targeted growth than the exponential. Much of this behavior was unexpected, and while the exponential case tends to make the sparsest background meshes for a given growth rate, speeding up later calculations, the lack of a clear optimal choice among the three easing functions indicates the problem is more complicated than we had initially thought; further investigation led us to the analysis and visualization we present in Chapter 4. Similarly, uniform background meshes sometimes outperform their adaptive counterparts, preventing us from using either option exclusively.


(a) Target size function


(b) Direction field and motorcycle graph

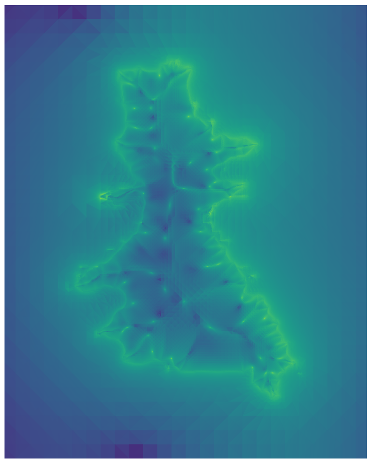

(c) Log-scale Dirichlet energy

FIGURE 3.27. The Sacramento Valley dataset. The original perimeter is clearly visible in the Dirichlet energy, where the linear interior size function meets the constant exterior size (dark purple in (a)).

3.4.3. Use Case: Geographic Dataset. We will use the Sacramento Valley dataset to explore the full meshing pipeline. Figure 3.27 shows the initial size function, the LIC and motorcycle graph, and a log-scale measure of the (minimized) optimization energy over the region. The LIC and motorcycle graph correspond to one possible phase of the cross field, while the energy function is the same for all phases.

The target size on the exterior of the region is constant to avoid adding extra singularities along the boundary, which might be necessary to accommodate the growth in a region that will be carved away from the mesh (as in Chapter 2) anyway. The straight streamlines towards the outside of Figure 3.27b show that the size is effectively constant along the outside of the region, matching the target size function.

Both the field and the motorcycle graph show the divergence of streamlines as they flow into the interior of the region, indicating that there is growth matching what we have requested. The energy measured in Figure 3.27c has bright spots coincident with all of the singularities, mostly placed along the perimeter and medial axis.

We will follow this dataset in the following chapters, measuring the growth rate compatible with our computed cross field in Chapter 4, and computing final meshes in Chapter 5.

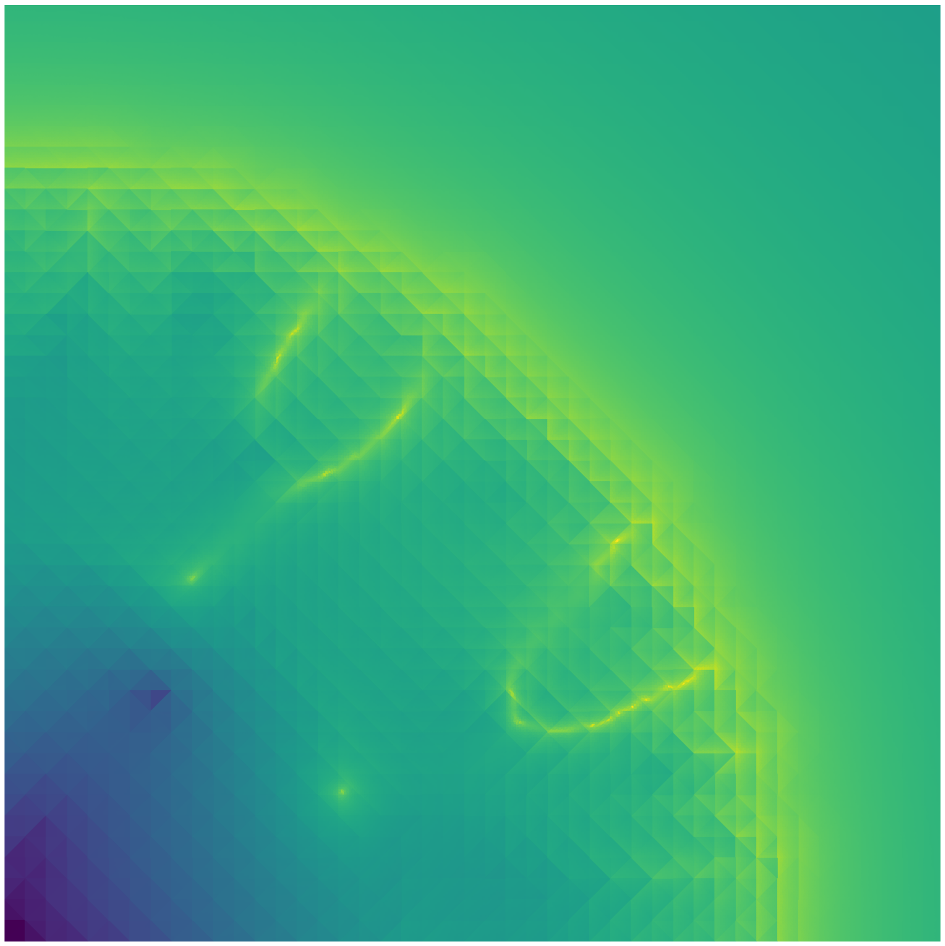


FIGURE 3.28. Log-scale heatmap of energy that is minimized by the solve process. Typically, the highest-energy regions correspond to singularities in the direction field, but in rare cases, singular bands form, which typically result in chains of singularities along a path of abnormally-high curvature. This was a linear $10\times$ growth case on a coarse fixed background mesh, which is normally a relatively easy target to achieve. The high-energy streaks are areas where there would typically be individual singularities, but here there are pairs and triplets, as if this configuration is slightly beyond what could be achieved with a normal arrangement of singularities, but is not far enough beyond to separate the singularities enough to induce growth.

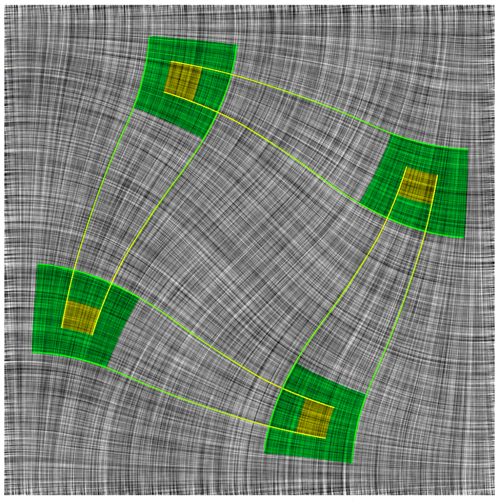


FIGURE 3.29. LIC render of a somewhat paradoxical field. Using the highlighted field line segments as a proxy for the size function, it appears as if the size function implied by this cross field is constantly shrinking while moving in a clockwise direction.

3.4.4. A Prelude to Anisotropy. Figure 3.29 shows an example that was instrumental in developing Chapter 4. Flowing between each of the colored blocks in a clockwise sequence implies that the implied size function is constantly decreasing, or has a negative directional derivative along the entire cycle. This is impossible for a size function, as we have been representing it, to achieve; if the size function is a scalar-valued field, the size measured along any closed path must return to the original value. We will revisit a very similar field in Figure 4.7 (right), where it is apparent that this field does achieve some growth without the use of singularities, at the expense of high *anisotropy*, the topic of the next chapter.

CHAPTER 4

Size Function Families from Cross Fields

Using the process described in Chapter 3, we create *size-compatible* cross fields corresponding to target size functions, although we were only able to approximate the growth in the resulting fields by measuring the divergence in pairs of parallel field lines. This chapter explores the infinite family of size functions implied by a given cross field, and the necessity of representing the size functions as two orthogonal vector fields instead of a single scalar field. The difference in magnitude of the two vectors at any point is the *anisotropy*, and measures the aspect ratios of the quads that will be produced when meshing the field. This chapter was published as *Anisotropy and Cross Fields* [**SA**], although here we consider additional datasets and conclude with a discussion of implementation caveats.

4.1. Introduction

A *cross field* assigns two orthogonal unoriented directions to every point in a two-dimensional region. A common approach to constructing a quad-mesh is, roughly, to construct a cross field on the input domain, and then sample streamlines from the cross field to produce quads; but as we saw in Chapter 3, it rarely works out to be so easy!

Here is one fundamental issue with this approach. Within any regular infinitesimally small region we can always sample streamlines to form a mesh in which the quads approach perfect squares as the density increases. But globally, the curvature of the cross field might force some quads to be arbitrarily anisotropic. This is well-known, in the sense that existing methods deal with it; for instance, Quantized Global Parameterization [**CBK15**] empirically chooses streamlines so as to minimize anisotropy (as in Figure 4.1). In this paper, we lay out the math describing how the curvature of a cross field may force any mesh directly derived from it to be be anisotropic, and we give a numerical algorithm to measure the minimal anisotropy inherent in a given cross field.

We only handle a simple case. We assume we are given a smooth cross field, defined on an open or closed domain R in the plane (not necessarily a disk). We allow the cross field to have singular points, but only of valence 3 or 5 (see Figure 4.2). We require every streamline to have finite length, and we do not

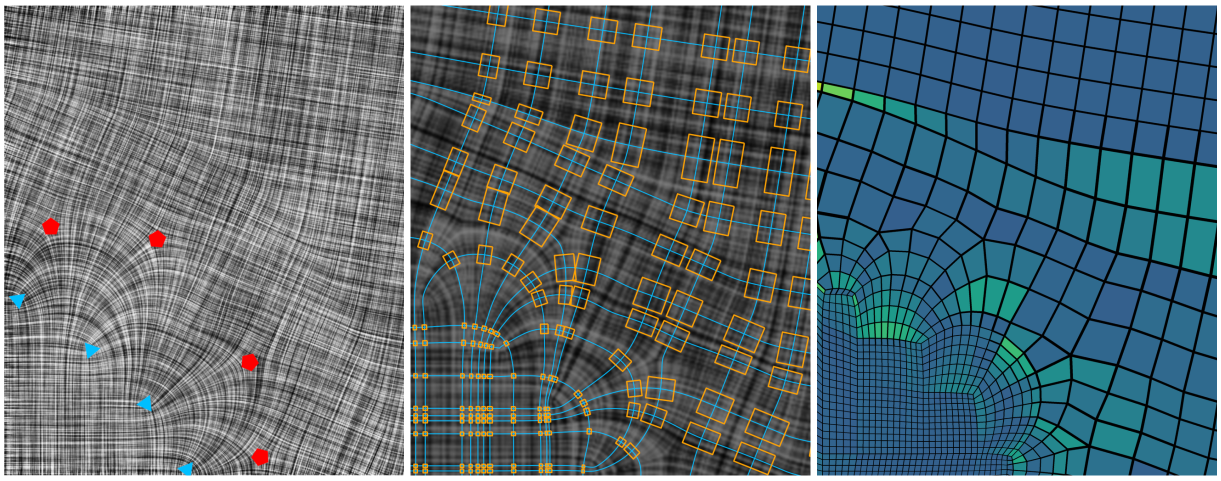


FIGURE 4.1. A cross field (left) in the plane, with singular points of valence 3 or 5 (highlighted). We assign two lengths at every point to define an anisotropic orthogonal frame field (visualized as an orange rectangle) at each point, and optimize to minimize the anisotropy (center). A quad mesh constructed with the cross field as a guide using an approach based on [CBK15] reflects the effects of the anisotropy inherent in the cross field (right). Brighter shades indicate larger element anisotropy.

allow any streamline to both begin and end at a singular point. Thus, although we allow cycles, we exclude cross fields that contain limit cycles (Figure 4.2), and also cycles containing singular points.

Given our limitations on R, our conditions on the the input cross field are reasonable. Other kinds of singular points can be (and often are) simulated by clusters of singular points of valence 3 and valence 5. A streamline with two singular endpoints can be eliminated by a local perturbation of the cross field. Limit cycles cause infinite anisotropy and streamlines of infinite length, so we cannot say much about cross fields that contain them.

We show that a cross field meeting our conditions is always 2-*integrable*, by which we mean we can assign two lengths to the two orthogonal directions at each point, consistent with the curvature of the cross field. We explain intuitively what we mean by "consistent" via the following purely theoretical construction. For some large enough n, we select a discrete set of n streamlines that decomposes R into intrinsic rectangles, that is, four-sided regions whose curved sides are segments of streamlines or closed boundaries, so that the region sides meet at right angles, except at singular points. We increase n towards infinity, interpolating, if possible, our initial discrete set of streamlines with a smooth distribution of streamlines. In the limit, the streamlines divide R into infinitesimally small perfect rectangles with straight sides ("tiny bricks"). In this sense, R is the the infinite sum of a set of infinitesimal rectangles. Definition 4.3.1, below, is a more technical definition of integrability.

The infinitesimal limit rectangles may be arbitrarily anisotropic (that is, their two side lengths may differ). Some of the anisotropy may come from our choice of the smooth streamline distribution, and some may be forced by the curvature of the cross field. What distribution of streamlines minimizes the anisotropy, so that our infinitesimal rectangles are as square as possible? That is, how much anisotropy is forced by the cross field? This is the question we address.

The infinitesimal rectangles might all be perfect squares, in which case we only need one length function (known as the *conformal factor*); we say such a cross field is 1-*integrable*. It is widely understood that a cross field is 1-integrable if and only if it can be produced by a conformal map from the plane to itself (see Section 4.5). But not all cross fields can be produced in this way; Figures 4.1 and 4.7 show examples. In this more general case, we show that there is a always a one-dimensional space (up to scale) of ways to assign two length functions consistent with the curvature of the cross field.

Next, we describe how we optimize the length functions within the set of feasible assignments so as to minimize anisotropy. Given an input cross field obeying our conditions, we can thus compute and visualize the minimal anisotropy of its best infinitely fine “tiny bricks” mesh. We present some results on measuring the anisotropy of an input cross field.

Finally, we observe that if a limit cycle is simple - that is, if it does not cross itself - it can be removed without adding any new singularities.

We emphasize that producing a finite quad mesh at a reasonable resolution poses additional challenges beyond controlling anisotropy. For instance, the finite mesh must incorporate the separatrices (the streamlines incident to the singular points) and any corners of the closed boundaries, and mesh quality includes other measures such as angle error. Thus the output mesh may depart from the optimal streamline distribution or from the streamlines themselves, as in, for example [**TPP+11**] or [**LCK21**], or in Figure 4.1.

Nonetheless our results are useful in various ways.

- If we construct a quad mesh which fails to meet a target sizing function or minimum anisotropy bound, we can estimate how much of this failure was inherent in the cross field and how much was due to downstream steps of the meshing process. This is useful since the cross fields we compute in practice are almost never exactly 1-integrable.
- Instead of designing cross fields only for smoothness under constraints, we could also optimize for isotropy or targeted anisotropy.
- The idea for removing limit cycles in Section 4.12 appears to be novel.

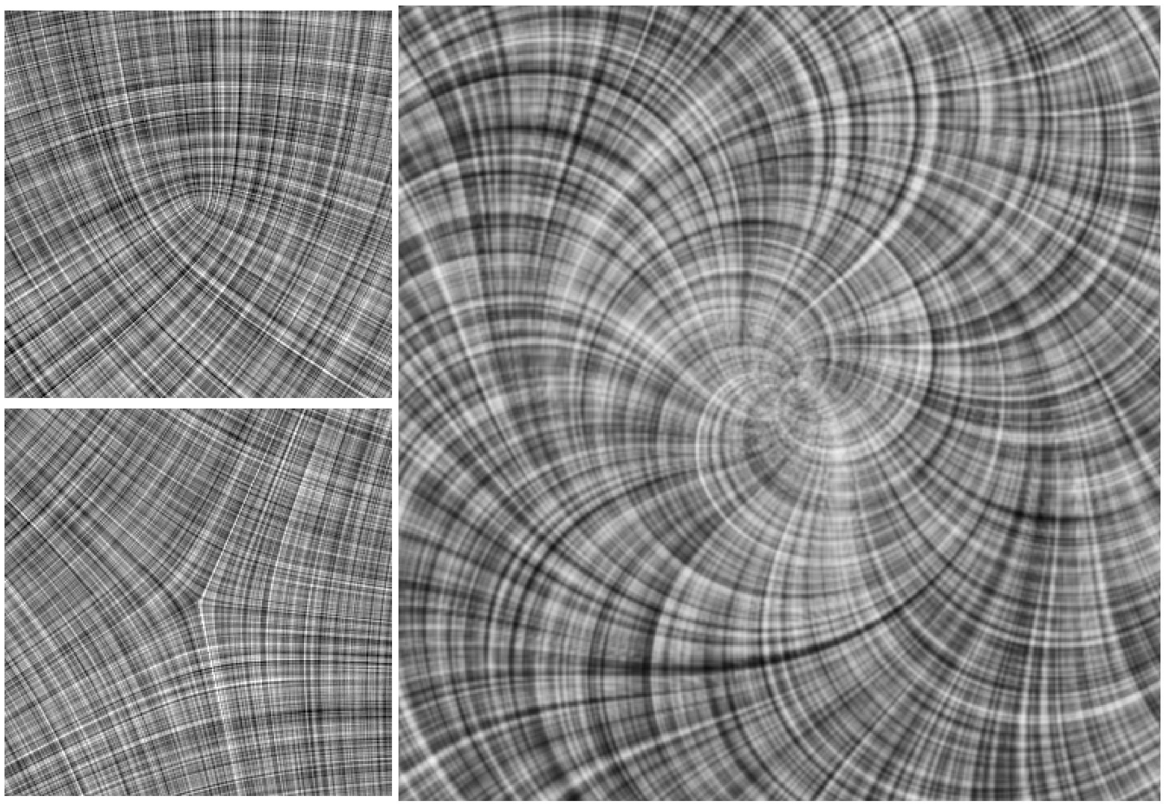


FIGURE 4.2. We allow our cross fields to contain singular points of valence 3 (top left) and 5 (bottom left), but not limit cycles (right).

4.2. Related work

There is a great deal of research on using cross fields to construct quad meshes. The design of cross fields was surveyed in [**VCD$^+$16**], and their use in quad meshing appears in the surveys [**BLP$^+$13**, **Cam17**].

Since harmonic angle functions lead to 1-integrable cross fields, most cross field design papers focus on this goal. This difficult computational problem has inspired a great deal of research, including [**RVAL09**, **KCPS13**, **ACBCO17**, **VO19**, **CCS$^+$21**]. Much of this research focuses on quad meshing on a closed orientable surface embedded in $\mathbb{R}^3$, where a cross field might be computed as part of a *seamless parameterization*, eg. [**BCE$^+$13**, **CSZZ19**]. In this situation there are stronger constraints on the cross field, which we avoid in our simple planar case; we return to this important topic in Section 4.14.

All singular point configurations allowed under Euler's Theorem can be quadrangulated, with the exception of the torus with two singular points of valence 3 and valence 5 [**SZC$^+$22**, **JT73**]. However, this does not directly imply that they all admit 2-integrable cross fields or that all smooth cross fields are 2-integrable.

More closely related to our work is research on the explicit construction of 2-integrable cross fields. Constructing a *frame field*, in which the two unoriented directions at a regular point need not be orthogonal and may have different lengths, was explored in [**PPTSH14**] and [**DVPSH15**]. *Odeco tensors*, studied in [**PSS21**], were used recently in a section of [**JCR23**] to construct 2-integrable frame fields, by solving a PDE with objectives equivalent to our Equations 4.3 and 4.4.

Another relevant line of research has approached the problem of constructing a frame field by constructing a metric on the domain. In [**JFH$^+$15**], a smooth orthogonal frame field is optimized so as match an arbitrary input metric as well as possible. In [**CZK$^+$19**], they construct a metric that is guaranteed to admit an orthogonal frame at every point, which is equivalent to constructing the orthogonal frame field itself (see Section 4.3).

We can handle our simple case using fairly accessible mathematics, although we will point out some relevant sophisticated concepts as we go along.

4.3. Definitions

4.3.1. Input. We assume that we are given a smooth cross field $X(\alpha)$ represented by an angle-valued function $\alpha : R \to \mathbb{R} \mod \frac{\pi}{2}$ in a region R of the plane; see Figure 4.3. Specifically, for any real-valued $a : R \to \mathbb{R}$, let $\alpha = a(x, y) \mod \frac{\pi}{2}$, and define the *cross* at (x, y) to be the unordered set of four directions $[\alpha, \alpha + \frac{\pi}{2}, \alpha + \pi, \alpha + \frac{3\pi}{2}]$, or, equivalently, the unordered set of two unoriented directions $[\alpha, \alpha + \frac{\pi}{2}]$. (Often the angle is represented by a complex number, which is easier for computation, but this representation will yield some intuitive equations later on.)

Locally, we can orient the two directions and write the cross at any point as two orthogonal unit vectors, the columns of a rotation matrix:

$$\begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$$

Cross fields are also called $4-$RoSy fields [**PZ07**] or $4-$directional symmetry fields [**RVLL08**].

A streamline in a cross field, like a streamline in a vector field or direction field, is a curve $\gamma(t)$ whose tangent at every point is the direction of the field. We called these field lines in Chapter 3. In a cross field, two unoriented streamlines pass through every regular point. Each streamline either forms a cycle or begins and ends at a boundary or singular point. In a cross field, a streamline may cross itself at right angles; that is, cycles are not necessarily *simple*.

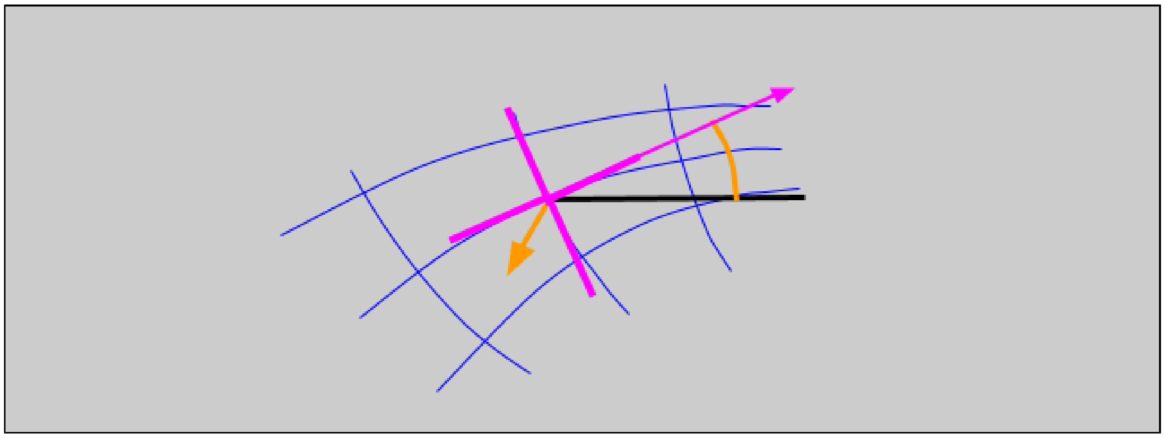


FIGURE 4.3. The cross field is given by a periodic angle-valued function $\alpha(x, y)$, in radians, defined on the region R. The range of $\alpha(x, y)$ is $[0, \pi/2]$. The cross at a point (magenta), is given by four rotations of the vector $(\cos\alpha, \sin\alpha)$. The blue curves are streamlines. The orange arc indicates α, and the orange vector indicates $\nabla\alpha$, the direction in in which the cross rotates counter-clockwise most quickly. We observe that $\nabla\alpha$ is not aligned with the streamlines.

The cross field $X(\alpha)$ may contain isolated *singular points*, at which the number of incident streamlines is not two. We only consider singular points of valence 3, incident to three streamlines, and valence 5, incident to five (these are also called singular points of *index* -1 and 1, respectively). Let S be the set of singular points.

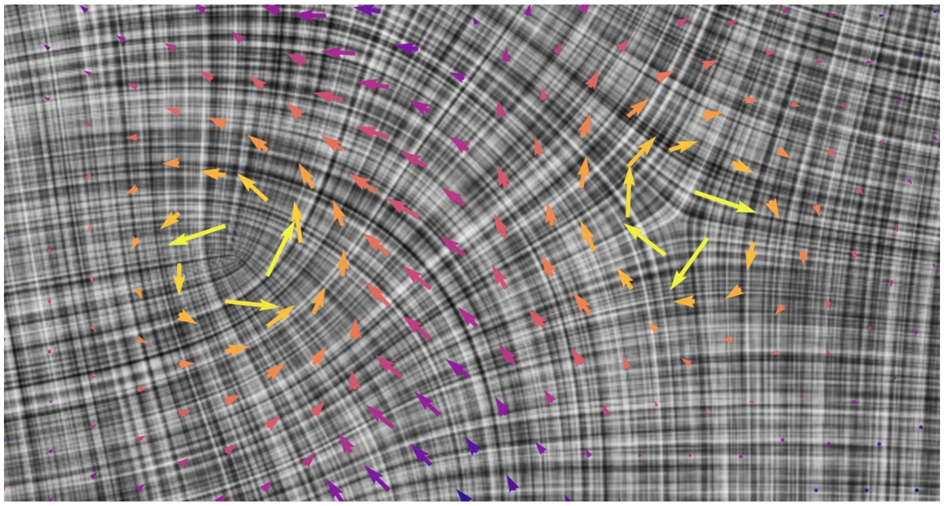


FIGURE 4.4. Some vectors from $\nabla\alpha$ superimposed on the cross field $X(\alpha)$, colored based on distance to the nearest singular points. The magnitude of $\nabla\alpha$ increases near the singular points.

In a disk-shaped neighborhood of any regular point, we can represent α using a scalar function a which may extend beyond the range $[0, \pi/2]$. In particular, the vector field $\nabla\alpha = \nabla a$ is well-defined, and it forms (like all gradients) a vector field with zero curl, although possibly with non-zero divergence (recall that the curl of vector field $(u(x, y), v(x, y))$ is $v_x - u_y$, and the divergence is $u_x + v_y$). The magnitude of $\nabla\alpha$ increases near a singular point s. The singular points of the vector field $\nabla\alpha$ are the singular points of $X(\alpha)$, and α is undefined at the singular points.

A *limit cycle* is a cycle into which other streamlines "spiral in"; that is, a streamline γ is a cycle and there is another streamline ζ such that $\zeta(t)$ approaches γ as t goes to infinity; see Figure 4.2. X is smooth at a limit cycle γ. Since we forbid limit cycles, the neighborhood of a cycle streamline is covered by a set of nearly parallel cycles.

The cross field X is defined on a region R of the plane. We can require X to be aligned with the boundaries of R, or not; but we do not allow periodic boundaries, at which X would have to match.

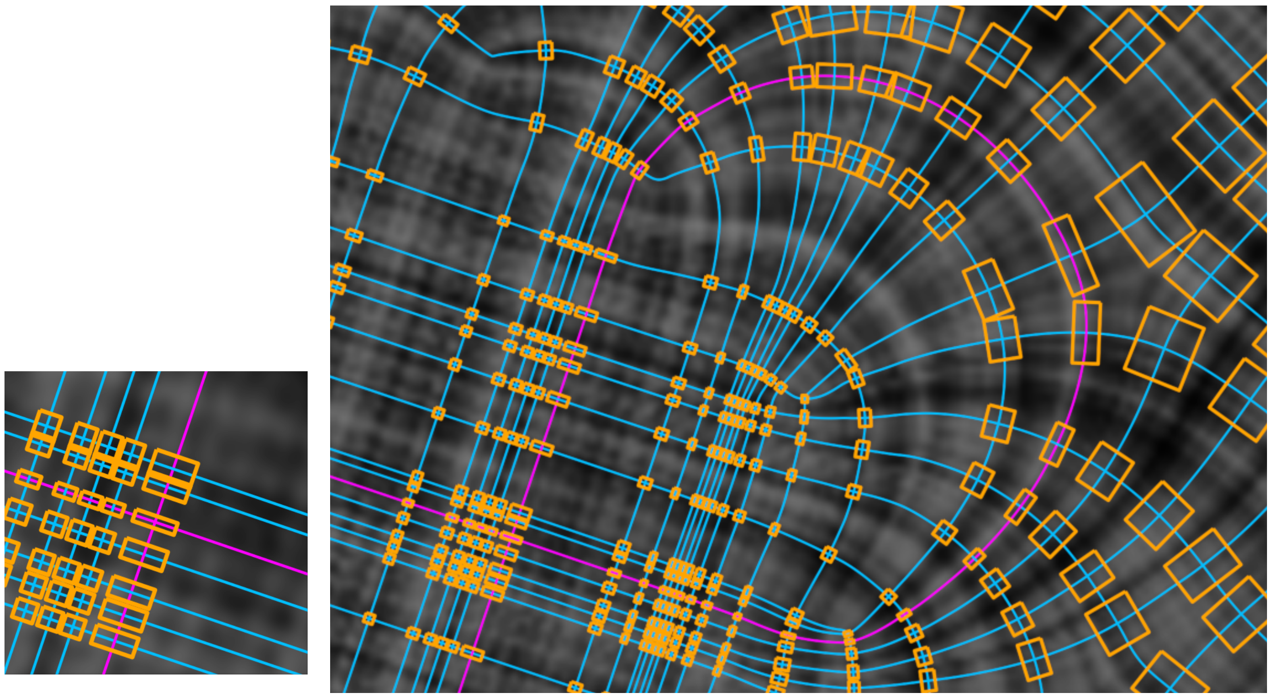


FIGURE 4.5. Sometimes streamlines, like the one highlighted in magenta, can cross themselves, locking the aspect ratio $\sigma : \tau$ at the point of intersection. This shows that some cross fields require anisotropy.

4.3.2. Orthogonal frame fields. Our goal will be to assign a length function along every undirected streamline, and hence two lengths at every regular point. Since, for example, a streamline can cross itself at right angles, as in Figure 4.5, we usually cannot define the lengths as two separate continuous scalar functions on R. Instead, we will focus on a local neighborhood of every regular point and require the length functions to be smooth in every neighborhood. Then we will make a global argument to show that the functions can be globally as well as locally smooth.

Let us define some local notation. In a neighborhood $N(p)$ of any regular point p, the two sets of streamlines, and their length functions, can be separated. Within the neighborhood, we call the two length functions $\sigma(x, y), \tau(x, y)$ (even if, globally, both come from the same set of streamlines). Also locally, we can orient both sets of streamlines. Now we can write a local orthogonal frame as two vectors, or the columns of a matrix:

$$\begin{bmatrix} \sigma \cos \alpha & -\tau \sin \alpha \\ \sigma \sin \alpha & \tau \cos \alpha \end{bmatrix}$$

In the next section, we will see that the change in σ or τ along a streamline is determined by the curvature of the orthogonal set of streamlines. We use these relationships to give the following definition.

DEFINITION 4.3.1. *A cross field is 2-integrable if we can define a length function for each direction vector at each point, creating an orthogonal anisotropic frame field, such that*

(1) (smoothness) in the neighborhood of any point $p \in R$, the two scalar length functions are both smooth (say, C^2-continuous), and

(2) (α-consistency) along any streamline, the associated length function is consistent with α, as defined by Equations 4.3 and 4.4, below.

4.4. Local parameterization

What is stopping us from assigning, for example, $\sigma = \tau = 1$ everywhere? Unless we are extremely lucky, this will not correspond to the aspect ratios of any possible set of infinitesimal rectangles aligned with $X(\alpha)$. We capture this idea as follows. Let $P_{x,y}$ and $P_{u,v}$ be two copies of the Euclidean plane. Assume there is a smooth map M from some neighborhood in $P_{u,v}$ to $N(p)$, the neighborhood of $p \in R \subset P_{x,y}$. Under map M, infinitesimal axis-aligned squares in $P_{u,v}$ are mapped to rectangles aligned with the cross field in $P_{x,y}$. See Figure 4.6.

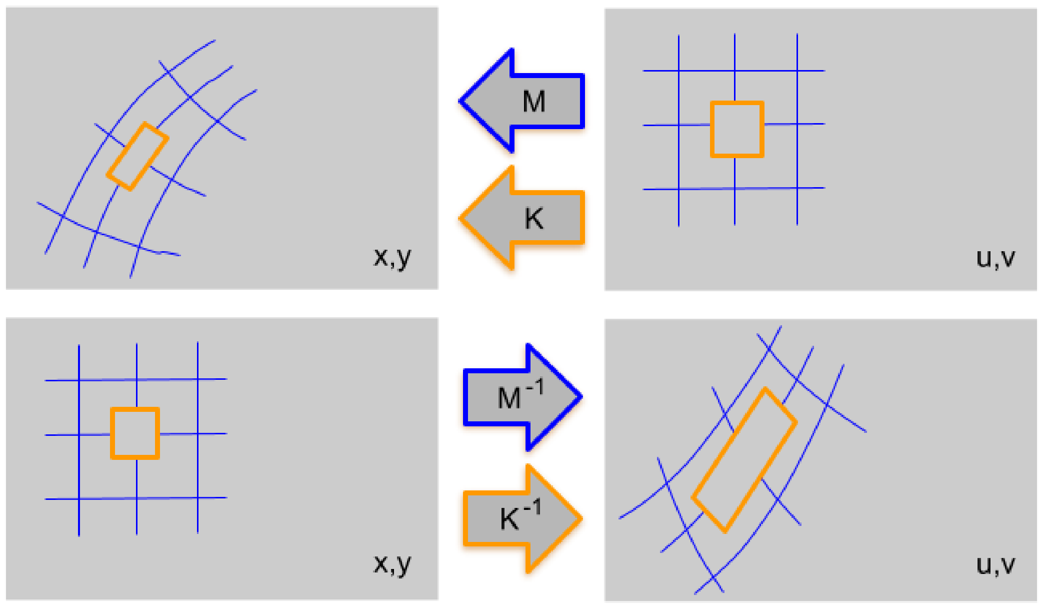


FIGURE 4.6. On any neighborhood, we consider a local map M from plane $P_{u,v}$ into plane $P_{x,y}$. The Jacobian K of M takes infinitesimal axis-aligned squares in (u, v) to infinitesimal rectangles aligned with the cross field in (x, y). The inverse Jacobian takes infinitesimal squares in (x, y) to parallelograms in (u, v).

We do not need to write down M explicitly in order to understand the local behavior of infinitesimal squares. We are just interested in its derivative, the Jacobian matrix:

$$K = \begin{bmatrix} x_u & x_v \\ y_u & y_v \end{bmatrix} = \begin{bmatrix} \sigma\cos\alpha & -\tau\sin\alpha \\ \sigma\sin\alpha & \tau\cos\alpha \end{bmatrix}$$

Here, σ, τ and α are functions of (x, y), but $x(u, v), y(u, v)$ are functions of u, v.

Locally, the inverse map M^{-1} always exists, and its Jacobian K^{-1} is the inverse of the Jacobian of M:

$$K^{-1} = \begin{bmatrix} u_x & u_y \\ v_x & v_y \end{bmatrix} = \begin{bmatrix} (1/\sigma)\cos(-\alpha) & -(1/\sigma)\sin(-\alpha) \\ (1/\tau)\sin(-\alpha) & (1/\tau)\cos(-\alpha) \end{bmatrix} \tag{4.1}$$

It is easier to figure out how σ, τ have to change with α by looking at K^{-1} because its elements u_x, u_y, v_x, v_y are all functions of (x, y).

We note that K^{-1} also determines a Riemannian metric at p. If $\gamma(t)$ is a unit-speed curve in (x, y), then the length of γ using the metric $K^{-T}K^{-1}$ is the length of $M(\gamma(t))$ in (u, v):

$$\int_{t=0}^{\text{length }\gamma} \left[(K^{-1}\,\gamma_t)^T(K^{-1}\,\gamma_t)\right]^{1/2} dt \tag{4.2}$$

We also observe that the rows, rather than the columns, of K^{-1} are orthogonal (unless $\sigma = \tau$), so K^{-1} is not in general an orthogonal frame, and it maps squares to parallelograms.

The functions u, v have to satisfy

$$(u_x)_y = (u_y)_x$$

$$(v_x)_y = (v_y)_x$$

That is, ∇u and ∇v, like all gradients, have zero curl. From $(u_x)_y = (u_y)_x$ we have

$$\begin{aligned} & ((1/\sigma)\cos\alpha)_y = ((1/\sigma)\sin\alpha)_x \\ \iff \quad & -\sigma_y \cos\alpha - \sigma \sin\alpha\ \alpha_y = -\sigma_x \sin\alpha + \sigma\cos\alpha\ \alpha_x \\ \iff \quad & \nabla\sigma\cdot(-\sin\alpha, \cos\alpha) = -\sigma\ \nabla\alpha\cdot(\cos\alpha, \sin\alpha) \end{aligned}$$

and using $(v_x)_y = (v_y)_x$ we have

$$\begin{aligned} & ((-1/\tau)\sin\alpha)_y = ((1/\tau)\cos\alpha)_x \\ \iff \quad & \tau_y \sin\alpha - \tau\cos\alpha\ \alpha_y = -\tau_x\cos\alpha - \tau\sin\alpha\ \alpha_x \\ \iff \quad & \nabla\tau\cdot(\cos\alpha, \sin\alpha) = \tau\ \nabla\alpha\cdot(-\sin\alpha, \cos\alpha) \end{aligned}$$

We highlight these two partial differential equations.

$$\nabla\sigma\cdot(-\sin\alpha, \cos\alpha) = -\sigma\ \nabla\alpha\cdot(\cos\alpha, \sin\alpha) \tag{4.3}$$

$$\nabla\tau\cdot(\cos\alpha, \sin\alpha) = \tau\ \nabla\alpha\cdot(-\sin\alpha, \cos\alpha) \tag{4.4}$$

(One can also derive these equations by requiring a Lie bracket to be the zero vector, as in [**JCR23**]. Alternatively, one could relate the metric $K^{-T}K^{-1}$ to the curvature of $\nabla\alpha$ via the Levi-Civita connection.)

4.5. Isotropic cross fields

If $\tau = \sigma$ everywhere, then the maps M, M^{-1} are *conformal*, and the isotropic metrics $K^T K, K^{-T}K^{-1}$ are called *isothermal*. Most cross field design programs optimize towards conformal maps (by minimizing the Dirichlet energy or the Ginzberg-Landau energy, or following the Ricci flow, etc.). With $\tau = \sigma$, our two differential equations become

$$\nabla\sigma\cdot(-\sin\alpha, \cos\alpha) = -\sigma\ \nabla\alpha\cdot(\cos\alpha, \sin\alpha)$$

$$\nabla\sigma\cdot(\cos\alpha, \sin\alpha) = \sigma\ \nabla\alpha\cdot(-\sin\alpha, \cos\alpha)$$

Thinking of the vectors $(\cos\alpha, \sin\alpha)$ and $(-\sin\alpha, \cos\alpha)$ as an orthonormal coordinate system, we see that

$$\sigma_x = -\sigma\alpha_y \tag{4.5}$$

$$\sigma_y = \sigma\alpha_x \tag{4.6}$$

So in a region where $\sigma = \tau$, there is a unique solution for σ given α, up to a global scale factor. We can think of Equations 4.5 and 4.6 as a way of writing the usual Cauchy-Riemann equations $u_x = v_y$ and $u_y = -v_x$, which describe a conformal map.

In order for the differential equations 4.5 and 4.6 to be solvable, however, (σ_x, σ_y) has to have zero curl. That is, at every point in R,

$$(\sigma\alpha_x)_x = (-\sigma\alpha_y)_y$$

$$\sigma_x\alpha_x + \sigma\alpha_{xx} = -\sigma_y\alpha_y - \sigma\alpha_{yy}$$

$$-\sigma\alpha_x\alpha_y + \sigma\alpha_{xx} = -\sigma\alpha_x\alpha_y - \sigma\alpha_{yy}$$

which shows that α must be harmonic (that is, $\alpha_{xx} + \alpha_{yy} = 0$), and therefore $\nabla\alpha$ has zero curl and zero divergence at every point. Clearly, there are functions a which are not harmonic, and thus cross fields α that are not harmonic.

4.6. Singular points

In a neighborhood containing a singular point s of odd valence, we cannot separate the streamlines into two distinct sets; so the functions σ, τ can be smooth on the entire neighborhood only if they are identical at s.

OBSERVATION 4.6.1. *At a singular point s, $\sigma(s) = \tau(s)$.*

So to understand the relationship of σ and α at singular points we can examine the well-understood isotropic case. A singular point s of valence 3 is similar (scaled, rotated and/or translated) to the singular point at the origin induced by the complex map $M^{-1}(z) = z^{3/4}$, which we can write in terms of (x, y) and (u, v) as

$$M^{-1}(x, y) = (x^2 + y^2)^{3/8} e^{(3/4)i \arctan(y/x)} = (u, v)$$

We can similarly write the complex derivative $K^{-1}(z) = 3/4z^{-1/4}$ as

$$K^{-1}(x, y) = 3/4(x^2 + y^2)^{-1/8} e^{(-1/4)i \arctan(y/x)} = (u, v)$$

From Equation 4.1, we see that $1/\sigma(x, y) = 3/4(x^2 + y^2)^{-1/8}$ and $-\alpha = (-1/4)\arctan(y/x)$. Recalling that $\nabla \arctan(y/x) = (-y, x)/(x^2 + y^2)$, we find

$$\begin{aligned} \sigma =& 4/3\ (x^2 + y^2)^{1/8} \\ \implies \qquad \nabla\sigma =& 1/3\ (x^2 + y^2)^{-7/8}(x, y) \\ \alpha =& 1/4\ \arctan y/x \\ \implies \qquad \nabla\alpha =& 1/4\ (x^2 + y^2)^{-1}(-y, x) \end{aligned}$$

We can verify that these equations obey Equations 4.5 and 4.6. They imply a counter-clockwise rotation of $\nabla\alpha$ around s (as in Figure 4.4) and a zero of σ at s.

Similarly, a singular point s of valence five at the origin is produced by $M^{-1}(z) \to z^{5/4}$, with $K^{-1}(z) = 5/4\ z^{1/4}$. In this case we find that $\nabla\alpha$ rotates clockwise, and s is a pole.

In either case, we avoid tracing streamlines very close to singularities due to the difficulty in accurately estimating σ and τ as $\nabla\alpha$ grows without bound.

4.7. Integrating σ and τ

When α is not harmonic, τ and σ differ. Equations 4.3 and 4.4 are not sufficient to determine $\nabla\sigma$ and $\nabla\tau$. But they do completely determine two directional derivatives per point, one for σ and one for τ. (This section is an example of the "method of characteristics" for solving first-order PDEs.)

In particular, for a unit-speed curve $\gamma(t)$ through a point $p \in R$, with unit direction vector $(-\sin\alpha, \cos\alpha)$, the directional derivative

$$\frac{d\sigma}{dt} = \nabla\sigma\cdot(-\sin\alpha, \cos\alpha) = -\sigma\ \nabla\alpha\cdot(\cos\alpha, \sin\alpha)$$

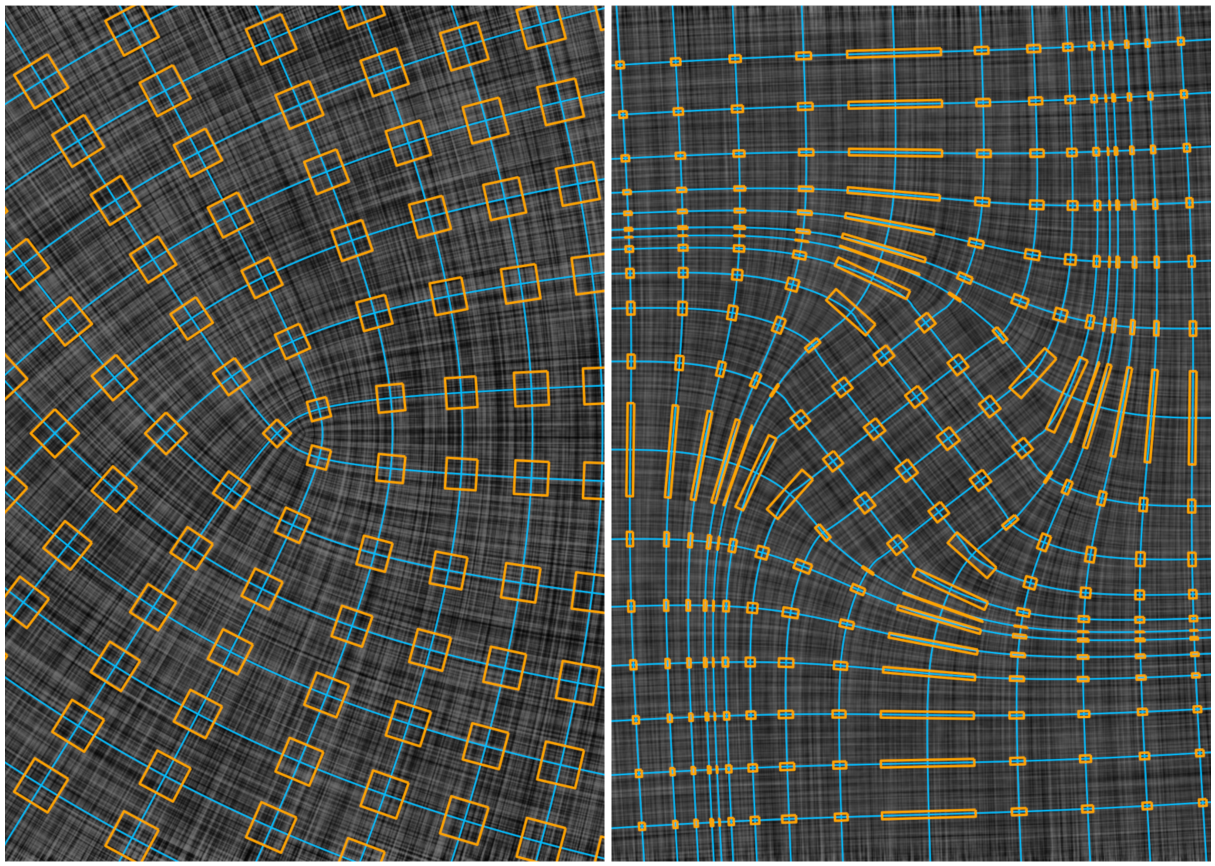


FIGURE 4.7. Given the isotropic cross field derived from by the conformal map $z \rightarrow z^{3/4}$ (left) we (of course) find that the minimum anisotropy is zero; that is, we can set $\sigma = \tau$ everywhere. A cross field with nonzero divergence (right) exhibits large anisotropy, even in some regions where $\nabla\alpha = 0$. We computed this twisting cross field by requesting a large isotropic scale in the center and a smaller isotropic scale on the outside, and minimizing the Dirichlet energy. Like most sets of user-defined cross field constraints, there is no harmonic α function that meets them. The non-harmonic $X(\alpha)$ we found does achieve the large isotropic scale in the center and smaller isotropic scale in the corners, but requires large anisotropy. Requesting a larger scale difference in this case produces singular points, which allows the scale change with lower anisotropy.

So, for a streamline following $(-\sin\alpha, \cos\alpha)$, given a single value of σ somewhere along the streamline, we can find all the other values of σ along the streamline by integrating using Equation 4.3.

Similarly we can integrate τ along the streamlines with direction $(\cos\alpha, \sin\alpha)$, using Equation 4.4:

$$\frac{d\tau}{dt} = \nabla\tau \cdot (\cos\alpha, \sin\alpha) = \tau \; \nabla\alpha \cdot (-\sin\alpha, \cos\alpha)$$

Fortunately, these are exactly the streamlines we have.

For example, consider an intrinsic rectangle not containing a singular point. We can set arbitrary values of σ along one edge and arbitrary values of τ along another, and then integrate both into the interior using equations 4.3 and 4.4. If the initial values of σ are smooth along the edge, the two-dimensional σ function across the interior of the intrinsic rectangle will be smooth, and similarly for τ.

Recall that the variables σ and τ are only defined on a small local neighborhood not containing a singularity. As seen in Figure 4.5, a streamline in a cross field can intersect itself at right angles, and at the intersection point σ and τ are defined by different points along the same field line. At points of self-intersection like these, the ratio between σ and τ is fixed, since any change to one value on the streamline changes the other.

4.8. Global consistency of σ and τ

The previous section established that we can locally integrate σ and τ along streamlines within a disk-shaped singularity-free neighborhood. So the length function along an entire streamline γ is determined if we set the value at any one point $p \in \gamma$. A connected set of adjacent streamlines whose length values are continuous along any curve orthogonal to them all will have continuous values throughout their length.

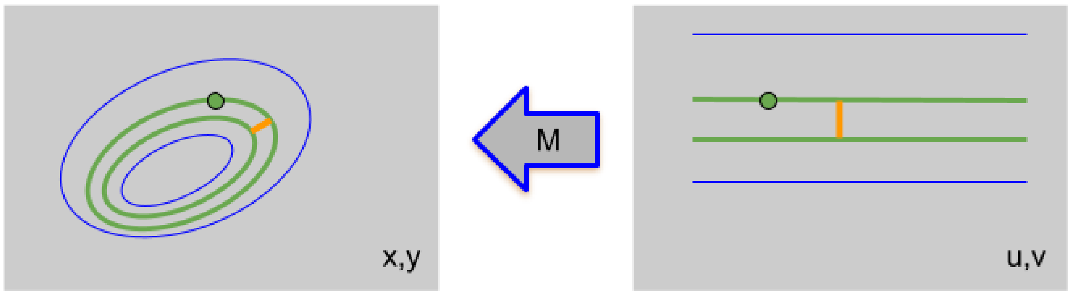


FIGURE 4.8. Illustration for Theorem 4.8.1. Mapping a point p on cycle $\gamma(0)$ to parameter space $P_{u,v}$ takes the cycle to a line. A parallel line at distance ϵ maps back (pulls back) to a nearby cycle. The distance between the two cycles in $P_{x,y}$ always returns to the same value as we loop around, and converges to $\epsilon\tau(p)$ as $\epsilon \to 0$.

Self-intersecting streamlines are not a problem. But what about cycle streamlines? Say γ is a cycle following direction $(\cos\alpha, \sin\alpha)$. If we fix the value of $\tau(p)$ at one point $p \in \gamma$, and we integrate along γ using Equation 4.4 and return to p, do we always get the value of $\tau(p)$ that we started with? Notice that this is not true of every function; for instance, integrating $\nabla\alpha$ around a cycle gives 2π rather than zero, even though $\nabla\alpha$ has zero curl.

THEOREM 4.8.1. *Let γ be a cycle streamline, not a limit cycle, with no singular point in its neighborhood. Let τ be the length function orthogonal to γ. The function τ is smooth everywhere on γ.*

PROOF. Since γ is not a limit cycle, a small neighborhood of γ is covered by a set of nearby disjoint parallel cycle streamlines.

Let $\gamma_0 = \gamma$. Recall the mappings $M : (u, v) \to (x, y)$ and $M^{-1} : (x, y) \to (u, v)$ from Section 4.4. At an arbitrary point $p \in \gamma$, let $q = M^{-1}(p)$. See Figure 4.8. From q, a an ϵ step in the v direction takes us to point $q + \epsilon v$, which is mapped by K to step orthogonal to γ_0, landing us on a nearby cycle γ_ϵ. As ϵ goes to zero, the distance from $p \in \gamma_0$ to the nearest point on γ_ϵ converges to the infinitesimal width

$$w(p, \epsilon) = \left[\begin{bmatrix} 0 & \epsilon \end{bmatrix} \begin{bmatrix} \sigma \cos \alpha & \sigma \sin \alpha \\ -\tau \sin \alpha & \tau \cos \alpha \end{bmatrix} \begin{bmatrix} \sigma \cos \alpha & -\tau \sin \alpha \\ \sigma \sin \alpha & \tau \cos \alpha \end{bmatrix} \begin{bmatrix} 0 \\ \epsilon \end{bmatrix} \right]^{1/2} = \tau \epsilon$$

with $w(p, \epsilon)/\epsilon$ converging to τ. Since cycles are smooth and closed, if we traverse all the way around and return to p, we get the same infinitesimal width $w(p, \epsilon)$ and thus we converge to the same value of τ. $\square$

Setting the value of $\sigma = \tau$ at a singular point s determines the length function along the (3 or 5) separatrices adjacent to s. We required each of these separatrices to end at the boundary, not at another singular point. Hence,

OBSERVATION 4.8.2. *We can choose the values of $\sigma = \tau$ at each singular point independently of each other.*

THEOREM 4.8.3. *There is an infinite one-dimensional set of ways to choose the length functions along each streamline, consistent with σ, τ being smooth on the neighborhood of any point in R.*

PROOF. First, we select the size values $\sigma = \tau$ at the singular points. There is a discrete finite set of such values, and we can choose them independently because of Observation 4.8.2. This choice determines the length values along the separatrices. The separatrices divide the remaining streamlines into groups, such that we cannot deform a streamline from one group into one from another group without crossing one or more singular points, or leaving the region (that is, the groups are *homotopic*).

Some groups may be unions of cycles, while others begin and end on the boundary. Within each group, we can pick any one-dimensional curve segment g crossing each streamline in the group once (for example, g might be a segment of one of the streamlines in the orthogonal direction, or a boundary curve). We pick a smooth length function along g, respecting any length values already fixed at the endpoints because they lie on separatrices. This fixes the length values all along the entire group of streamlines. The length values are continuous at the separatrices since they were continuous at g. All of the length values along all of the segments g form an infinite one-dimensional set, parameterizing all the possible length assignments. □

Now that we have a globally-integrable size representation, we can finally explain the paradox encountered in Chapter 3, specifically Figure 3.29. We will refer to a similar, mirrored field in Figure 4.7 (right), where chords seem to be narrowing when traveling in a counter-clockwise path. Along any of these counter-clockwise cycles through the region, as τ shrinks, σ is growing, and vice-versa. The fact that the change in σ or τ aligned with the direction of travel is not directly observable from the cross field itself, unlike the change in σ or τ perpendicular to the direction of travel which can be approximated by the relative distance of neighboring field lines, was an obstruction along the path to considering a two-component size function to begin with.

4.9. Divergence of $\tau\nabla\alpha$

In this section we comment on the behavior of $\nabla\alpha$ on a cycle streamline. Let γ be a cycle streamline γ, not a limit cycle, and let us use τ as the length function that is integrated along γ.

The *flux* of the vector field $\nabla\alpha$ across a cycle γ is

$$\int_\gamma \nabla\alpha\cdot(-\sin\alpha, \cos\alpha)ds$$

The usual physical intuition is that if α is the flow of some quantity as a function of time, the flux is the net movement of the quantity across the curve γ per unit time.

OBSERVATION 4.9.1. *If $x > 0$, we have $(\ln x)_x = 1/x$. So*

$$\nabla \ln\tau = \frac{\nabla\tau}{\tau}$$

LEMMA 4.9.1.1. *The flux of the vector field $\nabla\alpha$ is zero across any cycle streamline γ.*

PROOF. The proof of Theorem 4.8.1 showed that the function τ is smooth everywhere on cycle γ, and thus, so long as $\tau > 0$ everywhere, the function $\ln \tau$ is also smooth everywhere on γ. So

$$\begin{aligned} 0 &= \\ \int_\gamma \nabla(\ln\ \tau)\cdot(\cos\alpha, \sin\alpha) ds &= \\ \int_\gamma \frac{\nabla\tau}{\tau}\cdot(\cos\alpha, \sin\alpha) ds &= \\ \int_\gamma \nabla\alpha\cdot(-\sin\alpha, \cos\alpha) ds & \end{aligned}$$

□

If γ is a simple cycle, the flux is also the right-hand side of Gauss's divergence theorem:

$$\int\int_{\text{interior } \gamma} \nabla\cdot\nabla\alpha dA = \int_\gamma \nabla\alpha\cdot(-\sin\alpha, \cos\alpha) ds$$

That gives us:

OBSERVATION 4.9.2. *Let γ be a simple cycle streamline. The total divergence of the vector field $\nabla\alpha$ in the region bounded by γ must be zero.*

This tells us, on the one hand, that simple cycles only appear in quite special cross fields, since in general the divergence of $\nabla\alpha$ at an arbitrary point is not zero. The situation in which $\nabla\cdot\nabla\alpha$ is everywhere zero is the useful case in which α is everywhere locally harmonic.

On the other hand, since divergence can be positive or negative, the total divergence of $\nabla\alpha$ in the interior of a cycle streamline γ may be zero without $\nabla\cdot\nabla\alpha = 0$ at any point in the interior.

4.10. Definition of anisotropy

For non-harmonic α, we need a functional that we can optimize to choose among the valid values of σ, τ. In this section we consider minimizing the anisotropy A. We can optimize it as follows:

$$\min_{\sigma,\tau} \int_R A(x,y)dA$$

s.t.

$$\left.\begin{aligned} -(\sigma_y)\cos\alpha - \sigma\,\alpha_y \sin\alpha = {} & -(\sigma_x)\sin\alpha + \sigma\,\alpha_x \cos\alpha \\ (\tau_y)\sin\alpha - \tau\,\alpha_y \cos\alpha = {} & -(\tau_x)\cos\alpha - \tau\,\alpha_x \sin\alpha \end{aligned}\right\} \text{in } R,$$

$$\int_R (\sigma^2 + \tau^2)dA = 1$$

Making $\sigma^2 + \tau^2$ integrate to one over the whole region keeps them from going to zero.

There are many ways we could define the anisotropy function A. We use:

$$A = \left[\frac{\sigma}{\tau} + \frac{\tau}{\sigma} - 2\right]^{2k} \tag{4.7}$$

The exponent $k \geq 1$ is an integer. Raising the anisotropy to an even power ensures it is positive, although the constant offset -2 sufficiently penalizes negative aspect ratios that we do not see negative σ or τ in practice. The greater k, the more it penalizes the maximum anisotropy over R. Subtracting two from the sum of the two ratios gives an energy function that equals zero for isotropic size functions.

4.11. Discretization, optimization and visualization

In order to perform the optimization, we first select a sparse set L of streamlines to represent the cross field. For each $\ell \in L$, we arbitrarily set $\sigma(p)$ for some point in ℓ, which determines the σ values along the streamline, based on Equation 4.3. We then define a scalar multiplier for each streamline ℓ, which changes all of its σ values simultaneously. Each intersection between two streamlines in L is a point where both σ and τ are defined. We evaluate the function A from Equation 4.7 at each of these points.

Minimizing the anisotropy measure is achieved through an iterative process. We resize σ along each streamline $\ell \in L$ in turn so as to minimize the sum of the anisotropy measures of all intersections involving ℓ. If a pass through all streamlines yields no improvement, or an improvement below a given threshold, the optimization is complete. Since the total sum of the anisotropy measures at all intersections is reduced at each step, this process terminates.

To select the lines in L, we start with a subset T of streamlines that captures the topology of X. These may be the separatrices (as in Figure 4.1) or they may be regular streamlines (as in Figure 4.10). Along the streamlines in T, we select orthogonal streamlines to include in L until every point on a line in T is sufficiently close to a streamline of L. We do not use separatrices in L as growth of $\nabla\alpha$ near singularities makes robust size estimation difficult.

As a check, we find that this algorithm behaves well on conformal cross fields, even though we do not use Equations 4.5 and 4.6. The conformal cross field in Figure 4.7 (left) exhibits a worst-element anisotropy of $1 : 1.0033$.

We visualize anisotropy in two ways. In low-anisotropy cases, we draw an orange rectangle centered at every intersection point p of two lines in L, reflecting the aspect ratio and sizes given by $\sigma(p), \tau(p)$. One might expect that there is some scale factor that would grow all of the squares to nicely "collide" with their neighbors, creating a rough approximation of a mesh; this is not true because L is chosen before we find σ and τ. When anisotropy is very large, we visualize it instead using a color scale as in Figure 4.10. There, σ and τ are interpolated over the region, and a per-pixel anisotropy is calculated (except for near singularities along the wing surface where the singularity is not fully enclosed by representative streamlines).

4.12. Removable limit cycles

Simple limit cycles appear in vector fields as well as cross fields and are studied in dynamical systems. There are two kinds of limit cycles, stable and unstable. Unstable limit cycles do not cause much trouble; they disappear under small smooth perturbations of α, while stable limit cycles persist. A stable limit cycle γ is one in which, as we traverse γ, the nearby streamlines on either side either all approach or all move away from γ (depending on our direction of travel on γ). See Figure 4.9.

Some limit cycles in cross fields are forced by the topology, for example, the torus with one singular point of valence 3 and one of valence 5 must have a limit cycle which is also a fundamental cycle of the torus; this is why it cannot be meshed. But limit cycles may also appear where they are not forced.

As we see in Figure 4.9, a stable simple limit cycle γ in a planar cross field can be removed by replacing a small neighborhood around γ, without adding new singular points. Before removal, on one side of γ, one set of streamlines cycles in towards γ and the other set crosses γ, and similarly on the other side. To remove γ, we smoothly add a twist of $\pi/2$ to α in the neighborhood of γ, so that each streamline spiraling into the

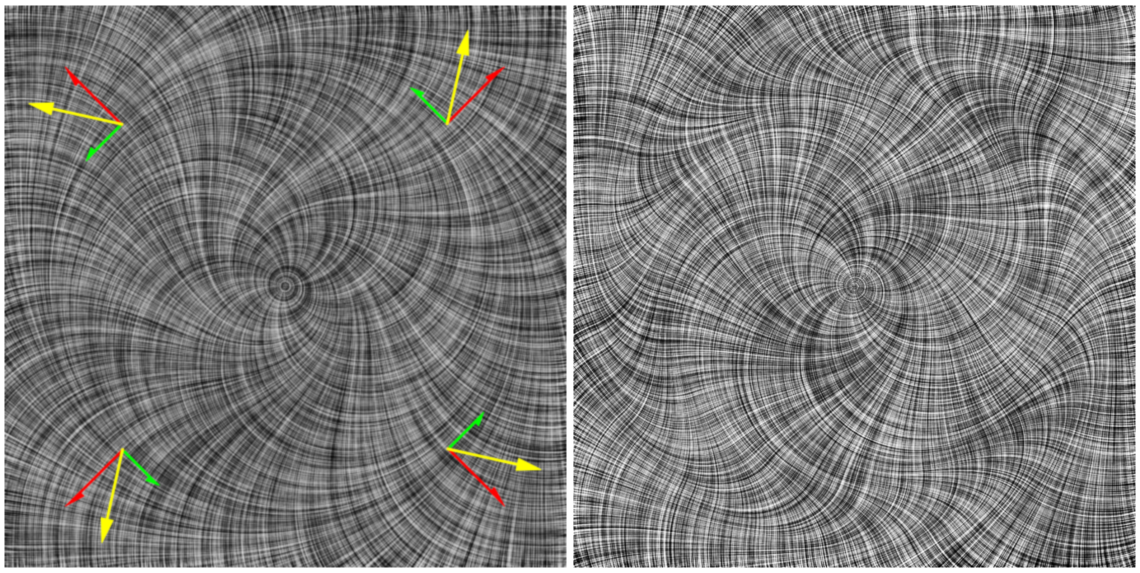


FIGURE 4.9. A simple stable limit cycle (left) can be smoothly replaced with a region within which a twist of $\pi/2$ swaps the two sets of streamlines (right). The streamlines away from the cycle remain unchanged, and the region in which the swap occurs can be made arbitrarily thin. An exaggerated $\nabla\alpha$ vector (yellow) has been drawn on top of the left image, along with the components tangential (green) and normal (red) to the limit cycle; limit cycles are distinguished by the nonzero normal component (and hence have a non-zero net flux of $\nabla\alpha$.)

neighborhood is connected to a streamline crossing γ, and visa versa. Since we need two sets of streamlines, this trick does not work on vector fields.

4.13. Results

The cross field in Figure 4.1 is the upper-right portion of a cross field computed using the adaptation of [**KCPS13**] described in Chapter 3; it was given initial conditions requiring small isotropic elements over a region in the center and large isotropic elements on the outside, as in Section 3.4.2.

In Figure 4.10, we show a cross field generated from a two-dimensional simulation of airflow over a wing, NACA Airfoil 8650 from [**SQB21**]. The original flow vector field had more than one hundred singularities. We created a cross field by adding the orthogonal direction at each point and then smoothed it. This reduced the number of singularities and had the additional benefit of removing two ellipse-shaped limit cycles corresponding to vortices in the fluid flow. Each limit cycle was replaced by a pair of valence-three singularities surrounded by a lens-shaped region with less total curvature. The central part of the image has very low anisotropy as the vertical path lines can be freely adjusted to match the horizontal path lines. The left side of the image has slightly more anisotropy as those collisions represent streamlines that form cycles around the entire wing. The turbulent airflow to the right of the wing creates extreme anisotropies as areas of high rotation in the field require tiny chord widths. The propagation of these extreme anisotropies away from the turbulent region could be fixed by adding more singularities as we move into the non-turbulent region.

In the following examples, we look at the minimum-anisotropy size function compatible with a cross field that was generated from a size function as described in Chapter 3. Given the target size function s from which the cross field was derived, the accuracy at any given pixel is the ratio of the computed size function $\sqrt{\sigma\tau}$ and the target size s. In the images, blue represents the fact that the size function $\sqrt{\sigma\tau}$ is smaller than the target size function s, and red indicates that it's larger, both getting darker as the ratio gets more extreme.

In Figure 4.11 we see the anisotropy for the Sacramento Valley dataset. It exhibits the same field-aligned streaks as the airfoil dataset, where singularities induce high anisotropy which propagate along the nearby streamlines. Unlike the CFD simulations that would be performed on the airfoil dataset, where a large volume around the wing is necessary for the simulation and thus adding additional singularities can serve to limit the anisotropy in important regions of the mesh, the exterior anisotropy here is not a concern because it's going to be trimmed anyway, as discussed in Chapter 2.

4.14. Discussion

The main theoretical limitation of this work is that it only applies to bounded regions in the plane. The interesting case of a cross field where all streamlines have finite length on a two-dimensional surface embedded in $\mathbb{R}^3$ is much more structured. The separatrices all begin and end at singular points, and all other streamlines must be cycles. Observation 4.8.2 does not hold. Might there be a cycle of separatrices for which there is no set of lengths obeying Equations 4.3 and 4.4 and Observation 4.6.1? In [**CZK+19**], they use a version of the "tiny bricks" argument to establish that we can consistently assign two length functions at every point away from the separatrices, but it is not clear that these length functions can or must be continuous at the separatrices or singular points.

A particular quirk of our cross-field generation process, the focus of Chapter 3, is that each of our cross fields $X(\alpha_0)$ are produced in such a way that the cross field $X(\alpha) = X(\alpha_0 + \phi)$ is also a valid solution, for any constant value ϕ, which we call the *phase* of the field. While the content in this chapter focuses on the analysis of fixed cross fields, the key factor in computing σ and τ is $\nabla\alpha$, which does not change with a change in phase ϕ. However, a change in phase ϕ does change the resulting field lines, changing the connectivity of the surface and thus the resulting optimal size function, except, again, in the case of a harmonic α which allows an isotropic size function $\sigma = \tau$. Figure 4.12 shows the relative size of $\sqrt{\sigma\tau}$ versus the desired size function s, for two different phases $\alpha_1 = \alpha_0 + \frac{\pi}{4}$. While largely similar, the extreme values in both directions are larger in magnitude for the right image, and there are regions throughout that are too large in one and too small in the other. While we expect the results of any two phases to generally agree, changing the connectivity over which σ and τ are optimized can have a large impact on the resulting size function's accuracy and anisotropy. We cannot yet offer intuition on how to select an optimal ϕ, as we have not observed a useful correlation between ϕ and *e.g.* size accuracy.

4.14.1. Implementation Caveats. Forward Euler integration has been used here for tracing streamlines, with a fixed step size s, and for improved accuracy the paths are *oversampled*: for every point saved along a field line, k steps of length ${}^{s}/_{k}$ are taken, with the intermediate points discarded.

Just as the streamlines themselves are approximated by this integrated polyline, calculating σ and τ depends on the integral of $\nabla\alpha$, which is assumed to be constant over each segment of the polyline. Unfortunately, this leads to a significant loss of accuracy near singularities. It is not necessarily a problem of sample density, but of bias; as a streamline passes a singularity, if there is a bias towards the singularity on one side and a bias away on the other, the computed change in σ on one side of the singularity will not match the opposing change in the other. Let us consider a simpler problem that exhibits the same behavior: summing over the range ± 10 of an odd function with a 10% bias:

```
sum(atan(i*0.01        ) for i in range(- 1000, 1001))  → -3.7214675785435247e-13
sum(atan(i*0.01  - 1e-3) for i in range(- 1000, 1001))  → -0.2942354325191827
sum(atan(i*0.001       ) for i in range(-10000,10001))  → -1.425082274408851e-12
sum(atan(i*0.001 - 1e-4) for i in range(-10000,10001))  → -0.2942265249199314
```

As demonstrated in the last two lines, this disparity is not eliminated by finer sampling; finer sampling can reduce the bias observed between some pair of approaching and receding sample points, but also increases the number of sample points taken, so it may even increase the integrated bias!

However, $\nabla\alpha$ does not necessarily need to be approximated in this manner; if α is stored as a scalar value at the vertices of a background triangle mesh, and linearly interpolated over the triangles, $\nabla\alpha$ is constant over every triangle in the background mesh. Then it is a matter of ensuring that the streamlines are properly discretized along triangle boundaries. Unfortunately, this representation does not work for singular triangles, which is where it is most needed, because barycentric interpolation of scalar angles cannot be performed over a singular triangle; there is no consistent assignment of angles at the three vertices such that barycentric interpolation correctly follows the correct interpolation along all three edges.

We propose a workaround: force $\nabla\sigma = \nabla\tau = 0$ within some distance of every singularity. It is important to keep these regions centered on the singularities to avoid recreating the same bias-induced errors. Values of σ and τ are not likely to be particularly reliable near a singularity anyway as $\nabla\alpha \to \infty$, so the loss of utility in having the values of σ and τ clamped near singularities is minor. On the other hand, it does mean that the influence from distant singularities is incorrectly clamped to zero, suggesting that these regions should be kept as small as possible.

As we have not implemented this proposed fix yet, we instead tried to minimize the effect of the singularities by discarding any representative streamline that gets too close to a singularity. This became a balancing act; too small a buffer zone results in larger errors in integrating $\nabla\sigma$, and too large a buffer zone means there are large gaps in the mesh that we use to compute anisotropy. This can be seen in Figure 4.14, where the areas of worst anisotropy (Figure 4.13) do not have sufficient coverage in both directions by the representative field lines.

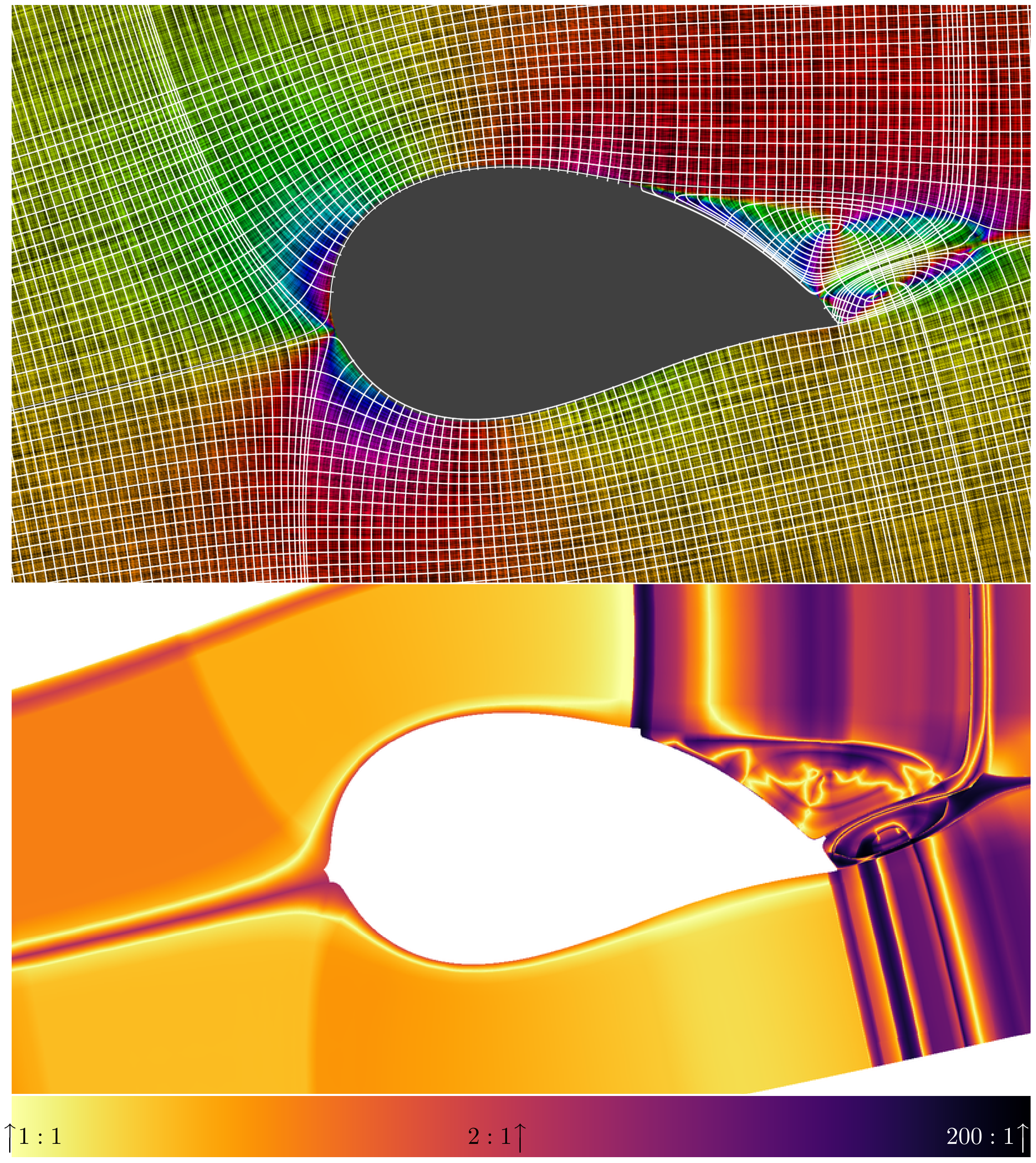


FIGURE 4.10. Airflow over an airfoil, with prominent trailing vortices. We start with a cross field that is aligned with the airflow around the wing (top); the background LIC is colored to match field direction to highlight singularities. Next, representative streamlines (top image, white) are traced, with some cells having more or less than four sides due to contained singularities. We optimize for a minimal-anisotropy configuration on the representatives, and we visualize the anisotropy by interpolating colors (bottom). The most extreme aspect ratio is 192:1.

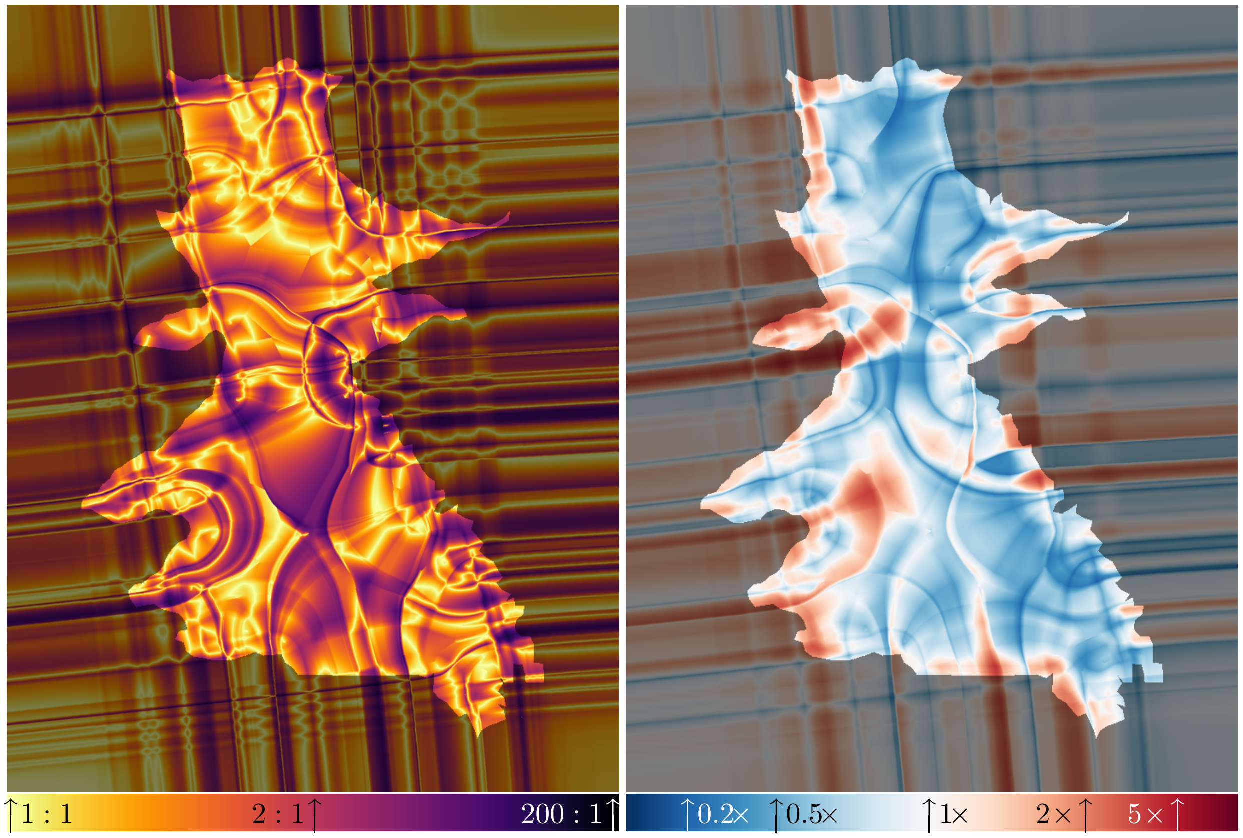


FIGURE 4.11. Anisotropy and size accuracy of the Sacramento Valley dataset from Section 3.4.3, with the area outside of the original boundary darkened.

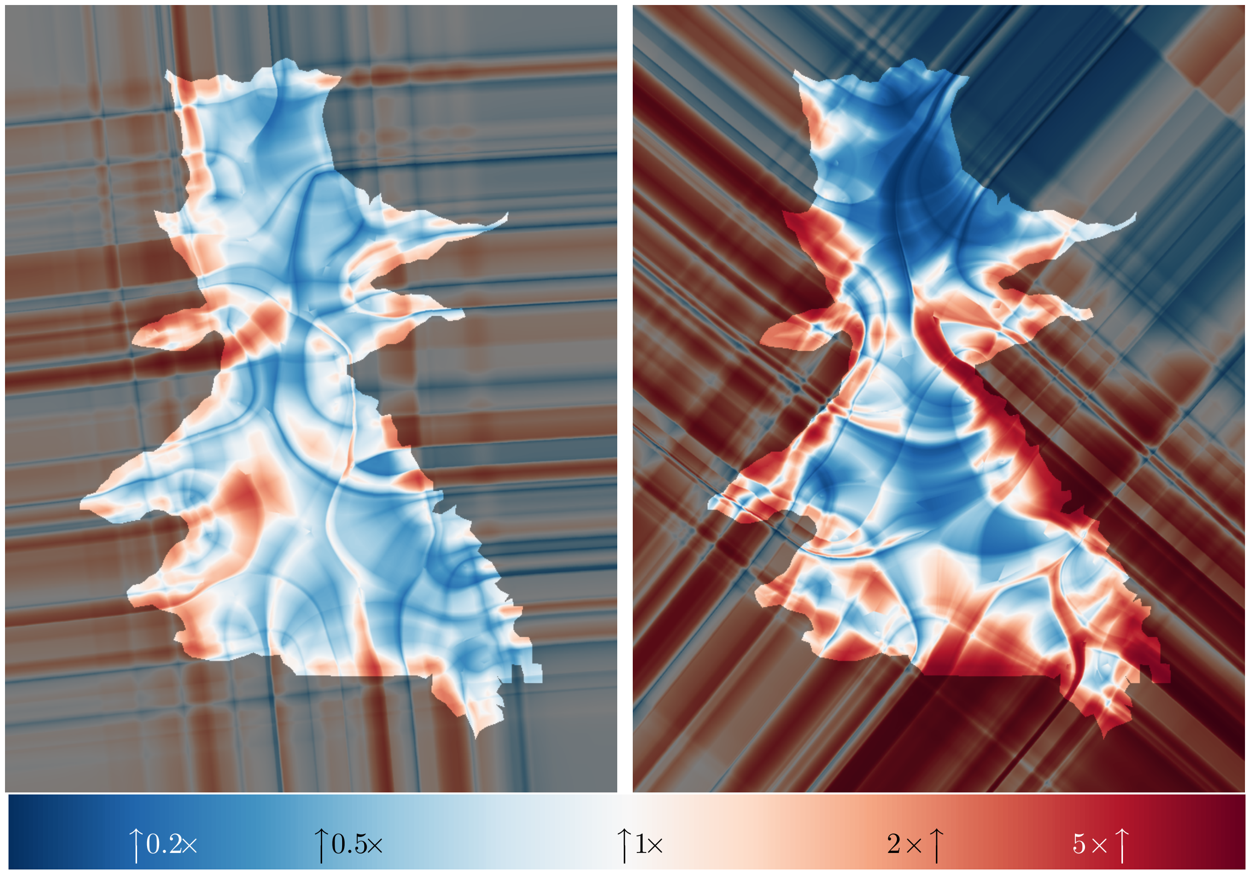


FIGURE 4.12. Accuracy of the produced size function of two different phases of the Sacramento Valley dataset. The relative accuracy across the regions differs due to the difference in phase changing the connectivity of the motorcycle graph and thus the paths that streamlines take through the region, demonstrating that the growth we achieve in our derived size functions (and later, meshes) is dependent on phase.

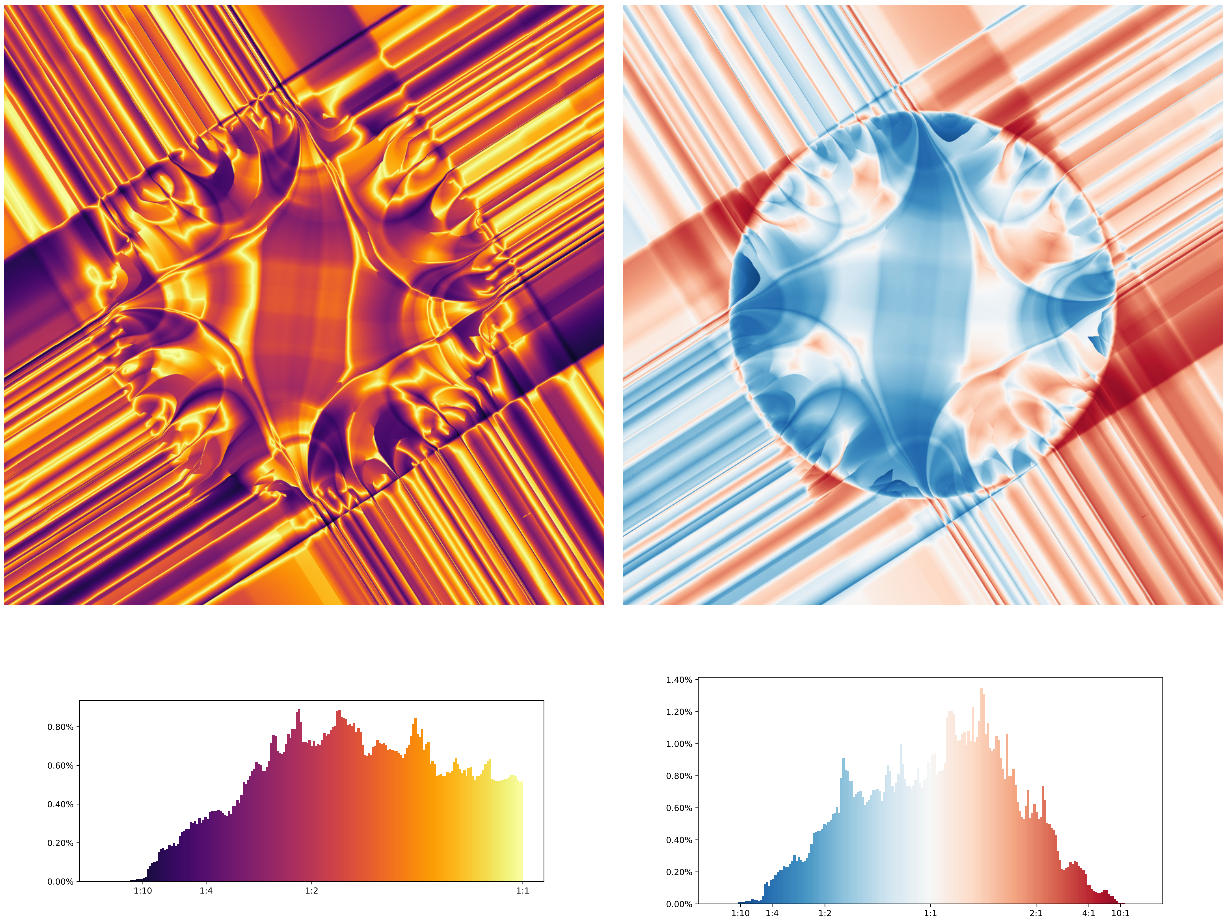


FIGURE 4.13. Anisotropy (left) and relative size (right) of the high-growth synthetic dataset from Section 3.4.2, both with the same phase. The worst anisotropy is 38.8:1, and the size ratios range from 11.5× too small to 12.2× too large. Overall, the cross field fell slightly short of the target 50× growth rate.

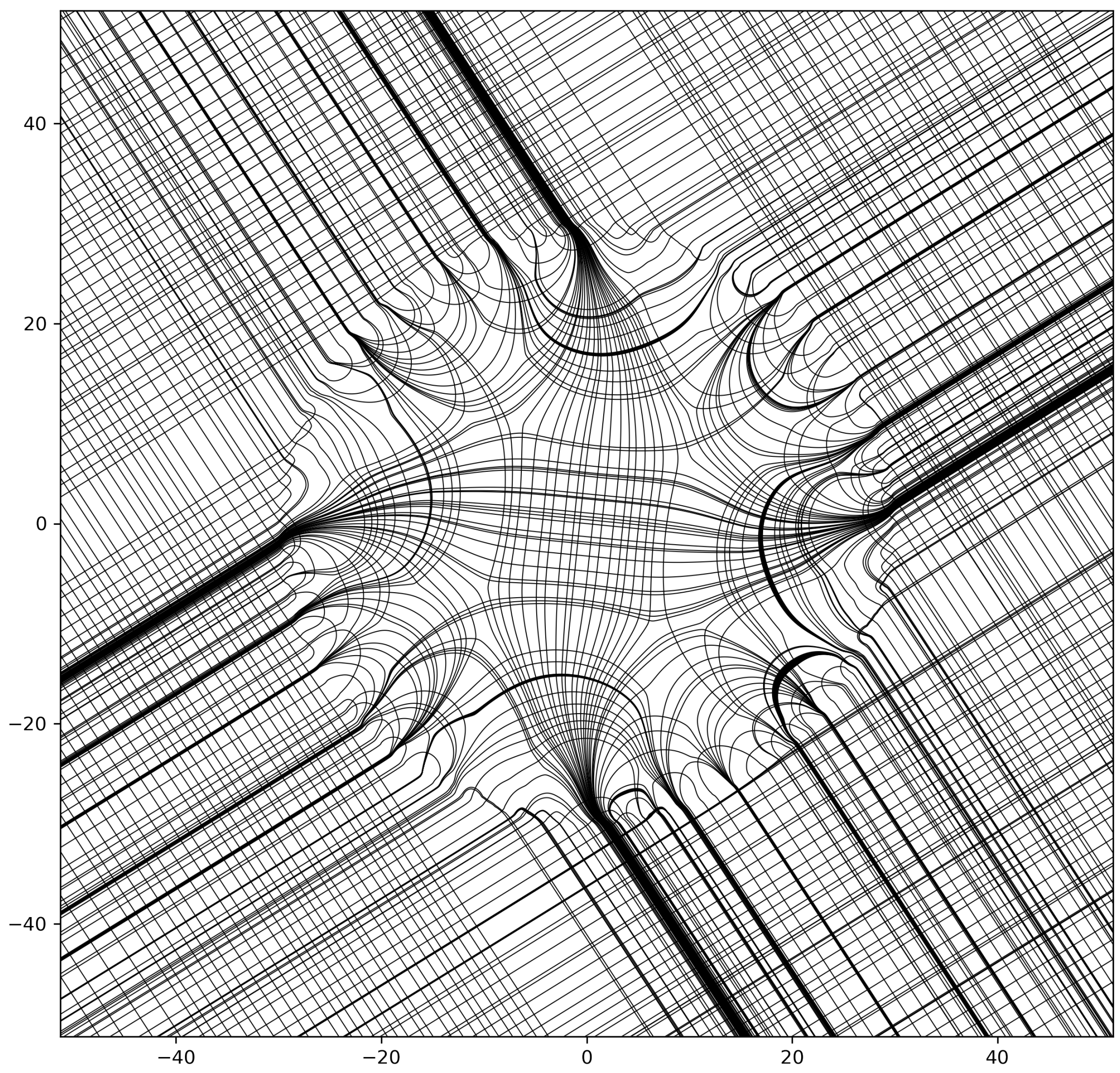


FIGURE 4.14. Representative streamlines used in the high-growth synthetic dataset. Regions of high compression, easily seen in Figure 4.13 as extreme anisotropy regions (nearly black) along the middle of each side of that image, are undersampled here because we avoid streamlines that come too close to singularities.

CHAPTER 5

Mesh Extraction from Cross Fields

Using the process in Chapter 3, we are able to create direction fields from size functions, and in this chapter we describe a process to extract a quad mesh from the direction field. Our approach is based on *Quantized Global Parameterization* [**CBK15**] by Campen, Bommes, and Kobbelt, which parameterizes the space into field-aligned tiles, which we can then quadrangulate as *Coons patches* [**Coo67**]. The resulting meshes can be used as the interior meshes for the approach in Chapter 2.

5.0.1. Quad Meshes and *4-RoSy* Direction Fields. There is a strong relationship between *4-RoSy* direction fields and quad meshes.

An analog of the direction field itself can be derived from an all-quad mesh. Within any quad, the two pairs of opposing edges form two axes, progress along which are the two parameters when performing bilinear interpolation over the quad. Progressing in either direction along both axes produces four directions, but these will not be properly orthogonal unless the quad has all right angles; the opposing directions will still be colinear, but the two pairs will be skewed in relation to each other. Sections 3.1 and 3.2 of [**BLH$^+$17**] describe a method where this skew can be eliminated to produce a *4-RoSy* direction field with the only discontinuities in the field at irregular vertices. These discontinuities are singularities in the direction field, with degree matching the valence of the source irregular vertex.

In this chapter, we explore this relationship in the other direction. Given a *4-RoSy* direction field, find the corresponding quad mesh. This is a discretization, with denser samplings providing more accurate representations, which we must balance against the size function we are targeting.

5.0.1.1. *Chords.* In a quad mesh, a *chord* is a series of quads connected end-to-end, such that each pair of neighboring quads share an edge and that only opposite edges in any given quad are used this way. In regions with no boundary, such as manifold surfaces, tracing out a chord inevitably lands back at the starting quad (facing the same direction), forming a loop, although the path may cross itself several times along the way. In bounded regions, chords may form loops or may begin and end at the region boundary.

Recall from Section 3.1.8 that two field lines are parallel if they travel in the same direction (in other words, both paths are isolines of u or both paths are isolines of v in some logical (u, v) coordinate space) and there are no singularities between them. In a direction field, we say that a *chord* is any region bounded by two parallel field lines. The two sets of edges forming the boundary of a chord of quads is very similar to the field lines bounding a chord in the field, as both represent a continuous ribbon.

A special case arises when the field line bounding a chord is a separatrix. The originating singularity for the separatrix lies on the boundary of the chord, and another of that singularity's separatrices will continue the chord boundary. In a case where both ends of a separatrix terminate at singularities, there will be multiple singularities on the same chord boundary. During the meshing process, these separatrices will be discretized, and those edges will become the boundaries of chords of quads in the resulting mesh.

Because a chord of quads traverses the entire region, either by ending at boundaries or forming closed loops, it can be removed topologically from a mesh: it can be *collapsed* by merging the two opposed non-shared edges in each quad, turning the chord of quads into a chain of edges connected end-to-end. This is a simple operation topologically and does not introduce any additional singularities. In fact, as this process causes vertices to be merged, it may actually remove singularities from the mesh, by merging two singular vertices together. The opposite operation is a *split*, where a chain of edges is expanded into a chord, or alternately where a chord is bisected into two chords by adding a new chain of edges down the center.

Starting with a complete mesh, repeatedly selecting and removing chords will eventually result in the entire mesh being collapsed down to a single vertex. In Section 5.1, we discuss the meshing technique described in [**CBK15**] which essentially reverses this process: we produce a mesh by starting with the motorcycle graph collapsed down to a single vertex and iteratively select edge paths through the mesh to expand into chords.

5.0.1.2. *Simple Tiles.* Both the separatrix graph (Section 3.1.8) and the motorcycle graph (Section 3.1.9 [**EGKT08**]) trace out separatrices in a direction field to form logically rectangular regions that are free from interior singularities. The motorcycle graph reduces the complexity of the resulting decomposition by allowing T-junctions in the graph and terminating separatrices when they intersect an existing separatrix, limiting the extent over which separatrices are active.

DEFINITION 5.0.1 (Simple Tile). *A simple tile is a region bounded on all sides by field lines, with no singularities in the interior of the region.*

In our case of using cross fields, simple tiles are logically rectangular; they are bounded by four field lines. Technically, a simple tile may be an annulus, where two opposite bounds of the region are along the same field line, but in practice there would be an enclosed singularity whose separatrices would cut the annulus into a rectangular region, so we will assume that all simple tiles are logically rectangles. In the separatrix graph, where the separatrices are split into segments where they collide, each tile is bounded by exactly four edges. The T-junctions introduced by the motorcycle graph allow an arbitrary number of edges along each face of a tile, although those edges are all segments of the four bounding field lines.

These tiles are referred to as *structured submeshes* in [**EGKT08**] because the it will typically be meshed as a structured $N \times M$ grid of quads. A *structured grid* is one in which the regular structure is exploited so that connectivity does not have to be explicitly represented or stored. We do not take advantage of the structured nature of the resulting quads and in fact violate this regularity in Section 5.1.4.

There is a natural connection between the simple tiles in a separatrix graph and chords; field-based meshing techniques typically use field lines as the basis for subdividing the region into quads to begin with, with each tile subdivided into an $M \times N$ grid of quads. Each tile encloses two orthogonal sets of parallel field lines that take the same path through the mesh, that will eventually become a sequence of quads forming a chord.

5.0.2. Arbitrarily-Dense Subdivisions. The separatrix graph provides a subdivision of the domain into simple tiles such that every tile is covered by either two distinct orthogonal chords or one chord that self-intersects orthogonally. Any chord may be further subdivided by selecting a field line interior to the chord and splitting along it, forming two new chords that together cover the same area as the original. Consider an ε-subdivision to be a repeated refinement such that every chord always has width less than some sufficiently-small ε. This subdivision produces small area elements that are each bordered by four field lines, with area less than ε^2. As ε approaches 0, the area elements shrink in size, and the curved segments of the bordering field lines are better approximated by straight lines. Except for the area elements immediately adjacent to a singularity, where colliding field lines are not perfectly orthogonal, as $\varepsilon \to 0$ these area elements eventually result in rectangular cells of infinitesimal area.

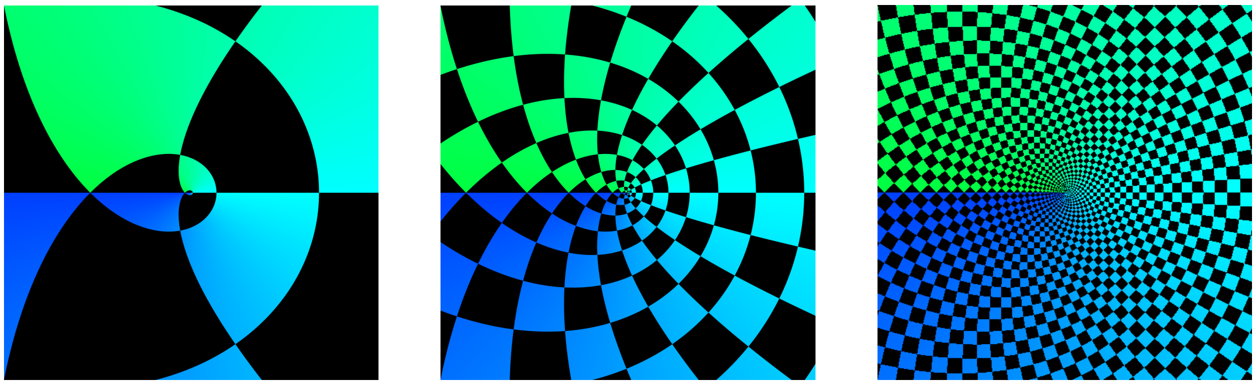


FIGURE 5.1. Three different subdivisions of the complex map $z \mapsto z^{1/4}$

A perfectly isotropic ε-subdivision would be one where all area elements (except those incident to a singularity) approach squares. While ε serves as an upper bound on cell size, the relative sizes of the area elements then yields a metric that varies across the region; each pair of neighboring area elements are both part of the same chord, so their relative sizes correctly reflect the divergence of field lines. An example of an isotropic subdivision is shown in Figure 5.1.

Any complex analytic function gives an example of the isotropic case, where the resulting size function is isotropic as described in Chapter 4. Thus, when the direction field φ is *harmonic*, defined as $\Delta\varphi = 0$, there is a compatible isotropic size function.

5.0.3. Anisotropy. While we have a rigorous definition of anisotropy in a cross field from Chapter 4, here we will look at anisotropy of quads as the result of iteratively subdividing chords. This serves as an introduction to the iterative process of *Quantized Global Parameterization* that will follow.

In general, ε-subdivisions of cross fields of planar regions do *not* produce square elements. This *anisotropy* is measurable as the aspect ratios of the area elements, as they deviate from being perfectly square. This anisotropy comes from two possible sources, the subdivision process and the direction field itself. Chapter 4 discussed the anisotropy inherent in the direction field; here we will focus on the subdivision process.

The choice of how to subdivide each chord in the separatrix graph to restrict its width to below ε determines the widths of the resulting chords, and the aspect ratio of each element is determined by the width of the two chords that overlap it. While Section 4.10 considered anisotropy in a continuous sense, here we deal with anisotropy in a very discrete sense: given a simple tile, what $N \times M$ subdivision gives the most isotropic set of quads?

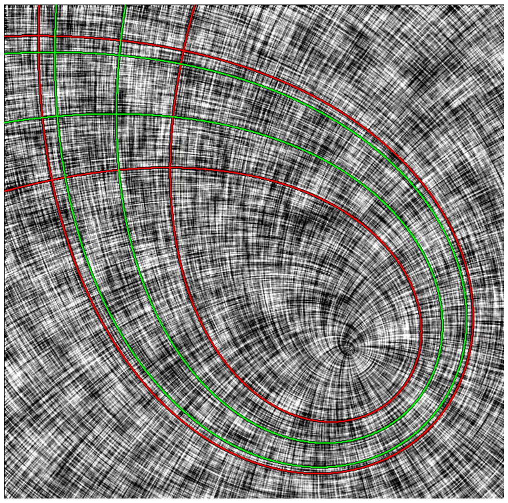


FIGURE 5.2. A potential subdivision of a self-intersecting chord

As we saw in Chapter 4, there are some regions where the direction field itself determines the aspect ratio over some simple tiles. Consider a tile where a chord intersects itself, as shown in Figure 5.2. The diagonal is comprised of all of the points where some field line within the chord self-intersects. For all of the infinitesimal quads produced along the diagonal, no choice of subdivision can affect one dimension of each quad without having an identical effect on the other dimension. Furthermore, if there is growth along any of these subdivided chords, meaning that the field lines enclosing the chord converge or diverge over the course of the loop, that enforces a fixed aspect ratio for each quad along the diagonal. As ε approaches zero, it becomes clear that for each point p on the diagonal, the aspect ratio at p is determined entirely by the growth along its corresponding field line, which is independent of any subdivision process; the aspect ratio of any non-infinitesimal quad along the diagonal is the integral of the aspect ratios for all of the points along the diagonal. The refinement process is effectively only choosing the intervals along the diagonal from which chords are extruded and cannot influence the aspect ratios further.

Most tiles are not covered by a single self-intersecting chord but instead by two independent chords, which will be subdivided independently. The subdivision process itself is not a discretization of a 2D region into smaller 2D regions but a discretization of 1D regions (any arbitrary width across each chord) that are extruded along the cords, the pairwise product of which yields the subdivision of each tile in the separatrix graph, covering the entire domain. The resulting aspect ratios of the quads within each subdivided tile are simply a function of how the two intersecting chords are individually subdivided.

5.0.4. Increasingly-Sparse Subdivisions. As we will see in Section 5.1.7, one of the challenges in meshing is that larger elements are inherently harder to produce, as smaller elements allow for more degrees of freedom. While an infinitesimal subdivision is a useful theoretical construct, it is not the end goal; in general, we want as few (and thus, as large) elements as possible meeting the quality constraints we have.

Not only is an ε-subdivision too fine, but the separatrix graph from which it was generated is also very likely to be too fine. If a very aggressive change in growth is requested, this will induce many singularities and in turn may even cause the motorcycle graph to be too fine a subdivision; this is a failure case that we will not consider here. This highlights an inherent conflict when trying to have large elements: a more aggressive growth rate requires more singularities, but a higher singularity count will increase the number of separatrices, decreasing the size of the simple tiles in the motorcycle graph, limiting the size of the largest elements.

While the bulk of this chapter consists of a process for extracting a quad mesh from a motorcycle graph, the important intuition is that the motorcycle graph intentionally removes parts of separatrices far from their originating singularity.

5.0.5. Separatrix Graph Techniques. The separatrix graph is a natural starting point for generating all-quad meshes. Each simple tile within the separatrix graph can be subdivided along both axes to form a grid of simple tiles, with the number of subdivisions of each chord allowing some control over final element size. In order to prevent T-junctions in the final mesh, neighboring tiles within the separatrix graph must have the same number of subdivisions along the separatrix segment separating them, setting up an integer system of equations as seen in [**FAR15**]. This integer solve is necessary because any given separatrix segment might have an ideal number of subdivisions based on the local metric given by the input sizing function, which may not perfectly match the number of subdivisions on corresponding segments elsewhere, none of which are likely to be integer values. Using a global solve is one way to find a set of integer subdivision counts that best respects each segment's local desired subdivision count while maintaining global consistency.

[**FAR15**] also employ templated subdivisions, some of which add new singularities in order to allow regions to have different numbers of subdivisions along opposite sides. We will revisit this concept in Section 5.1.4.

5.0.5.1. *The Motorcycle Graph.* [**FAR15**] enumerates several problems arising from using the separatrix graph, noting that there are regions that can be arbitrarily small (along one or both axes), that separatrices are not guaranteed to end at a boundary, and, as mentioned earlier, that the locally-optimal number of subdivisions across a chord may differ significantly along its length. They suggest using the motorcycle graph as a means of reducing region count. While they do not directly mention limit cycles, they give consideration to the *nautilus*, a specific configuration which arises in the motorcycle graph in the presence of a limit cycle, which we address in Section 5.1.4.

Instead of performing one global solve to determine the optimal number of subdivisions along each separatrix segment, [**CBK15**] uses an iterative approach that progressively improves the mesh, which we will discuss in depth shortly. The most striking benefit is that the mesh implied by the current number of subdivisions across each edge is always valid. Degenerate scenarios, such as non-simple tiles with more or less than 4 bounding field lines (which must therefore contain singularities that are not in the separatrix graph), are collapsed to a point, ensuring every simple tile remains in a meshable state.

While not directly mentioned in [**FAR15**] or [**CBK15**], the T-junctions along region boundaries in the motorcycle graph can alleviate some of the issue with chains of regions disagreeing about subdivision count. Chords in the separatrix graph are bounded by parallel separatrices, so each perpendicular separatrix they cross represents an opportunity to terminate the progress of the separatrices and chords. With shorter chords, there is less opportunity for divergence in the desired subdivision count.

5.1. QGP: Quantized Global Parameterization

The Motorcycle Graph construction was described in Sec 3.1.9. The following process is based on the approach of *Quantized Global Parameterization* [**CBK15**] for computing a meshable subdivision of the motorcycle graph into simple tiles. To complete our quad-meshing algorithm, we use *Coons patches* [**Coo67**] to mesh the simple tiles according to the computed subdivisions, and extend the QGP approach with additional singularity placement similar to [**FAR15**].

5.1.1. Initialization. The goal is to assign a subdivision count s_i for every edge e_i in the Motorcycle Graph, where these edges are the separatrix segments bounding the regions.

In a separatrix graph, requiring every simple tile be subdivided into a simple grid results in a constraint that $s_0 = s_2$ and $s_1 = s_3$, for some region r bounded by edges e_0, e_1, e_2, and e_3. In a motorcycle graph, each side (*e.g.* r_0) of the simple tile may be composed of several edges, so the constraint becomes:

$$\sum_{e_i \in r_0} s_i = \sum_{e_j \in r_2} s_j \tag{5.1}$$

with a similar constraint for the remaining sides r_1 and r_3. This ensures that each region may be subdivided with quads without adding any further singularities to the mesh. The condition $s_i \geq 0$ prevents degenerate topologies; $s_i = 0$ means that edge is collapsed, which is valid, but a negative number of subdivisions is not.

Every edge e_i has an ideal real-valued subdivision count $\hat{s}_i = \int_{e_i} \frac{dx}{\sigma}$. This would be $\frac{|e_i|}{\sigma}$ if the size function was constant over the edge. [CBK15] define the following energy function:

$$\sum_{e_i} \left(\frac{s_i}{\hat{s}_i} - 1 \right)^2$$

The goal then becomes to minimize the difference between the actual and ideal subdivision count for each edge. Here, σ is the isotropic size function that was used to generate the cross field, but an optimized σ such as the minimum-anisotropy fields from Chapter 4 could be used as well, although optimizing toward an isotropic size function naturally reduces anisotropy.

We start with the initial condition $s_i = 0 \ \forall i$. This configuration represents all vertices, singularities or otherwise, collapsed down to a single vertex.

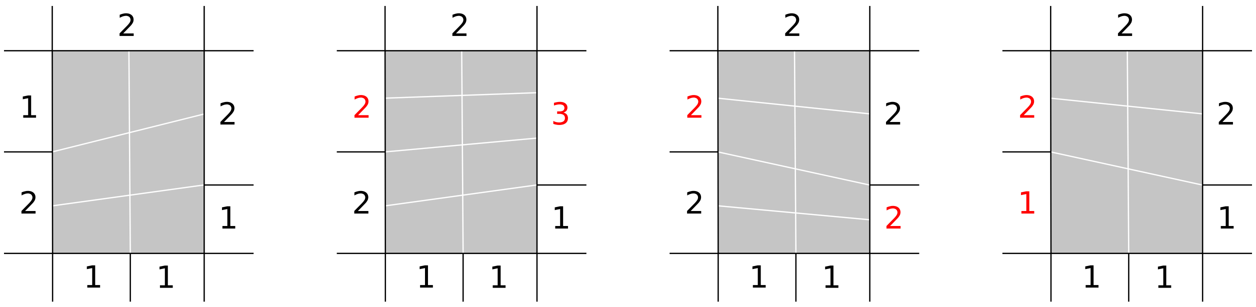


FIGURE 5.3. Three potential ways to preserve the region constraint when increasing the number of subdivisions along an edge.

5.1.2. Iteration. In order to improve the mesh (and lower the energy metric), we must modify the number of subdivisions s_i along some edge e_i to be closer to its goal $\hat{s}_i$. Ensuring that $s_i \geq 0$ is straightforward, but preserving Equation 5.1 is more involved. As seen in Figure 5.3, increasing some s_i requires either *increasing* some s_j along the opposite side of the region, or *decreasing* some s_k along the same side of the region. That change to s_j or s_k will, in turn, require further changes, until either reaching the boundary of the motorcycle graph or returning to the origin edge e_i. This turns into a search process, finding an *augmenting path*, either a loop or a perimeter-to-perimeter path, that results in the largest drop to the energy function, starting with the selected edge e_i. All edges are tested, and the edge with the most-beneficial augmenting path is selected and the changes to all subdivision counts s_i along the path are updated accordingly. This continues until no edge can produce a beneficial augmenting path.

5.1.3. Region Subdivision. Once the iteration has completed, we are left with an integer for every edge in the motorcycle graph, representing the number of chords that will cross that edge. Each simple tile of the motorcycle graph needs to be subdivided accordingly, to generate the final mesh. However, the perimeters of these simple tiles are arbitrary field lines; while logically rectangular, they can have significant curvature, as in Figure 5.4 where the tile is bent such that opposite sides are bounded by the same field line! This does not require special handling; it just means that one edge can appear on two sides of the same simple tile.

In order to facilitate the process of meshing each rectangular region, we treat each region as a *Coons Patch*. This defines a (u, v) parameterization over the patch from the boundary curves, with both $u, v \in [0, 1]$, and the curves themselves having fixed u or v values of 0 or 1 [**Coo67**]. This allows us to take points and lines in the u, v space where each region is an axis-aligned rectangle, and map them to points and curves in the x, y space of what will be the mesh.

In particular, we use bilinear Coons patches, so the blending functions F_0 and F_1 used in [**Coo67**] are simply linear interpolation. We start with the four boundary curves C_i all with counterclockwise winding, such that $C_i(u)$ is the point at distance $u|C_i|$ along curve C_i, and C_0 is the bottom curve with coordinates $(u, 0)$. The (x,y) coordinates for any point on the patch are derived from three different interpolants: the point linearly interpolated between the points with progress u along the top and bottom curves *from the left*, added to the point linearly interpolated between the points with progress v along the left and right curves *from the bottom*, *minus* the point bilinearly interpolated between the four corners where the curves meet.

$$\begin{aligned}
P(u, v) = {} & lerp(v, C_0(u), C_2(1-u)) \\
& + lerp(u, C_3(1-v), C_1(v)) \\
& - lerp(v, lerp(u, C_0(0), C_0(1)), \\
& \qquad lerp(u, C_2(1), C_2(0)))
\end{aligned} \tag{5.2}$$

It is not as simple as tracing axis-aligned paths in (u, v) to separate the region into quads because the subdivisions along each edge are not necessarily aligned with each other; for example, none of the lines separating left-to-right chords in Figure 5.3 are perfectly horizontal. So, for a horizontal path from $(0, v_0)$

to $(1, v_1)$, the curve traces the path $f(u) = (u, lerp(u, v_0, v_1))$. These are precisely the paths seen in logical (u, v) space in Figure 5.3. As a result, the edges of our mesh are *not* field lines and do not necessarily meet at right angles.

5.1.4. Additional Singularities and the Nautilus. Inspired by the approach of [**FAR15**], we can relax the restriction that opposite faces of each rectangular region must have the same number of chords. This requires adding extra singularities to each region; adding two singularities, one of valence-3 and one of valence-5, preserves the neutral holonomy of the tile while allowing us to redirect a chord in a direction of our choice. This must be done with care, so we add a penalty to each chord redirected in this way, to bias the meshing to prefer chords that travel straight through each rectangular region, and to encourage later augmenting paths to eliminate these redirections wherever possible.

The nautilus is a case that poses a particular problem for this style of meshing and is the structure seen in the motorcycle graph when the field contains a limit cycle (discussed in Section 4.12); it is shown in Figure 5.4. The problem arises due to the chord restriction on the tiles, quantified by Equation 5.1, where the nautilus forces the same edge to appear on two sides of the tile: $r_0 = e_0$ and $r_2 = e_0, e_1$. This results in the relationship $s_0 = s_0 + s_1$, implying $s_1 = 0$, resulting in all chords that impact edge e_1 throughout the rest of the mesh getting collapsed, which can cause a significant drop in element quality.

We instead relax Equation 5.1, allowing unequal subdivisions along opposite sides of the tile, as long as the discrepancies between both pairs of sides have the same magnitude. The relative signs of the differences determine the orientation of the placed singularity pairs; chords from the side of the vertical borders with more subdivisions need to get redirected to the corresponding "denser" side of the horizontal borders (left and top of Figure 5.4, respectively).

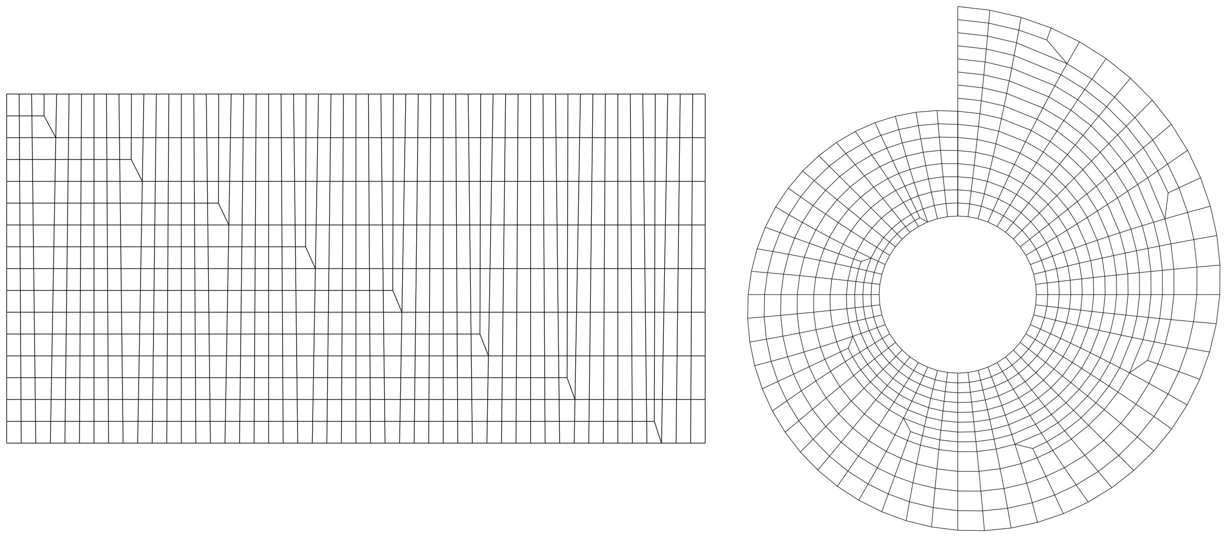


FIGURE 5.4. Subdivisions of the nautilus in logical (u,v) coordinates (left) and physical (x,y) coordinates (right). The extra singularities allow chords to be redirected, which would otherwise require the collapse of the upper ridge and all incident chords.

The optimal placement of these extra singularities within the region remains an open problem, although the combinatorial space grows quickly enough where we advise against a brute-force approach. As is apparent in Figure 5.4, each valence-3 singularity stops a sequence of edges from reaching the opposite edge. In this sense, each singularity has integer coordinates (i, j), and the coordinates of the singularities in Figure 5.4 were chosen to maximize the stride of the i and j sequences, spacing out the singularities within the tile. The subdivisions are equally spaced along each tile edge, with singularity-free rows and columns simply directly connected across the tile, and the remaining rows and columns bisecting their nearest neighbors and terminating at singularities.

5.1.5. Important Behavior.

5.1.5.1. *Chord Drift.* The invariant across simple tiles in the motorcycle graph is simply that the number of chords entering one edge matches the number of chords leaving the opposite edge. The positioning and width of these chords is not enforced; the resulting mappings do not follow field lines through the interior of the region when the chords are connected, which results in skew in the resulting quads as seen in Figure 5.3. In general, the denser the subdivision, the more the chords will naturally align, resulting in less skew; this can be seen in the two leftmost images in Figure 5.3, where the increase in perimeter subdivisions allows the left-right subdivisions to be closer to horizontal, meaning the resulting edges will be closer to following field lines.

5.1.5.2. *Singularity Merging.* As mentioned in Section 5.0.4, a large number of singularities simultaneously allows for a large change in size while hindering the largest achievable size for any individual quads. In some circumstances, there may be singularities generated to induce growth that cannot be achieved due to the density of the singularities nearby; looking at the direction field as an infinitely fine grid, the addition of the singularities allows for the relative growth we are looking for, but when scaled up to match the target size function, the mesh cannot admit quads of the growth induced by the singularities.

In these circumstances, it is helpful to merge singularities together. The restriction that [**CBK15**] adds to prevent this behavior is due to a difference in context; their singularities are a result of a parameterization process that reflects the surface topology and must be obeyed, while our singularities are all opportunities for growth that may not be achievable.

When two singularities are merged, their *indices*, the difference from the field degree, are added together. So, merging a valence-3 singularity with a valence-5 singularity, indices -1 and $+1$, respectively, results in a non-singular point of index 0 and valence 4. Merging two valence-3 singularities results in a valence-2 singularity, which induces quads with an internal angle of π. This last degeneracy generally results in poor performance of the simulations run on the mesh, so it is best to avoid merging singularities into valences of two or less.

Even in cases where two singularities merge to form a regular non-singular vertex, the singularity is not totally removed as the separatrices remain. While the separatrices represent a slice through the mesh that is not required, resulting in smaller quads than would be otherwise possible, merging the singularities together also merges the separatrices, collapsing the thin chord that would have separated the separatrices had the singularities not been merged. The remaining separatrices may also be further merged with neighbors, further reducing their cost.

5.1.6. Degenerate Tile Collapse. Any simple tile with no splits along any of its incident edges represents a configuration where all vertices in the region get merged into a single logical vertex. This is one mechanism by which singularities are merged, but it also applies to cases where, for example, an excessively large step size induced a turn along a separatrix in the motorcycle graph, resulting in an interior cell in the graph with more (or less) than four incident edges. These degenerate cases are no longer simple tiles, as topologically they must contain a singularity even if there is no singularity in the field; the error in tracing the boundary of the region means it has induced some amount of turn in the perimeter and has created a virtual singularity in its interior. Even though the region itself is degenerate, it can be collapsed into a single vertex, which allows a valid mesh to be constructed. The index of the resulting vertex will be the sum of the indices of all points in the region, including the corners of the tile and the virtual singularity; while this most typically happens in areas of abnormally high curvature near clusters of singularities, where the relative density of singularities makes for small simple tiles that can be collapsed without significantly reducing overall mesh quality, if the simple tile is large, its collapse can have a drastic negative impact on the resulting mesh.

It is preferable to ensure these degenerate tiles do not get produced in the first place, typically via dense sampling rates for streamlines, and oversampling, where the streamlines are densely sampled but only a fraction of the samples are retained. There are performance penalties associated with both of these methods, and they do not guarantee that any given motorcycle graph will be free of degenerate tiles. The motorcycle graphs in Figure 5.5 have these degenerate tiles, and were successfully meshed as seen in Figure 5.6.

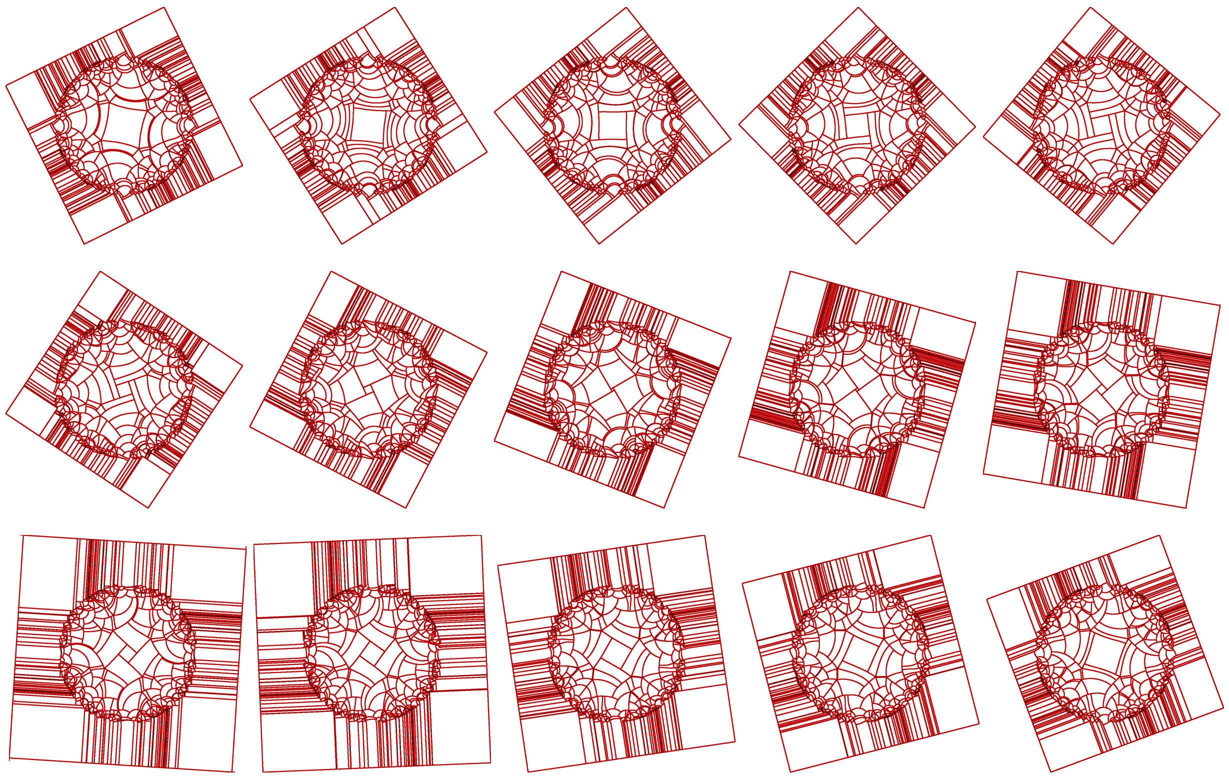


FIGURE 5.5. Series of motorcycle graphs for different phases of the same underlying field. While simple tiles may be collapsed to allow for the generation of quads larger than any single simple tile, larger simple tiles are less likely to need to be collapsed and are therefore preferable to configurations that cover the same area with many smaller simple tiles.

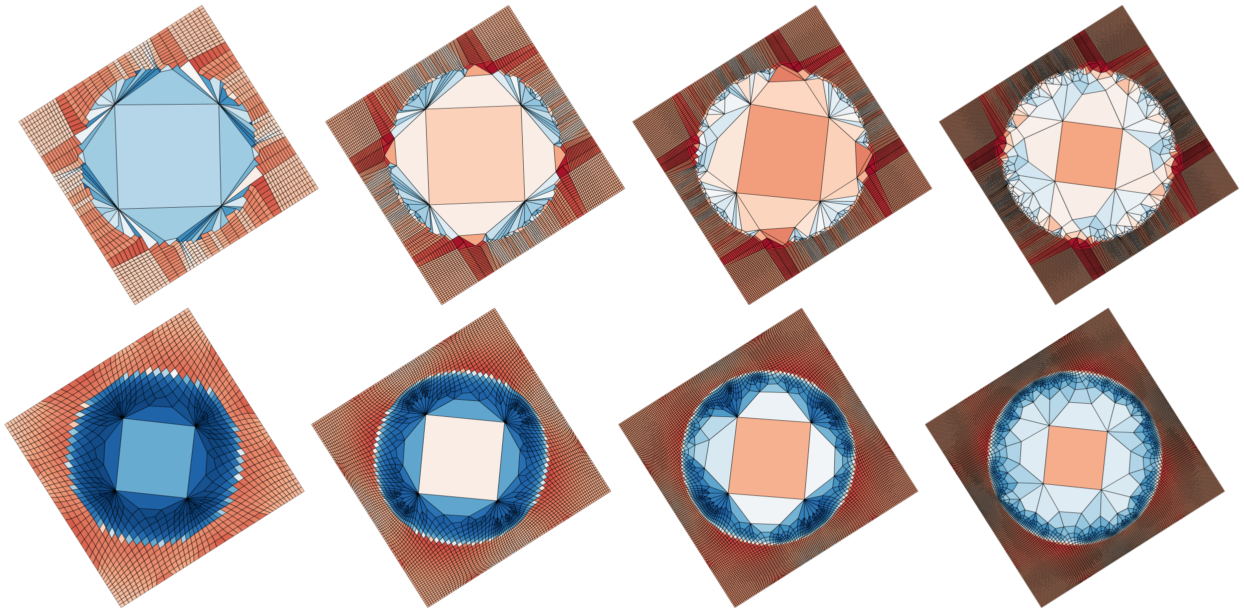


FIGURE 5.6. The same motorcycle graph, meshed at $1\times$, $2\times$, $3\times$, and $4\times$ densities, followed by smoothed variants of the same meshes. Quad color indicates size accuracy $\sqrt{\text{area}}/s$ with respect to the target size function s.

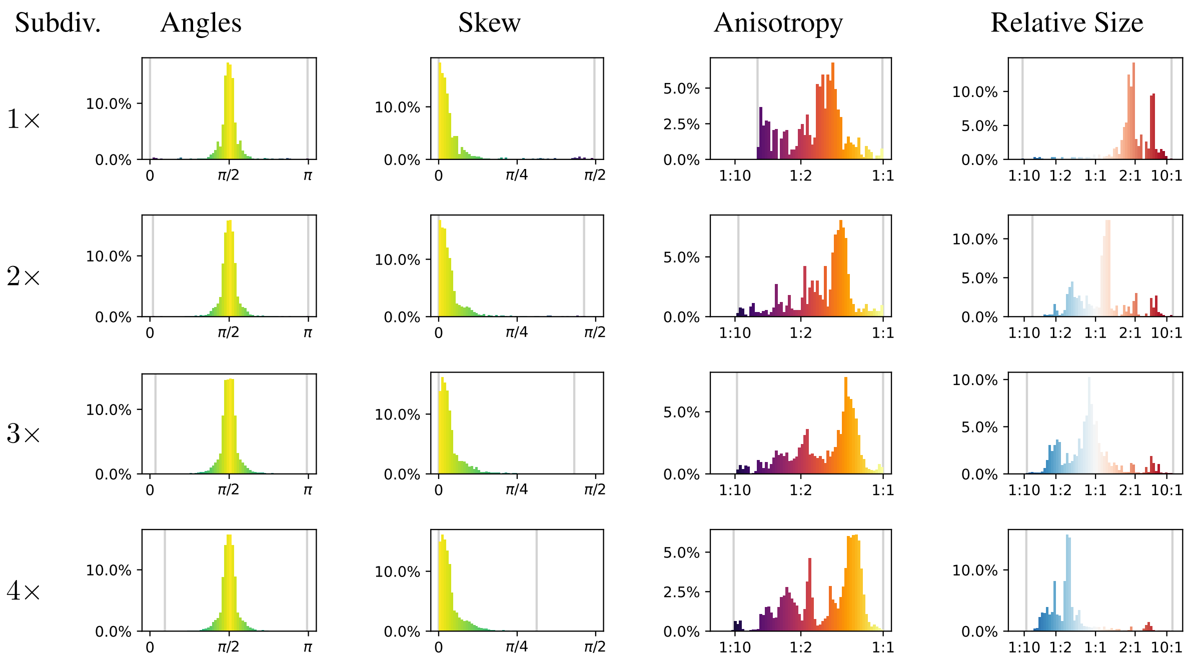


FIGURE 5.7. Quality measures for the different subdivision levels of the unsmoothed meshes from Figure 5.6, with gray vertical lines highlighting extremal values. With increasing subdivision levels, interior angles, skew, and relative size all drastically improve, while anisotropy gets worse. Given that the meshing process is minimizing size error, the anisotropy of increasing subdivision levels will converge to a distribution *worse* than the continuous anisotropy calculated in Figure 4.13, which minimized anisotropy. Merged cells have resulted in quads with almost 180° angles as well as extreme-valence singularities, and both effects are diminished with increasing subdivision levels.

5.1.7. Scale Factor Comparison. The quantization of the region into a mesh is easier to accomplish with smaller elements as illustrated in Figure 5.6. The same motorcycle graph was meshed at four different scale factors, and the resulting quads were colored based on their size relative to the target size. The high-density outer region is relatively easy to achieve, as demonstrated by all four scale factors having nearly identical normalized element sizes. However, results vary in the interior region, with the densest mesh exhibiting the highest growth rates. Due to the large number of singularities required to achieve this growth, the less-refined examples have to more aggressively collapse chords, resulting in singularities with extreme valence.

5.1.8. Results.

5.1.8.1. *Synthetic Growth Case.* We return to the high-growth synthetic example from Section 3.4.2.

Figure 5.5 shows a series of motorcycle graphs for varying phases of the field. The structure of the motorcycle graph is dependent on the phase of the cross field, as how the separatrices align with each other determines the shapes of the simple tiles. The larger the tiles, the larger the quads that can be made without collapsing nearby chords. The large central simple tile (as in the upper-right image) is strongly affected by phase, with some phases causing it to shrink like a camera aperture until it is effectively collapsed and the area is covered by four simple tiles instead. This has a significant impact on achievable growth rate. This shows that searching for the best phase for a given mesh may be a valid, if computationally-expensive, optimization strategy.

In Section 4.13, we computed the size function with minimal anisotropy for this dataset and found that the compatible size function for the cross field did not admit the target growth rate, as seen in Figure 4.13. While the original size function is used for the meshing process, we are unable to achieve the $50\times$ ratio between interior and exterior elements. Looking at the meshes in Figure 5.6, the outer perimeter elements are all too large, with the innermost element ranging from too small to too large as the subdivision level increases. With further subdivision, we expect that the size accuracy of the mesh elements to converge toward the continuous size accuracy of Figure 4.13 in Section 4.13.

Figure 5.6 looks at different densities of the same phase, where the motorcycle graph is split according to the size function s/k for $1 \leq k \leq 4$. The aggressive target growth rate ends up forcing many simple tiles to be collapsed, with large-degree singularities present in Figure 5.6, and extreme-degree singularities shown in Figure 5.8.

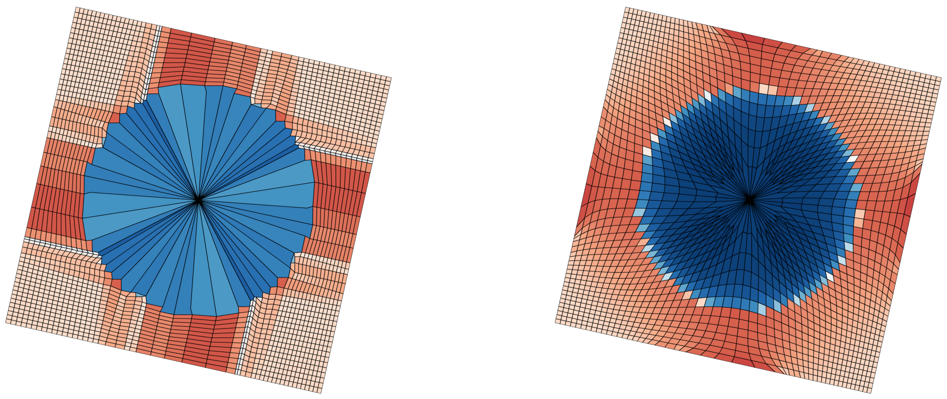
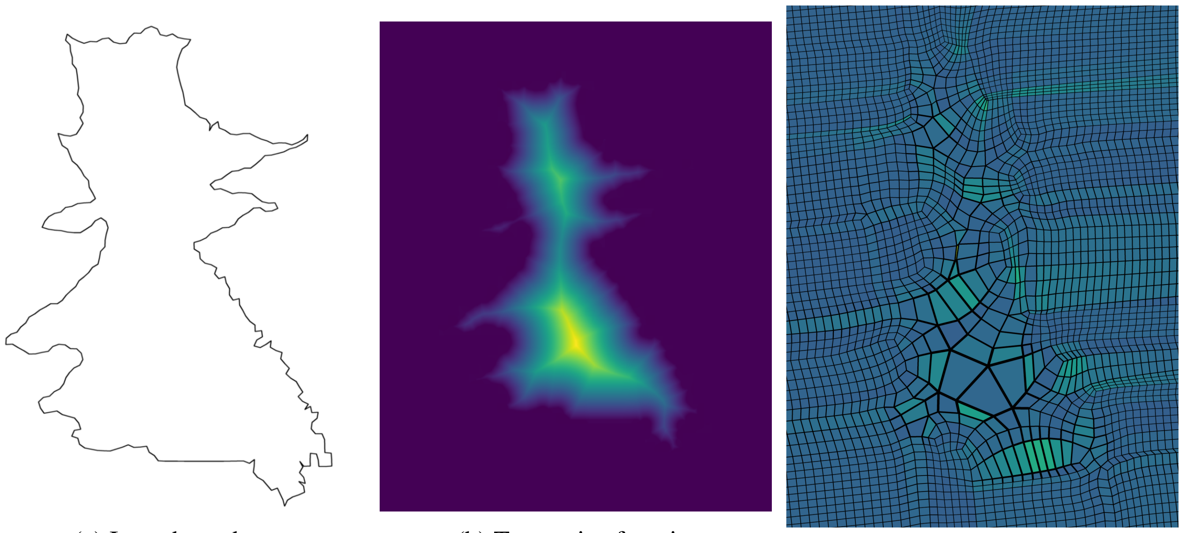
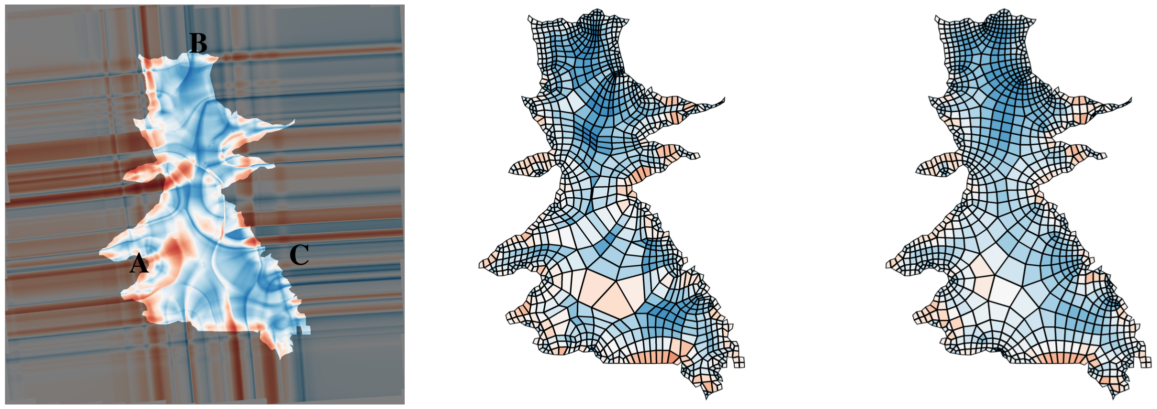


FIGURE 5.8. While the meshes in Figure 5.6 successfully collapse simple tiles to form a quad that is larger than any of the tiles individually (see Figure 5.5), the collapse of tiles can lead to very high-degree singularities which would be unsuitable for simulation.



(a) Input boundary (b) Target size function (c) Extracted mesh



(d) Expected size accuracy (e) Stitched result (f) Result with Laplacian Smoothing

FIGURE 5.9. The Sacramento Valley dataset.

5.1.8.2. *Sacramento Valley.* Figure 5.9 showcases our full pipeline using the Sacramento Valley dataset. Starting with a polyline boundary that is often quite jagged, we derive a size function that increases into the region based on the distance to the boundary and the edge length at the boundary. We use that size function to produce a cross field which extends beyond the original boundary, as described in Chapter 3, which is then meshed using the process outlined in this chapter. Quads from this mesh are carved away until the resulting mesh fits within the original boundary, with a channel being formed between the boundary and the interior mesh. This channel is meshed using the technique from Chapter 2. We also consider a smoothed version of the resulting mesh.

The final mesh does not exactly achieve the targeted growth rate, especially in the upper region of the image. However, the open area in the lower portion of the mesh does achieve the desired growth, with the largest elements actually exceeding the expected size. Once smoothed, the largest elements shrink down to below the target size, but the quality of the quads is significantly improved.

The stitching process, while adding many singularities along the channel, is able to isolate the interior regions from the chaotic boundaries; this is most evident along the lower portion of the right side of the mesh, where the boundary is particularly jagged and transitions nicely to the interior mesh. This effect is rather drastic in the smoothed mesh.

The size estimate in Figure 5.9d was for a minimally anisotropic configuration and is a reasonable match for the size of the quads actually achieved in Figure 5.9e. A region where we expected quads to exceeded the target size, marked **A**, were instead slightly smaller than the target size, potentially due to the isotropy in the region as meshed not matching the minimally isotropic continuous solve. Region **B** correctly estimates quads being smaller than the target size function, and Region **C** correctly estimates slightly larger quads along the perimeter (although most of these will be replaced as the mesh is stitched to the perimeter) followed by an area of undersized quads toward the interior before growing again.

CHAPTER 6

Concluding Remarks

Our goal in this thesis was to construct all-quadrilateral, high-quality, low-singularity meshes that meet the constraints imposed by groundwater flow simulations, in particular user size control and exact conformance to highly irregular boundaries. We split this problem into phases, with each phase tackling a different constraint, and demonstrate promising results with the resulting pipeline. In exploring the problem space, we have encountered several interesting problems, and we have presented our solutions to many of them. We would like to end with some of the problems that we have not fully solved.

6.1. Combinatorial Explosions in Channel Coverage

There are several different steps in the stitching process of Chapter 2 that needed to be artificially bounded due to the combinatorial nature of the problems. In particular, the generation of patches to cover the channel can rapidly grow out of control if the number of patches is not carefully constrained. We are able to assume that the size of the quad edges in the interior mesh is close to the size of the perimeter edges in the boundary because the interior meshes we generated were either fixed-size grids or meshes explicitly built to match a size function. This allows us to limit the number of subdivisions across the chord naïvely, simply considering candidate vertices within a fixed radius. A more general solution would likely be a more incremental approach, starting with the most orthogonal edges across the channel and only searching for additional candidates as the threshold for a completed circuit around the channel drops. This would be necessary to support channels that varied in width, as opposed to the fixed-width channels we employ now.

A lazier approach to patch generation would likely dramatically improve performance. Currently, patches along the channel are generated and ranked in batches, with relaxed constraints for each subsequent batch as the previous batches were unable to successfully cover the channel. When a batch of patches fails to cover the channel, it is typically only in a few regions where the region boundary is poor and forces the generation of patches with small angles. Relaxing the quality constraints to generate a new batch of patches over the entire channel is likely to generate many unnecessary patches, as most of the channel is already covered by high-quality patches. Integrating the patch generation along with the search process, generating and ranking patches on-demand, could allow the lower-quality patches to be generated only in areas of low coverage where they are needed, instead of over the entire channel.

6.2. Singularity Placement

We based our work in Chapter 3 on *Globally Optimal Direction Fields* [**KCPS13**] due to their automatic creation and placement of singularities based on curvature, in our case the curvature induced by the changing metric. While the optimization performed by GODF is able to exactly match harmonic angle functions (such as the conformal map used in Figure 4.7), most size functions will not be harmonic, including the size functions we are using where distance grows from a region boundary. In these non-harmonic cases, there will be disagreement between the target size function and the growth in the produced cross fields.

The GODF optimization process is looking for a small quantity of integer-valence singularities to produce a smooth cross field. Within these constraints, it produces a cross field that is in some sense optimally close to having the growth prescribed in the target size function. Exactly how the result differs from the target size function is unconstrained, although in general the resulting cross fields tend to have less growth than requested. So, it may be helpful to perform a second optimization step, tweaking the input parameters to see if a solution to a similar problem may be more suitable than our initial result, such as by overestimating the requested growth to find results that more closely match it (typically at the cost of a less smooth mesh with more singularities).

We have already been performing this second layer of optimization, both directly, by testing against multiple eased variants of the desired growth pattern and different growth rates, and indirectly, by changing the background mesh, which changes the search space of the optimization process. Unfortunately, how this set of inputs influences the accuracy of the produced cross fields is unclear, as the results of changing any parameter tend to be erratic as seen in Figure 3.25. The intuition we have built up is that exponential growth on an adaptive background mesh is a good first bet, offering consistent results, although one of the other configurations will often better match the target growth.

While we have achieved good results, we have not been able to establish a link between these parameters and the growth and anisotropy of the resulting cross fields. This is clearly a problem that needs to be explored further.

The confusion around this problem first led us to look for a better way to quantify the growth expressed in the resulting cross fields. In exploring this, we encountered the seemingly-impossible size functions with non-conservative gradients, due to the unexpected anisotropy in the results, which was the inspiration for Chapter 4.

6.3. Sampling Accuracy of Streamlines

Chapters 4 and 5 both depend heavily on the accuracy of the streamlines, whether it is the paths themselves or values sampled and integrated along them. In discussing the field φ as being represented by some angle function α, it is tempting to follow the same abstraction in the implementation by having the field be some function of only position, with the software layers built on top oblivious to the representation below.

By abstracting away the triangles of the background mesh, critical information is lost. Instead of the field being some function α that gets sampled at regular intervals to form polyline streamlines, the field should instead produce line segments such that each segment is entirely contained by a single background mesh triangle. The most important components of the streamline, specifically the integral of α to define the path and the integral of $\nabla\alpha$ to determine the growth, are entirely defined by the position and heading of the streamline on entering the triangle and the position of and angles stored at the triangle vertices.

If the angle on the background mesh is a scalar angle, as we suggest in Section 4.14.1, $\nabla\alpha$ is constant over the triangle, making integration much easier. Even if the angles are represented as complex values, as we use, the constant $\nabla\alpha$ is still a very close approximation for non-singular triangles, and knowing the bounds of the triangle can further help with integration by helping to eliminate the bias that comes with integrating over these samples, also discussed in Section 4.14.1.

Furthermore, with a scalar-valued α, it is possible to represent the field over each non-singular triangle as two potential functions whose gradients are orthogonal, where all streamlines over the triangle are isolines in one of the two potentials. This representation would practically eliminate the issue of high curvature causing streamlines with too large a sample interval to incorrectly snap to the orthogonal direction discussed when we first introduced path tracing in Section 3.1.10. Unfortunately, handling singular triangles appears to either require non-orthogonality of the direction field near a singularity or nontrivial remeshing around singularities, so additional work is needed for this be a complete representation of cross fields.

Bibliography

[ABE99] Nina Amenta, Marshall Bern, and David Eppstein. Optimal point placement for mesh smoothing. *Journal of Algorithms*, 30(2):302–322, 1999.

[ABO09] Bret D Anderson, Steven E Benzley, and Steven J Owen. Automatic all quadrilateral mesh adaption through refinement and coarsening. *Proceedings of the 18th International Meshing Roundtable*, pages 557–574, 2009.

[ACBCO17] Omri Azencot, Etienne Corman, Mirela Ben-Chen, and Maks Ovsjanikov. Consistent functional cross field design for mesh quadrangulation. *ACM Transactions on Graphics (TOG)*, 36(4):1–13, 2017.

[AFTR15] Cecil G Armstrong, Harold J Fogg, Christopher M Tierney, and Trevor T Robinson. Common themes in multi-block structured quad/hex mesh generation. *Procedia Engineering*, 124:70–82, 2015.

[BCE$^+$13] David Bommes, Marcel Campen, Hans-Christian Ebke, Pierre Alliez, and Leif Kobbelt. Integer-grid maps for reliable quad meshing. *ACM Transactions on Graphics (TOG)*, 32(4):98, 2013.

[BLH$^+$17] Pierre-Alexandre Beaufort, Jonathan Lambrechts, François Henrotte, Christophe Geuzaine, and Jean-François Remacle. Computing cross fields: A pde approach based on the ginzburg-landau theory. *Procedia Engineering*, 203:219–231, 2017. 26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain.

[BLP$^+$13] David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. Quad-mesh generation and processing: A survey. *Computer graphics forum*, 32(6):51–76, 2013.

[Bol21] Will Bolden. Complex function plotter. https://people.ucsc.edu/~wbolden/complex/, 2021. *archived* https://web.archive.org/web/20211021063704/https://people.ucsc.edu/~wbolden/complex/.

[Bro01] Christopher H Brooks. An introduction to amoeba, 2001.

[BS91] Ted D. Blacker and Michael B. Stephenson. Paving: A new approach to automated quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 32(4):811–847, 1991.

[Cam17] Marcel Campen. Partitioning surfaces into quadrilateral patches: A survey. *Computer Graphics Forum*, 36(8):567–588, 2017.

[CBK15] Marcel Campen, David Bommes, and Leif Kobbelt. Quantized global parametrization. *ACM Trans. Graph.*, 34(6), October 2015.

[CC78] Edwin Catmull and James Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-aided design*, 10(6):350–355, 1978.

[CCS$^+$21] Marcel Campen, Ryan Capouellez, Hanxiao Shen, Leyi Zhu, Daniele Panozzo, and Denis Zorin. Efficient and robust discrete conformal equivalence with boundary. *ACM Transactions on Graphics (TOG)*, 40(6):1–16, 2021.

[Cha12] John Chawner. Accuracy, convergence and mesh quality. *The Connector*, 2012.

[CL93] Brian Cabral and Leith Casey Leedom. Imaging vector fields using line integral convolution. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, page 263–270, New York, NY, USA, 1993. Association for Computing Machinery.

[CMP98] Scott A Canann, SN Muthukrishnan, and RK Phillips. Topological improvement procedures for quadrilateral finite element meshes. *Engineering with Computers*, 14(2):168–177, 1998.

[Coo67] S. A. Coons. Surfaces for computer-aided design of space forms. Technical report, USA, 1967.

[Cro21] Benjamin Crowell. *Special Relativity*. Fullerton College, feb 28 2021. [Online; accessed 2021-10-31].

[CSZZ19] Marcel Campen, Hanxiao Shen, Jiaran Zhou, and Denis Zorin. Seamless parametrization with arbitrary cones for arbitrary genus. *ACM Transactions on Graphics (TOG)*, 39(1):1–19, 2019.

[CW84] John B Czarnecki and Richard K Waddell. Finite-element simulation of ground-water flow in the vicinity of yucca mountain, nevada-california. Technical report, Geological Survey, Denver, CO (United States), 1984.

[CZK$^+$19] Wei Chen, Xiaopeng Zheng, Jingyao Ke, Na Lei, Zhongxuan Luo, and Xianfeng Gu. Quadrilateral mesh generation i: Metric based method. *Computer Methods in Applied Mechanics and Engineering*, 356:652–668, 2019.

[D'A00] E. F. D'Azevedo. Are bilinear quadrilaterals better than linear triangles? *SIAM J. Sci. Comput.*, 22(1):198–217, January 2000.

[DBG$^+$06] Shen Dong, Peer-Timo Bremer, Michael Garland, Valerio Pascucci, and John C Hart. Spectral surface quadrangulation. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 1057–1066. ACM, 2006.

[DCB13] Crawford Doran, Athena Chang, and Robert Bridson. Isosurface stuffing improved: acute lattices and feature matching. In *ACM SIGGRAPH 2013 Talks*, page 38. ACM, 2013.

[DSC09] Joel Daniels, II, Claudio T. Silva, and Elaine Cohen. Semi-regular quadrilateral-only remeshing from simplified base domains. In *Proceedings of the Symposium on Geometry Processing*, SGP '09, pages 1427–1435, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.

[DVPSH15] Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. Integrable polyvector fields. *ACM Transactions on Graphics (TOG)*, 34(4):1–12, 2015.

[EBCK13] Hans-Christian Ebke, David Bommes, Marcel Campen, and Leif Kobbelt. Qex: robust quad mesh extraction. *ACM Transactions on Graphics (TOG)*, 32(6):168, 2013.

[EDF10] Mohamed S Ebeida, Roger L Davis, and Roland W Freund. A new fast hybrid adaptive grid generation technique for arbitrary two-dimensional domains. *International Journal for Numerical Methods in Engineering*, 84(3):305–329, 2010.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[EGKT08] David Eppstein, Michael T Goodrich, Ethan Kim, and Rasmus Tamstorf. Motorcycle graphs: Canonical quad mesh partitioning. In *Computer Graphics Forum*, volume 27, pages 1477–1486. Wiley Online Library, 2008.

[EKMD10] Mohamed S Ebeida, Kaan Karamete, Eric Mestreau, and Saikat Dey. Q-tran: a new approach to transform triangular meshes into quadrilateral meshes locally. In *Proceedings of the 19th International Meshing Roundtable*, pages 23–34. Springer, 2010.

[FAR15] Harold J Fogg, Cecil G Armstrong, and Trevor T Robinson. Automatic generation of multiblock decompositions of surfaces. *International Journal for Numerical Methods in Engineering*, 101(13):965–991, 2015.

[FAR16] Harold J Fogg, Cecil G Armstrong, and Trevor T Robinson. Enhanced medial-axis-based block-structured meshing in 2-d. *Computer-Aided Design*, 72:87–101, 2016.

[GR09] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[JCR23] Jovana Jezdimirović, Alexandre Chemin, and Jean-François Remacle. Integrable cross-field generation based on imposed singularity configuration—the 2d manifold case. In *International Meshing Roundtable*, pages 343–369. Springer, 2023.

[JFH$^+$15] Tengfei Jiang, Xianzhong Fang, Jin Huang, Hujun Bao, Yiying Tong, and Mathieu Desbrun. Frame field generation through metric customization. *ACM Transactions on Graphics (TOG)*, 34(4):1–11, 2015.

[JT73] Ernest Jucovič and Marián Trenkler. A theorem on the structure of cell–decompositions of orientable 2–manifolds. *Mathematika*, 20(1):63–82, 1973.

[JTPSH15] Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. Instant field-aligned meshes. *ACM Trans. Graph.*, 34(6):189, 2015.

[KCPS13] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Globally optimal direction fields. *ACM Transactions on Graphics (TOG)*, 32(4):59, 2013.

[KFDT] Patrick Knupp, Lori Freitag-Diachin, and Boyd Tidwell. *MESQUITE Mesh Quality Improvement Toolkit User's Guide*. Sandia National Laboratories.

[KLF15] Nicolas Kowalski, Franck Ledoux, and Pascal Frey. Automatic domain partitioning for quadrilateral meshing with line constraints. *Engineering with Computers*, 31(3):405–421, 2015.

[Knu01] Patrick M Knupp. Algebraic mesh quality metrics. *SIAM journal on scientific computing*, 23(1):193–218, 2001.

[LCK21] Max Lyon, Marcel Campen, and Leif Kobbelt. Simpler quad layouts using relaxed singularities. In *Computer Graphics Forum*, volume 40, pages 169–180. Wiley Online Library, 2021.

[LHJ$^+$14] Ruotian Ling, Jin Huang, Bert Jüttler, Feng Sun, Hujun Bao, and Wenping Wang. Spectral quadrangulation with feature curve alignment and element size control. *ACM Transactions on Graphics (TOG)*, 34(1):11, 2014.

[LR89] David Lovelock and Hanno Rund. *Tensors, differential forms, and variational principles*. Courier Corporation, 1989.

[LS07] François Labelle and Jonathan Richard Shewchuk. Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. In *ACM Transactions on Graphics (TOG)*, volume 26, page 57. ACM, 2007.

[NM65] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[OJH96] Steven J Owen, Norman L Jones, and Jeffrey P Holland. A comprehensive modeling environment for the simulation of groundwater flow and transport. *Engineering with Computers*, 12(3):235–242, 1996.

[Per85] Ken Perlin. An image synthesizer. In *Computer Graphics, Vol. 19, No. 3.*, pages 287–296, 1985.

[Poi86] H. Poincaré. Sur les courbes définies par les équations différentielles (iv). *Journal de Mathématiques Pures et Appliquées*, 2:151–218, 1886.

[PPTSH14] Daniele Panozzo, Enrico Puppo, Marco Tarini, and Olga Sorkine-Hornung. Frame fields: Anisotropic and non-orthogonal cross fields. *ACM Transactions on Graphics (TOG)*, 33(4):134, 2014.

[PSS21] David Palmer, Oded Stein, and Justin Solomon. Frame field operators. In *Computer Graphics Forum*, volume 40, pages 231–245. Wiley Online Library, 2021.

[Pun91] Abraham P Punnen. A linear time algorithm for the maximum capacity path problem. *European Journal of Operational Research*, 53(3):402–404, 1991.

[PZ07] Jonathan Palacios and Eugene Zhang. Rotational symmetry field design on surfaces. *ACM Transactions on Graphics (TOG)*, 26(3):55, 2007.

[QRPG04] WR Quadros, K Ramaswami, FB Prinz, and B Gurumoorthy. Laytracks: a new approach to automated geometry adaptive quadrilateral mesh generation using medial axis transform. *International journal for numerical methods in engineering*, 61(2):209–237, 2004.

[RLS$^+$12] J.-F. Remacle, J. Lambrechts, B. Seny, E. Marchandise, A. Johnen, and C. Geuzainet. Blossom-quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm. *International Journal for Numerical Methods in Engineering*, 89(9):1102–1119, 2012.

[RVAL09] Nicolas Ray, Bruno Vallet, Laurent Alonso, and Bruno Levy. Geometry-aware direction field processing. *ACM Transactions on Graphics (TOG)*, 29(1):1–11, 2009.

[RVLL08] Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lévy. N-symmetry direction field design. *ACM Transactions on Graphics (TOG)*, 27(2):1–13, 2008.

[SA] L. Simons and N. Amenta. Anisotropy and cross fields. *Computer Graphics Forum*, n/a(n/a):e15132.

[SA17] Lance Simons and Nina Amenta. All-quad meshing for geographic data via templated boundary optimization. *Procedia Engineering*, 203:388–400, 2017. 26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain.

[Say15] Francisco-Javier Sayas. A gentle introduction to the finite element method. `https://team-pancho.github.io/documents/anIntro2FEM_2015.pdf`, 2015.

[Sch96] Robert Schneiders. A grid-based algorithm for the generation of hexahedral element meshes. *Engineering with computers*, 12(3-4):168–177, 1996.

[She96] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[She02] Jonathan Richard Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational geometry*, 22(1-3):21–74, 2002.

[SQB21] Andrea Schillaci, Maurizio Quadrio, and Giacomo Boracchi. A database of CFD-computed flow fields around airfoils for machine-learning applications, March 2021.

[SZC$^+$22] Hanxiao Shen, Leyi Zhu, Ryan Capouellez, Daniele Panozzo, Marcel Campen, and Denis Zorin. Which cross fields can be quadrangulated? global parameterization from prescribed holonomy signatures. *ACM Transactions on Graphics (TOG)*, 41(4):1–12, 2022.

[TPP$^+$11] Marco Tarini, Enrico Puppo, Daniele Panozzo, Nico Pietroni, and Paolo Cignoni. Simple quad domains for field aligned mesh parametrization. In *Proceedings of the 2011 SIGGRAPH asia conference*, pages 1–12, 2011.

[VCD$^+$16] Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. Directional field synthesis, design, and processing. *Computer graphics forum*, 35(2):545–572, 2016.

[vdWS05] Stéfan van der Walt and Nathaniel Smith. Matplotlib colormaps, 2005. [Online; accessed 2017-08-08].

[VGO$^+$20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[VO19] Ryan Viertel and Braxton Osting. An approach to quad meshing based on harmonic cross-valued maps and the Ginzburg–Landau theory. *SIAM Journal on Scientific Computing*, 41(1):A452–A479, 2019.

[VS17] Chaman Singh Verma and Krishnan Suresh. A robust combinatorial approach to reduce singularities in quadrilateral meshes. *Computer-Aided Design*, 85:99–110, 2017.

[VT11] Chaman Singh Verma and Tim Tautges. Jaal: Engineering a high quality all-quadrilateral mesh generator. In *Proceedings of the 20th International Meshing Roundtable*, pages 511–530. Springer, 2011.

[WA95] Herbert F Wang and Mary P Anderson. *Introduction to groundwater modeling: finite difference and finite element methods*. Academic Press, 1995.