BIJECTIVE DEFORMATIONS IN $\mathbb{R}^N$ VIA INTEGRAL CURVE COORDINATES

by

Lisa Huynh

A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Science

Committee:

_____	Dr. Yotam Gingold, Thesis Director
_____	Dr. Jyh-Ming Lien, Committee Member
_____	Dr. Zoran Duric, Committee Member
_____	Dr. Sanjeev Setia, Chairman, Department of Computer Science
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: 12-3-2013

Fall Semester 2013
George Mason University
Fairfax, VA

Bijective Deformations in $\mathbb{R}^n$ via Integral Curve Coordinates

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Lisa Huynh
Bachelor of Science
George Mason University, 2012

Director: Dr. Yotam Gingold, Professor
Department of Computer Science

Fall Semester 2013
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to my family for their unrelenting support.

Acknowledgments

I would like to thank the following people who made this possible. Christopher Vo encouraged me to enroll in the Master's program. Dr. Pearl Wang who has been my advisor for several years and who has always been available all these years. Dr. Gingold for pushing me to put my research into a thesis and working with me every step of this process. I would also like to thank Drs. Lien and Duric and the other committee members for their advice and assistance. Also, I am grateful to Ofir Weber, who shared the beautiful frog model in Figure 6.10.

Table of Contents

List of Tables

List of Figures

Abstract

BIJECTIVE DEFORMATIONS IN $\mathbb{R}^N$ VIA INTEGRAL CURVE COORDINATES

Lisa Huynh, M.S.

George Mason University, 2013

Thesis Director: Dr. Yotam Gingold

Shape deformation is a widely studied problem in computer graphics, with applications to animation, physical simulation, parameterization, interactive modeling, and image editing. In one instance of this problem, a "cage" (polygon in 2D and polyhedra in 3D) is created around a shape or image region. As the vertices of the cage are moved, the interior deforms. The cage may be identical to the shape's boundary, which has one fewer dimension than the shape itself, and is typically more convenient, as the cage may be simpler (fewer vertices) or be free of undesirable properties (such as a non-manifold mesh or high topological genus).

We introduce Integral Curve Coordinates and use them to create shape deformations that are bijective, given a bijective deformation of the shape's boundary or an enclosing cage. Our approach can be applied to shapes in any dimension, provided that the boundary of the shape (or cage) is topologically equivalent to an n-sphere.

Integral Curve Coordinates identify each point in a domain with a point along an integral curve of the gradient of a function f, where f has exactly one critical point, a maximum, in the domain, and the gradient of f on the boundary points inward. By identifying every

point inside a domain (shape) with a point on its boundary, Integral Curve Coordinates provide a natural mapping from one domain to another given a mapping of the boundary.

We evaluate our deformation approach in 2D. Our algorithm is based on the following three steps: (i) choosing a maximum via a grassfire algorithm; (ii) computing a suitable function f on a discrete grid via a construct called the *cousin tree*; (iii) tracing integral curves. We conclude with a discussion of limitations arising from piecewise linear interpolation and discretization to a grid.

Chapter 1: Introduction

Shape deformation is a widely studied problem in computer graphics. In shape deformation, a geometric shape is given along with information suggesting a desired, non-rigid deformation; a pure translation and rotation of the shape will not suffice. The idea is to find new locations for all points given new locations for some points. The deformation information typically comes in the form of desired locations for a subset of points, line segments, or faces. This is inherently an under-specified problem; without additional assumptions or desirata, such as that the deformation should vary smoothly [1] or be as close to a rigid deformation as possible [2].

This general problem statement has many applications [3]: in animation, an artist wishes to have a character walk without manually specifying the motion of every vertex; in finite element simulation, the motion of each element—such as the vertices of a polygon—must be extrapolated to the interior of the element; in parameterization, the surface of a 3D shape must be deformed to lie in a plane; in interactive modeling, an industrial designer or digital sculptor wishes to adjust the shape; in image editing, an artist wishes to adjust the shape of a region of a photograph. This is desirable for shapes in 2D and 3D, as well as for higher-dimensional shapes. The animation of a volumetric 3D shape, such as the CT scan of a human, is in fact a 4D problem, as the 3D volume must deform through time. Interpolating parameterized modeling spaces, such as the space of human body shapes [4], can be an arbitrary dimensional problem.

In the version of the problem that we study, a boundary or "cage" is created around a shape. As this cage is manipulated, the interior is also deformed. The cage may be identical to the shape's boundary, which has one fewer dimension than the shape itself, and is typically more convenient, as the cage may be simpler (fewer vertices) or be free of undesirable properties (such as a non-manifold mesh or high topological genus).

One important attribute for a deformation is bijectivity. A bijective function is a one-to-one mapping such that every point in the undeformed figure is be mapped to a distinct point in the deformed figure and vice-versa. Bijectivity is useful in that it ensures that no part of the shape flips "inside-out" as a result of deformation. Bijective deformations are notable due to the difficulty in achieving bijectivity in shape deformation. Few guaranteed bijective deformations have been introduced in 2D [3, 5, 6] or 3D [7, 8], and only one in 3D or higher dimensions [9].

The generalization of barycentric coordinates is related to the issue of cage deformation [5, 10–15]. Traditionally defined, barycentric coordinates express coordinate in the interior of a simplex (triangle in 2D, tetrahedron in 3D) in terms of the vertices of the simplex. The simplex can be thought of as a simple cage around its interior; the deformations induced by modifying vertices of the simplex are bijective. Generalized barycentric coordinates extend this idea to more complex polygons or polyhedra than triangles and tetrahedra, but are not bijective [16].

We propose a technique that creates a guaranteed bijective deformation of the shape given a bijective deformation of its boundary or a cage in any dimension. To do so, we introduce Integral Curve Coordinates that identify each point in the shape as a point along an integral curve of the gradient of a function f. Given these coordinates and a bijective deformation of the boundary points, it generates a mapping from the original shape to the deformed shape.

Our contributions are:

- Integral Curve Coordinates that can be used to produce a guaranteed bijective deformation between two closed, simply connected domains in any dimension given a bijective deformation of their boundaries.
- An algorithm to create a scalar function space on a regularly discretized simply connected domain in any dimension; functions in this space have exactly one critical point, a maximum.

- A practical (simple, robust, and efficient) algorithm for tracing 2D integral curves on a (piecewise linear) triangulated regular grid.

Chapter 2: Literature Review

2.1 Background

Before continuing discussion on this work with Integral Curve Coordinates, a few concepts much be covered first. These concepts contain the foundation that this work and others rely upon for a theoretical base.

2.1.1 Morse Theory

Several general theories have been formed for studying smooth manifolds, and one of the most useful is Morse Theory. A manifold is a topological space where the surface around every point locally resembles Euclidean space, or $\mathbb{R}^n$.

Morse Theory extends the observation that the topology of smooth manifolds can be correlated to the critical points of a smooth function on the manifold. The critical points may be defined as the points in the function where the derivative is zero, or as either a maximum, minimum, or saddle point. By examining the differentiably functions on the shape, Morse Theory allows one to analyze information about the topology of manifold shapes [17]. Of particular interest is creating a homeomorphism from one space to another, which is a bijective mapping.

Definition 1. *A homeomorphism $f\colon X \to Y$ is a 1-1 onto function, such that both f, f^{-1} are continuous. We say that X is homeomorphic to Y, $X \cong Y$, and that X and Y have the same topological type.*

All of this is well for continuous spaces, but digital geometry is typically represented using discrete (and often piecewise linear, as in the present work) functions. Fortunately, Morse Theory has been extended to such discrete functions [18, 19].

This is important as we use the topology in order to bijective shape deformations. The Integral Curve Coordinates we use map every point in the domain to a point along an integral curve of the gradient for some function f. This function f is required to be differentiable with a single critical point, a maximum, within the domain. These integral curves of the gradient of a function are also sometimes called integral lines [20]. We make use of several concepts when formally defining the Integral Curve Coordinates, following [20].

Definition 2. *An integral line* $x\colon \mathbb{R} \to M$ *is a maximal path whose tangent vectors agree with the gradient of a function* h*, that is,* $\frac{d}{dt}x(t) = \nabla h(x(t))$ *for all* $t \in \mathbb{R}$*. We call* $\mathrm{org}x = lim_{t\to\infty}x(t)$ *the origin and* $\mathrm{dest}x = lim_{t\to+\infty}x(t)$ *the destination of the path* x*.*

Each integral line is open at both ends, and the limits at each end exist, as M *is compact. A critical point is an integral line by itself.*

Theorem 1. *Integral lines have the following properties:*

(a) two integral lines are either disjoint or the same,

(b) the integral lines cover all of M*,*

(c) and the limits $\mathrm{org}x$ *and* $\mathrm{dest}x$ *are critical points of* f*.*

2.1.2 Bijectivity

A bijective function $f\colon X \to Y$, also known as a one-to-one correspondence, is one that gives an exact matching of elements from two sets (X, Y). It is both injective (into) and surjective (onto). With an injective function, the mapping is one-to-one as every element from the first set (X) is mapped to a distinct element in the second set (Y). A surjective function occurs when every element from the second set (Y) has a corresponding element in the first (X). Thus, every element is uniquely paired with an element from the other set.

As this mapping is one-to-one, of course there is an inverse function to map Y back to X. Many definitions for mapping contain bijectivity as one of their prerequisites, and it is

a useful property to have when performing deformations. For deformations, the one-to-one property (injectivity) is the most difficult to achieve. A lack of injectivity manifests as regions of X that collapse or invert (flip "inside-out") when mapped via f. For piecewise linear shapes, this corresponds to collapsed or inverted elements (triangles in 2D, tetrahedra in 3D, etc). Expressed symbolically, a function f must have a Jacobian whose determinant is everywhere positive to be injective:

$$det(J_f(x)) > 0 \qquad \forall x \in X \tag{2.1}$$

Where the determinant is negative, f has locally flipped X inside out (inverted). Where the determinant is zero, f has locally collapsed X, though not inverted it. (In 2D, a triangle collapses to a line or point.)

For piecewise linear shapes, the Jacobian is piecewise constant within each element and can be constructed as follows. Consider a k-dimensional simplex P (triangle in 2D or tetrahedron in 3D). P has $k+1$ vertices $v_0, v_1, \ldots, v_k$. Ignoring the overall translation of f, the mapping $f(P)$ can be expressed via matrix multiplication. To ignore translation, choose a vertex of P arbitrarily (v_0, without loss of generality). The matrix we seek maps $v_i - v_0$ to $f(v_i) - f(v_0)$. Assuming that P is non-degenerate, $v_1 - v_0, \ldots, v_k - v_0$ are linearly independent. The matrix M such that $M(v_i - v_0) = f(v_i) - f(v_0)$ is, therefore, the product of two matrices whose columns are vertices:

$$M = [f(v_1) - f(v_0)|f(v_2) - f(v_0)|\cdots|f(v_k) - f(v_0)]\,[v_1 - v_0|v_2 - v_0|\cdots|v_k - v_0]^{-1} \tag{2.2}$$

The determinant of M determines whether f is locally injective within P. If the determinant is zero, f has locally collapsed P to a line segment or point. If the determinant is negative, f has inverted P.

2.2 Related Work

One main distinction among deformation techniques is between deformations that warp a shape intrinsically versus deformations that warp the ambient space (so-called space-warp approaches). For example, in a space-warp approach a 2D shape is represented in terms of its absolute coordinates. The deformation affects all of $\mathbb{R}^2$, so the absolute coordinates of the shape also deform [1, 3, 21]. Intrinsic shape deformation approaches deform the shape itself, by calculating a deformation in terms of relative coordinates and without regard for the ambient space [2, 22, 23]. Techniques such as cage-based deformation fall somewhere in between. Cage-based deformations enclose the ambient space around the shape in a sort of structural boundary; this boundary, when deformed, induces a deformation of the interior points [13, 24–27]. Several approaches take a more flexible hybrid approach by allowing a flexible combination of cages, line segments, and sparse points while still computing an intrinsic shape deformation [28, 29].

A family of approaches for intrinsic deformation are physics-based, simulating the deformation of a shape as if made of a user-specified (typically elastic) material [30, 31]. Due to the potential computational intensity, such techniques often trade accuracy for real-time interactivity.

A related problem is the generalization of barycentric coordinates beyond simplices, to polytopes other than triangles in 2D and tetrahedra in 3D [5, 10–15, 32–34]. In barycentric coordinates, the coordinates of a point are defined as the center of mass of weights placed at the vertices of the simplex. Barycentric coordinates specify a scalar weight for each vertex of the simplex; to convert a barycentric coordinate to a Euclidean coordinate, one computes the weighted sum of the simplex's vertices. So long as the simplex is non-degenerate, barycentric coordinates are bijective with Euclidean coordinates. Generalizations of barycentric coordinates allow polytopes other than simplices to be used.

Generalized barycentric coordinates are used in finite element analysis and shape deformation. Unfortunately, generalized barycentric coordinates have been shown to be not bijective [16].

2.2.1 Bijectivity

Few deformation approaches guarantee bijectivity. Bijectivity guarantees that the deformation does not invert (flip inside-out) or collapse any part of a shape. One application of shape deformation is to map a texture (rectangular 2D image) onto the surface of a shape, which is essentially a 2D-to-2D deformation problem. Several approaches have been presented for creating bijective texture maps with positional constraints [35–37].

Few guaranteed bijective approaches have been introduced. In two-dimensions, Weber et al. [5] introduced conformal mappings based on complex coordinates. Xu et al. [6] introduced a solution to the challenging problem of finding 2D locations for the vertices of a triangular graph given locations for vertices on its boundary such that no triangles end up inverted. Their approach can be used to solve for a bijective cage-based deformation with the additional constraint that the discretization remain unchanged, including detecting inputs with no solution and suggesting input boundary vertex positions that are solvable.

While generalized barycentric coordinates have been shown by Jacobson [16] not to yield bijective mappings, Schneider et al. [8] recently introduced a technique that splits a deformation into a finite number of steps, each of which is implemented via generalized barycentric coordinates. While the analysis provably guarantees bijectivity only for convex cages, in practice the technique creates bijective mappings at pixel accuracy. An additional limitation is that a continuous non-self-intersecting interpolation between the source and target cages is required, which is an unsolved problem for dimensions greater than two. This technique has not yet been shown to have any theoretical guarantees in 3D.

Lipman's restricted functional space, introduced initially in 2D [38] and extended to 3D and higher by Aigerman and Lipman [9] can be used to find bijective deformations of piecewise linear meshes similar to a desired one. The technique maps mesh elements with

strict bounds by projecting an input simplicial map onto the space of bounded-distortion simplicial maps. While they do not prove convergence for given bounds in general, the technique is guaranteed given a bijective deformation of the mesh boundary.

The approaches of Weinkauf et al. [39] and Jacobson et al. [40] both create restricted function spaces that prevent undesirable extrema (maxima and minima), but cannot control the placement of saddles. Jacobson et al. applied their technique to shape deformation. Our approach creates a restricted function space with exactly one extrema and no saddles, in any dimension.

The approach of Schüller et al. [7] prevents inverted elements (i.e. guarantees local injectivity) by augmenting any variational deformation approach with a non-linear penalty term that goes to infinity as elements collapse. The approach therefore requires a continuous deformation from an initial shape, and is incompatible with hard constraints (such as a required target pose).

The goal of this work is to introduce a cage-based deformation technique that guarantees bijectivity in any dimension, and provide a practical piecewise linear implementation for the 2D problem.

Chapter 3: Overview

Our goal is to deform the interior of a shape given a deformation of its boundary or enclosing cage. Formally, our approach takes as input

1. The boundary ∂D of a closed, simply connected domain D of $\mathbb{R}^n$.
2. A homeomorphism h deforming ∂D.
3. A set S of points in D.

and outputs the deformed location of each point in S. In other words, our approach extends h to the interior of D.

To accomplish this, we introduce Integral Curve Coordinates (formally defined below) to identify a point $\mathbf{p}$ in the domain D with an integral curve $\mathbf{x}(t)$ and a parameter t_p such that $\mathbf{x}(t_p) = \mathbf{p}$. The integral curves follow the gradient of a special function f_D: $\frac{d}{dt}\mathbf{x}(t) = \nabla f_D(\mathbf{x}(t))$. The function f_D is constructed such that f_D has exactly on critical point in D, a maximum, and its gradient on the boundary points inward.[1] Integral curves of the gradient of a function are sometimes called integral lines [41]. The integral lines of a C^1 function are well-defined everywhere except critical points, and two integral lines never meet unless they are identical. Because every integral curve of ∇f_D traces a path from a point on the boundary of the domain to the maximum, we can uniquely identify every integral curve with its boundary point. We denote the integral curve passing through boundary point $\mathbf{b}_m$ as $\mathbf{x}_{\mathbf{b}_m}(t)$. The Integral Curve Coordinate of a point $\mathbf{p}$ is

$$\mathcal{I}_{f_D}(\mathbf{p}) = \begin{cases} (\mathbf{b}_m, t) & \text{if } \mathbf{p} \neq \mathbf{p}_0 \\ \emptyset & \text{if } \mathbf{p} = \mathbf{p}_0 \end{cases} \tag{3.1}$$

[1] Harmonic functions, which obey the strong maximum principle, are not suitable in dimensions greater than two, as they may contain spurious saddle points.

where $\mathbf{x}_{\mathbf{b}_m}(t) = \mathbf{p}$ and $\mathbf{p}_0$ is the maximum point of f_D.

Since distinct integral curves never meet, $\mathcal{I}$ is a bijection from points in the domain to Integral Curve Coordinates. Moreover, given a bijective deformation h of the boundary of D, $h(\partial D) = \partial D'$, we can similarly construct a special function $f_{D'}$ on D' to create $\mathcal{I}_{f_{D'}}$. Transforming from $\mathcal{I}_{f_D}$ to $\mathcal{I}_{f_{D'}}$ can therefore be achieved by mapping the b_m to $h(b_m)$.

To summarize, the steps to extend h to a point $\mathbf{p}$ in the interior of D are as follows (Figure 3.1):

1. Create a function $f_D \colon D \to \mathbb{R}$ with a single critical point, a maximum, and whose gradients on the boundary point inward.

2. Create a similar function f'_D for D'.

3. Compute the Integral Curve Coordinate $I_p = \mathcal{I}_{f_D}(\mathbf{p})$ by tracing the integral curve of ∇f_D in both directions from $\mathbf{p}$.

4. Transform I_p to $I'_{p'}$ by mapping the boundary point b_m of the Integral Curve Coordinate with h (or, if $p = p_0$, identifying the unique maximum p_0 of f_D with the unique maximum p'_0 of f'_D).

5. Compute $\mathcal{I}^{-1}_{f_{D'}}(I'_{p'})$ by tracing the integral curve of $\nabla f_{D'}$ from the boundary point to the maximum.

(To warp points backwards from D' to points in D, swap D with D' and h with h^{-1} in the above steps.)

Our implementation of this approach, described in the following chapters, creates functions on a regular grid discretization of the the domain. We trace integral curves on a piecewise linear interpolation of the function values. In the piecewise linear setting, gradients are piecewise constant and discontinuous across piece boundaries. This guarantees monotonic interpolation (no spurious critical points within an element) and greatly reduces numerical issues in tracing integral lines. However, it violates the assumption that integral lines never meet, and can result in regions of the shape collapsing (though not inverting).

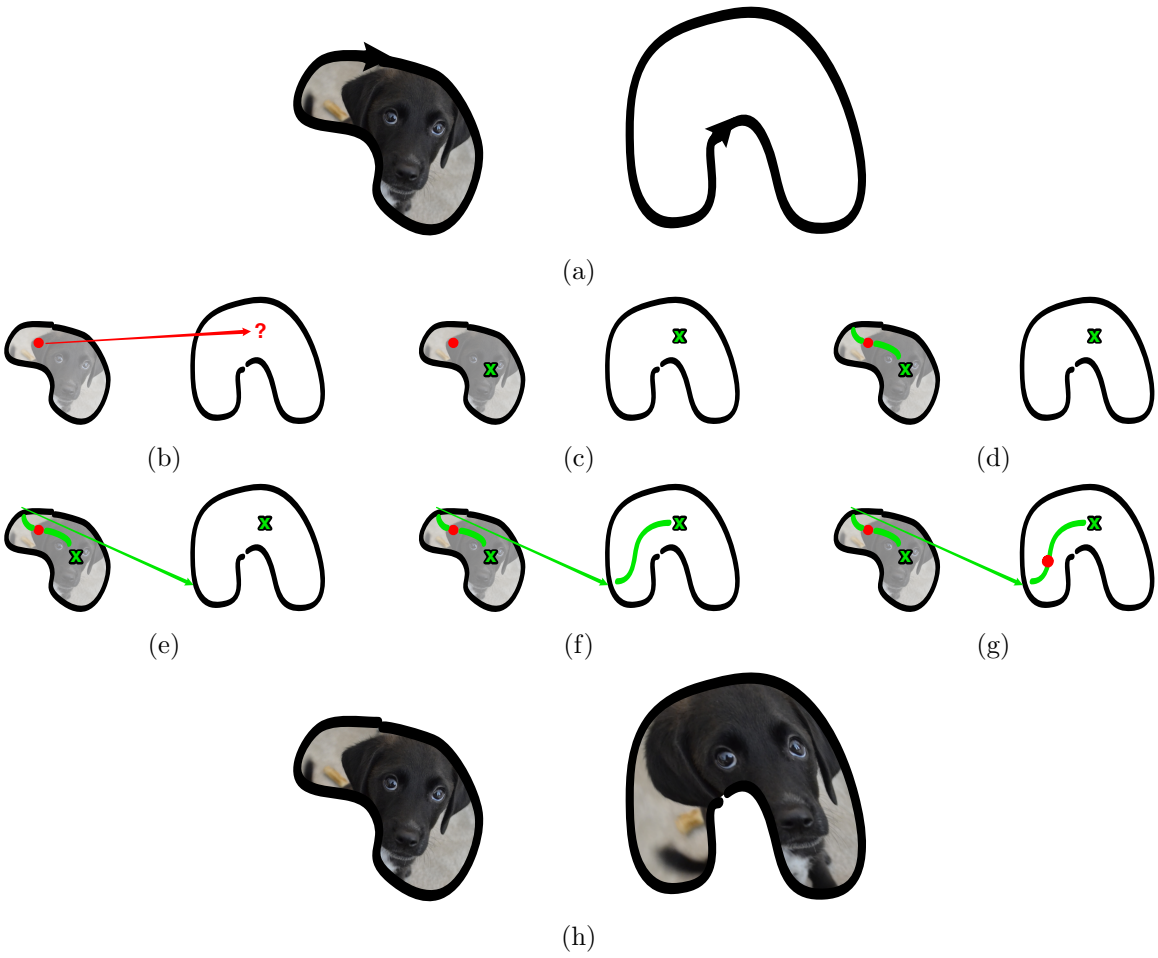


Figure 3.1: Overview of our approach: (a) Given a shape and a homeomorphism of its boundary, (b) to find the position of an interior point within the deformed boundary (c) we first create two functions, each with only a single critical point, a maximum. (d) We then trace the function's integral line passing through the point up to the maximum and down to the boundary. (e) Following the homeomorphism of its boundary, (f) we trace the integral line of the function from the boundary point in the deformed domain (g) to find the position of the interior point within the deformed boundary. (h) Repeating this process everywhere allows us to extend the boundary homeomorphism to the interior.

Chapter 4: Function Creation

In order to trace integral lines, we need to create suitable functions in the interior of the undeformed and deformed shape boundaries (or cages). Namely, we need to create functions with a single critical point, a maximum, and whose gradients on the boundary point inward. Our discrete implementation creates a function space satisfying the above requirements. The function space takes the form of a set of inequality relationships on edges of a regular grid enclosing the shape boundary or cage.

Our approach first chooses a grid vertex to be the maximum (Section 4.1) and then generates inequality relationships and vertex values such that all other points are regular (Section 4.2). We then trace integral lines on a piecewise linear interpolation of the function values (Chapter 5).

4.1 Maximum Selection via the Grassfire Transform

Any grid vertex can be selected to be the maximum. However, because integral lines converge at a maximum, we wish to choose a maximum vertex in the center of the shape. We find such a point with the grassfire transform [42].[1] The grassfire transform iteratively "burns away" the boundary of a shape; we use it to find the farthest point from the boundary. To do so, we construct a regular grid covering the mesh. The vertices of the grid are placed in a list. We iteratively remove vertices from the list that are not completely surrounded (along axis-aligned directions) by other vertices in the list, until none are left. Any vertex from the final iteration can then be arbitrarily chosen. This algorithm's pseudocode is presented in Algorithm 1. Illustrations of this algorithm can be seen in Figures 4.1 and 4.2.

[1] A variation of the grassfire transform, the extended grassfire transform [43], could be used to generate better centers of 2D shapes.

Algorithm 1 *grassFireMaximum*(*vertexList*)

candidateMaxima ⇐ *vertexList*
while *candidateMaxima* is not empty **do**
    keepVertices ⇐ vertices in *candidateMaxima* whose axis-aligned grid neighbors are all also in *candidateMaxima*
    if *keepVertices* is empty **then**
        break
    else
        candidateMaxima ⇐ *keepVertices*
    end if
end while
return vertex from *candidateMaxima*

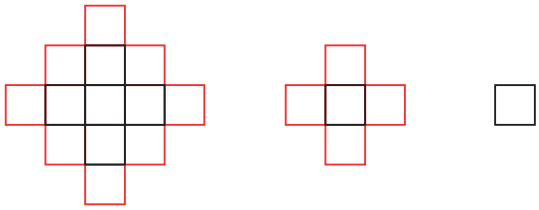


Figure 4.1: The grassfire farthest-point algorithm iteratively removes boundary cells.

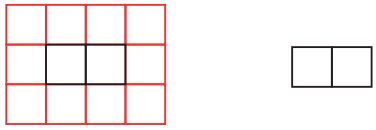


Figure 4.2: The grassfire farthest-point algorithm resulting in a tie. Ties are broken arbitrarily.

4.2 Assigning Function Values

Once the maximum is chosen, we assign function values to all grid vertices inside and just enclosing the shape boundary or cage. To do so, we build a function space containing functions that have a single critical point, a maximum, and gradients on the boundary that point inward. The function space takes the form of a *cousin tree* of inequality relationships on edges of the regular grid.

Definition 3. *Let T be a tree defined on a subset of a regular grid. We call T a* cousin tree *if every pair of axis-aligned neighbor vertices in the tree are related to each other as parent-child or as tree cousins (Figure 4.3). Two vertices are tree cousins if they are neighbors in the grid and their tree parents are also neighbors in the grid.*

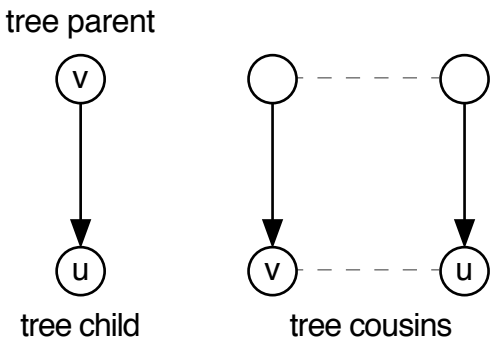


Figure 4.3: Cousin tree vertex relationships: Two vertices u, v that are axis-aligned neighbors in the grid must be either parent-child (left) or tree cousins (right). Tree cousins' parent vertices must also be axis-aligned neighbors in the grid.

Algorithm 2 $createCousinTree(maximumVertex, boundaryVertices)$

```
frontier ← {maximumVertex}
functionValue ← empty dictionary
functionValue(maximumVertex) := 10
while frontier is not empty do
    frontierNext ← {}
    for dimension in fixed dimension order do
        for v in frontier do
            if v or any axis-aligned neighbor of v is within boundary then
                candidates ← { unvisited neighbors of v along dimension direction }
                // A vertex outside the boundary can't have a child inside.
                filtered ← {n ∈ candidates unless n is within boundary and v is outside }
                functionValue(filtered) := functionValue(v) * 0.95
                frontierNext ← frontierNext ∪ filtered
            end if
        end for
    end for
    frontier ← frontierNext
end while
return functionValue
```

Our algorithm for creating a cousin tree is provided in Algorithm 2 and illustrated in Figure 4.4. The algorithm is a form of breadth-first search (BFS) that, for all vertices on the frontier in a given iteration, always expands first along the first coordinate axis, second

along the second coordinate axis, and so on. (The order of coordinate axes is not important so long as it is consistent across all iterations.) When a vertex expands, it is the tree parent of the vertices it expands into. In 2D, this means that all potential children along the x-axis are connected to the tree in one round. The next round, all potential children along the y-axis are connected to the tree, if they have not already been included. Vertices outside of the boundary and whose neighbors are all outside of the boundary are not added to the tree. See the Appendix for a proof that this algorithm indeed creates a cousin tree.

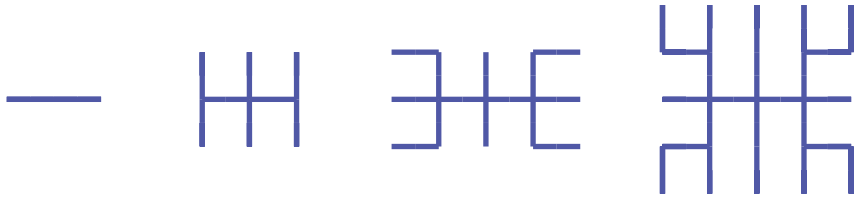


Figure 4.4: A cousin tree being built starting from the x direction.

The cousin tree represents our function space; the edge from a parent to a child vertex represents the inequality $f(parent) > f(child)$. Note that the Manhattan distance lies within this function space, as breadth-first search can be used to directly compute the graph distance to a given node. See the Appendix for a proof that this function space contains no critical points other than the maximum.

One the cousin tree is created, we can assign values to grid vertices such that the tree parent always has greater value than the tree child. In our experiments, we set the maximum vertex's value to 10 and use the following simple formula to assign values to the remaining vertices:

$$f(child) := f(parent) * .95 \tag{4.1}$$

We ensure that the function has inward pointing gradients at the boundary by treating vertices located outside the shape boundary or cage as having values smaller than its neighbors inside. Figure 4.5 depicts an example function.

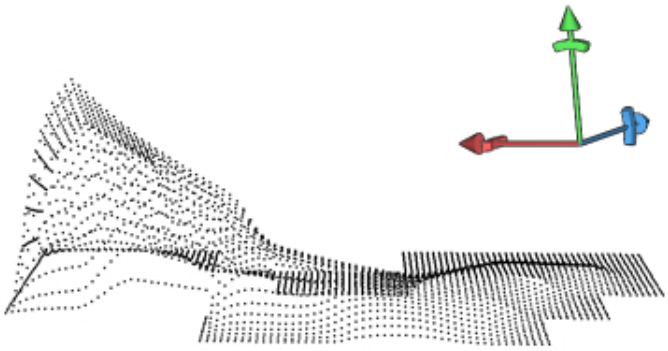


Figure 4.5: A 3D visualization of an example of the function values created using Equation 4.1 and the cousin tree.

Chapter 5: Tracing Integral Lines

With a suitable function f_D in hand, we are ready to trace integral lines to convert a point $\mathbf{p}$ in the interior of the shape to its Integral Curve Coordinate $\mathcal{I}_{f_D}(\mathbf{p}) = (\mathbf{b}_m, t)$, where $\mathbf{b}_m$ is the point on the boundary reached by tracing the integral line through $\mathbf{p}$ in the decreasing direction (gradient descent), and t is the fraction of arc-length along the integral line from $\mathbf{p}$ to the maximum, reached by gradient ascent from either $\mathbf{p}$ or $\mathbf{b}_m$. (Note that given a piecewise linear boundary or cage, we store $\mathbf{b}_m$ in barycentric coordinates with respect to the boundary piece it belongs to.)

In order to perform gradient ascent and descent, the function values defined at vertices of the regular grid must be interpolated monotonically, so that spurious critical points are not introduced by the interpolation. For our 2D implementation, we perform piecewise linear interpolation via a regular triangulation that divides each grid cell into two triangles. Notably, piecewise linear interpolation is monotonic and has constant gradient, which greatly reduces numerical issues when tracing integral lines and simplifies arc-length parameterization.

Because the gradient is constant within each triangle, we can trace the integral line from a given starting point by simply computing the intersection of the ray from the starting point in the gradient direction with the boundary of the triangle. However, naive implementation of these numerical calculations results in errors due to limitations in floating point precision. Such imprecision can cause discrepancies and inconsistencies such as loops. In order to obviate numerical precision issues, we separate the topological calculation (which edge of the triangle the integral line passes through) from the geometric calculation (the coordinates of the point on the edge).

Our algorithm traces an integral line by iteratively computing the sequence of triangle edges it intersects (and points on those edges). As a special, initial case, the integral line

may originate from a point inside a triangle, but thereafter will be tracked via the triangle edges it intersects.

Our integral line tracing algorithm proceeds as follows. Pseudocode is provided in Algorithms 3–5. A vector is created from the incoming point to the corner opposite the incoming edge (the red line in Figure 5.1, right). The opposite corner is the triangle vertex that the incoming edge is not incident to. The sign of the cross product of these two vectors determines the outgoing edge of the integral line (Table 5.1 with edge labeling given by Figure 5.2). This robustly and stably computes the outgoing edge. The outgoing point (p_2 in Figure 5.1, right) is then computed via the intersection of the ray from the incoming point along the gradient with the outgoing edge. Tracing proceeds in the triangle on the other side of the outgoing edge.

In the special, initial case of an integral line originating at a point inside a triangle, the cross product of the gradient is computed with each of the three vectors from the point to the triangle's vertices (Figure 5.1, left). The sign of the cross products determines which vectors the gradient is to the left and right of, indicating the outgoing edge accordingly. The point on the edge can then be computed numerically, and the general case of the algorithm proceeds as usual.

Figure 5.1: An integral line is traced (left) from p_0 inside triangle **A**. The gradient direction (the purple arrow) is compared to the red dotted lines, indicating which edge of **A** is intersected. The line/line intersection gives us the next point p_1 lying a triangle edge, which is the general case. Tracing then proceeds inside the opposite triangle **B** (right), and the gradient direction only needs to be compared to one dotted red line.

Finally, because our piecewise linear domain is not C^1, integral lines may meet. This manifests in our integral line tracing algorithm as a triangle whose gradient points backwards

Algorithm 3 $fetchNext(point, edge, triangle, gradient)$

// If *edge* is NULL, *point* is inside *triangle*
if *edge* is NULL **then**
    compute the cross product of *gradient* with each of the three vectors from *point* to each of *triangle*'s corners
    next edge is given by the sign of the cross products
    next point is the intersection of the next edge and the ray from *point* along *gradient*
else
    map *triangle* to base case triangle according to Figure 5.2
    get outward normal to *edge*
    if dot product of *outwardnormal* and *gradient* > 0 **then**
        $d \leftarrow$ dot product of *gradient* and *edge*
        next point is the endpoint of *edge* that gradient points towards according to d
        next edge is the adjacent edge that contains next point
    else
        op $\leftarrow$ the opposite corner to *edge*
        compute the cross product of the vector from *point* to *op* and *gradient*
        next edge is given by the sign of the cross product according to Table 5.1
        next point is the intersection of next edge and the ray from *point* along *gradient*
    end if
end if
next triangle is the adjacent triangle that contains next edge
return next point, next edge, next triangle

Algorithm 4 $traceUphill(point, maximum, functionvalues, triangles)$

path $\leftarrow$ []
triangle $\leftarrow$ find starting triangle in *triangles*
edge $\leftarrow$ NULL
while *point* is not *maximum* **do**
    gradient $\leftarrow$ *triangle*'s gradient based on $functionvalues$
    point, *edge*, *triangle* $= fetchNext(point, edge, triangle, gradient)$
    $path.append(point)$
end while
return *path*

Algorithm 5 $traceDownhill(point, boundary, functionvalues, triangles)$

path $\leftarrow$ []
triangle $\leftarrow$ find starting triangle in *triangles*
edge $\leftarrow$ NULL
while *point* inside *boundary* **do**
    gradient $\leftarrow -1 *$ *triangle*'s gradient based on $functionmap$
    point, *edge*, *triangle* $= fetchNext(point, edge, triangle, gradient)$
    $path.append(point)$
end while
return *path*

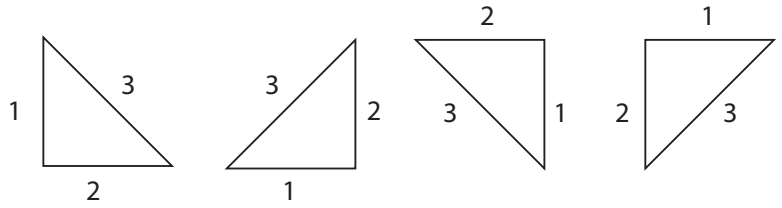


Figure 5.2: An edge labeling for triangles in the regular grid, such that one lookup table (Table 5.1) can be used to compute the outgoing edge.

Table 5.1: Edge Sign Mapping

Edge	Sign	Next Edge
1	+	3
1	-	2
2	+	1
2	-	3
3	+	2
3	-	1

against the incoming edge. We call such edges *compression edges*. Without special care, the integral line would zig-zag or staircase between the two triangles. We detect this by evaluating the sign of the dot product of the outward (from the triangle) normal vector to the incoming edge with the triangle's gradient. Because our triangles triangulate a regular grid, these dot products reduce to simple sign comparisons between components of the gradient (without arithmetic). If the dot product is positive, then we select as the next point the end point of the edge pointed towards by the gradient. Which end point can also be determined by a dot product that also reduces to a simple sign comparison between components of the gradient. The next edge is the edge of the triangle sharing the next point. The next triangle is the triangle opposite the next edge. This can be assumed to be the end result of the bouncing between the two triangles as their gradients merge, and so skipping the intermediate steps.

We note that computing the sign of a cross product or dot product is amenable to a symbolic perturbation scheme [44–46] for robust, exact computation. We did not implement such a scheme, as problems did not arise from our floating point implementation.

5.1 Computing Deformations

Pseudocode combining all steps of our algorithm is presented in Algorithms 6–8.

Algorithm 6 *preprocessing*(*boundary*)

create *grid* based off of *boundary*
$maximum \leftarrow getMaximum(grid)$
$function\ values \leftarrow createCousinTree(maximum, boundary)$
triangulate cousin tree vertices
return $maximum, function\ values, cousin\ tree, triangles$

Algorithm 7 $CartesianToICC(interior\ cartesian\ points, boundary)$

$maximum, function\ values, cousin\ tree, triangles \leftarrow preprocessing(boundary)$
// Integral Curve Coordinates store a boundary point and fraction along an integral line
integral curve coordinates $\leftarrow$ []
for *point* in *interiorcartesianpoint* **do**
    $ascentPath \leftarrow traceUphill(point, maximum, function\ values, triangles)$
    $descentPath \leftarrow traceDownhill(point, boundary, function\ values, triangles)$
    $path \leftarrow ascentPath + descentPath$
    boundary relative location $\leftarrow$ end of *descentPath* in *boundary* relative coordinates
    integral curve coordinates.append(boundary relative location, fraction along path)
end for
return *integral curve coordinates*

Algorithm 8 $ICCToCartesian(integral\ curve\ coordinates, boundary)$

$maximum, function\ values, cousin\ tree, triangles \leftarrow preprocessing(boundary)$
cartesian coordinates $\leftarrow$ []
for (boundary relative location, fraction along path) in *integral curve coordinates* **do**
    point $\leftarrow$ boundary relative location as absolute point
    $path \leftarrow traceUphill(point, maximum, function\ values, triangles)$
    calculate *cartesian coordinate* based on *path* and fraction along path
    cartesian coordinates.append(*cartesian coordinate*)
end for
return *cartesian coordinates*

Chapter 6: Results

In this section, we present deformations computed using our algorithm. Rather than computing the per-pixel deformation, we compute the deformation at a sparse set of points and then linearly interpolate this deformation inside the triangles of a Delaunay triangulation created using the Triangle mesh generator [47].

6.1 Implementation

Our experiments were run on an Intel Core i5 CPU M 460 with 4 cores running at 2.53GHz. Our (unoptimized) implementation has been written in Python with the following external dependencies:

- NumPy for linear algebra
- PyCairo, bindings for the Cairo graphics library, for the general point-in-polygon test (the `in_fill` function) to test whether a point is within the boundary of the figure.
- Triangle [47] for producing Delaunay triangulations given a 2D boundary polygon and internal points.

6.2 Examples

Figures 6.1–6.8 demonstrate a variety of shapes deformed using Integral Curve Coordinates. Identical deformations at 1x and 2x grid resolutions are shown for every shape, and one shape is shown at 5x resolution as well (Figure 6.7 and 6.8). Our deformation never creates inverted elements. Increasing the grid resolution does not have a discernable affect on the integral lines or texture mapping.

We experimented with the stability of Integral Curve Coordinates. Figure 6.6 conversions from Integral Curve Coordinates after applying 5°, 45°, and 90° rotations to the undeformed bunny in Figure 6.5, left. The 90° rotation appears identical; this is also equivalent to swapping the order of expansion in the cousin tree breadth-first search algorithm. If our discrete algorithm were perfectly stable, the 5°, 45° rotations would also appear indistinguishable. Figure 6.5, right, depicts an identical deformation to the boundary, but the maximum selection algorithm was modified to return a point to the right of the one computed by the grassfire transform. Our coordinate transform is somewhat sensitive to the choice of maximum.

Running times at various grid resolutions are presented in Table 6.1. In our admittedly unoptimized implementation, the performance bottleneck is the BFS cousin-tree generation and the integral line tracing.

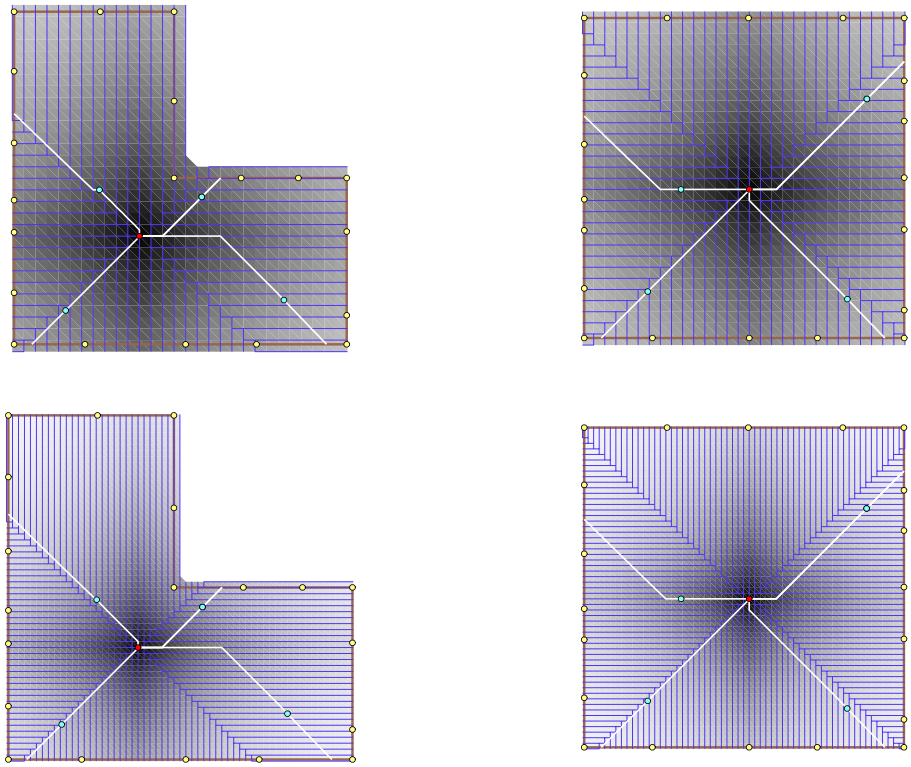


Figure 6.1: The cousin tree, points, and integral lines for an L shape deformed into a square, at 1x and 2x resolution.

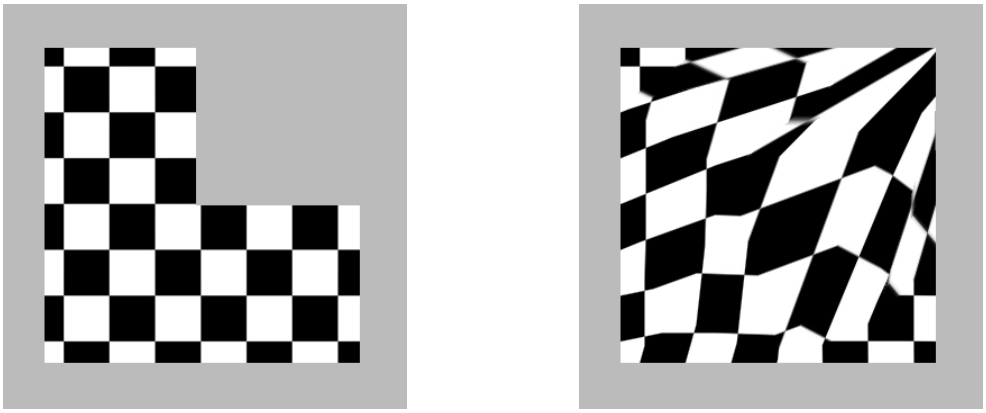


Figure 6.2: A checkerboard pattern applied to the undeformed L shape (left) and its deformation into a square (right).

6.3 Comparison

Figures 6.9–6.11 compare the results of our deformation approach to Complex Barycentric Coordinates [5], Controllable Conformal Maps [26], Composite Mean Value Mappings [8], and Locally Injective Mappings [7]. Our deformation, while less smooth than some of the techniques, can be generalized to any dimension. Although some integral lines do flatten out as a result of deformation, we can see that no inverted elements are created by our approach.

We can clearly see in Figure 6.11 that when a shape becomes unevenly narrowed and skewed, the maxima can shift relative location quite substantially. The integral lines near such sharp, concave areas may lead to collapsed regions of the shape.

6.4 Expansion Edges

Compression edges (Chapter 5) occur when the gradients on either side of an edge point towards the edge, resulting in multiple integral lines meeting. Similarly, we call an edge an *expansion edges* when the gradients on either side point away from it (Figure 6.12, left). More formally, an expansion edge is a shared edge where in both triangles:

$$ExpansionEdge : InwardNormal(currentedge) \cdot Gradient > 0 \tag{6.1}$$

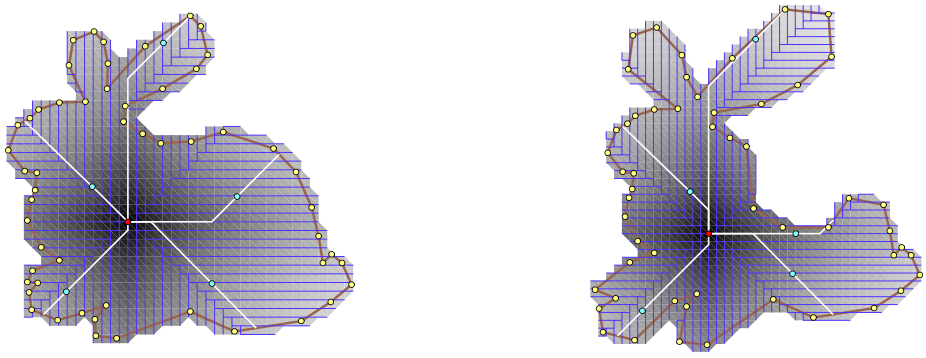


Figure 6.3: The points and integral lines for the rabbit figure.

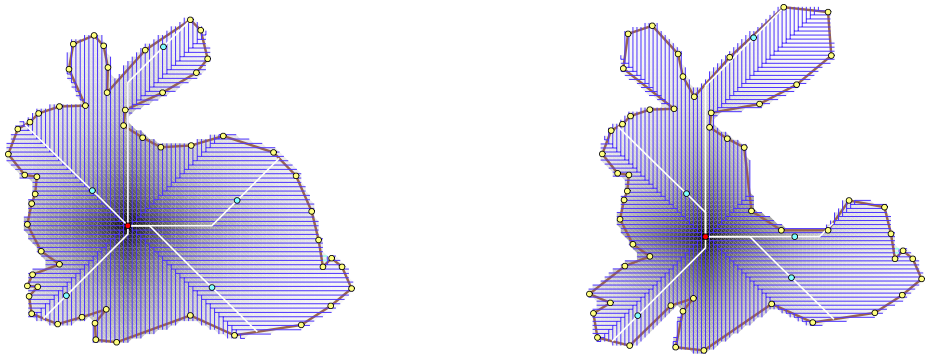


Figure 6.4: The cousin tree, points, and integral lines for a bunny shape undergoing deformation, at 1x and 2x resolution.

Expansion edges are edges where, when tracing integral lines downhill, multiple integral lines converge at an edge. More insidiously, the gradient causes integral lines on either side of the edge to diverge, leading to an area whose interior cannot be reached by any integral line from the boundary (Figure 6.12, right). Expansion edges are visualized in our figures whenever they occur as cyan edges. When tracing an integral line, if such an edge is reached, it is ambiguous as to which triangle / gradient to follow.

This is in contrast to compression edges, which are edges upon ascent where the integral lines converge. In our setting, we have a single critical point, a maximum, and all integral lines must converge towards it, resulting in inevitable compression. Expansion edges rarely occur for our function value assignment, except near small concavities on the exterior of

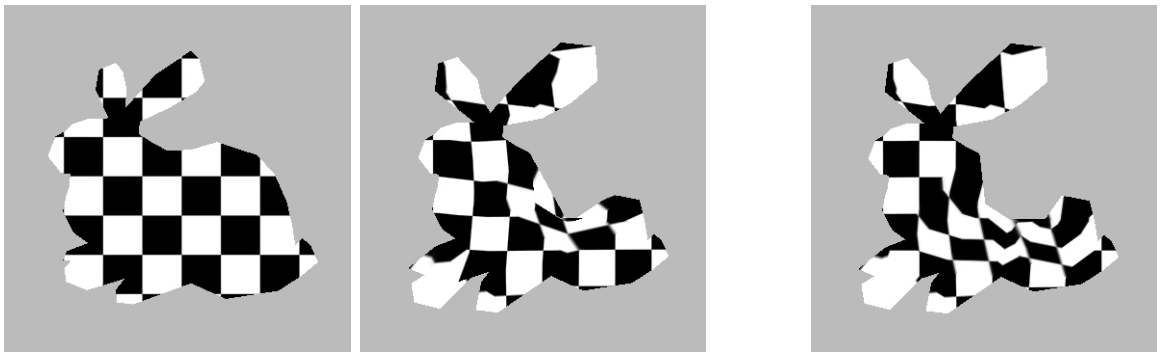


Figure 6.5: A checkerboard pattern applied to the undeformed bunny shape (left) and its deformation (center) according to the cousin tree, points, and integral lines from Figure 6.4. At right, a deformation computed using the same boundary and interior points, but with a different maximum location.

Figure 6.6: Converting from Integral Curve Coordinates after applying 5°, 45°, and 90° rotations to the undeformed bunny in Figure 6.5, left. If our discrete algorithm were perfectly stable, the images would be indistinguishable.

the boundary. To remove expansion edges, the granularity of the grid could be increased so that the distance between any concave boundary faces of the figure is greater than the size of the unit grid.

6.5 Preliminary Examinations

Preliminary research into creating C^1 continuous bicubic bezier patches with continuity constraints shows they do create smoother integral lines. Numerical integration is inevitable, however, due to integral lines being defined as a non-separable non-linear first order differentiable system of equations. In our preliminary investigations, numerical issues arise,

Table 6.1: Performance Times for the Tripus

Resolution	Boundary Points	Internal Points	Grid Points	Function	Time (s)
1x	54	11	673	Maxima	0.00
				BFS	0.02
				Gradient	0.24
				Other	0.09
				Total	0.14
2x	54	11	2306	Maxima	0.01
				BFS	0.12
				Gradient	0.74
				Other	0.24
				Total	1.1
5x	54	11	11584	Maxima	0.00
				BFS	2.53
				Gradient	3.85
				Other	1.5
				Total	7.88

and extreme care must be taken to ensure a monotonic interpolation. Several approaches to monotonic quadratic splines interpolation appear promising [49, 50]. If patches were created so that they were C^1 or G^1 continuous across edges, expansion edges (and compression edges) would no longer occur. A looser requirement would even be that the gradients on either side of an edge cannot point towards or away from each other and yet remain discontinuous; a tangible solution for this relaxed condition is less clear.

We also experimented with harmonic and biharmonic functions, as well as harmonic and biharmonic functions constrained to lie within the cousin tree's function space. Unfortunately, harmonic and biharmonic functions created without cousin tree constraints contain a greater number of expansion edges (Figure 6.13), possibly due to the boundary $= 0$ constraints. In the absence of cousin tree constraints, biharmonic functions may not be monotonic, and harmonic functions may contain spurious saddles in higher dimensions.

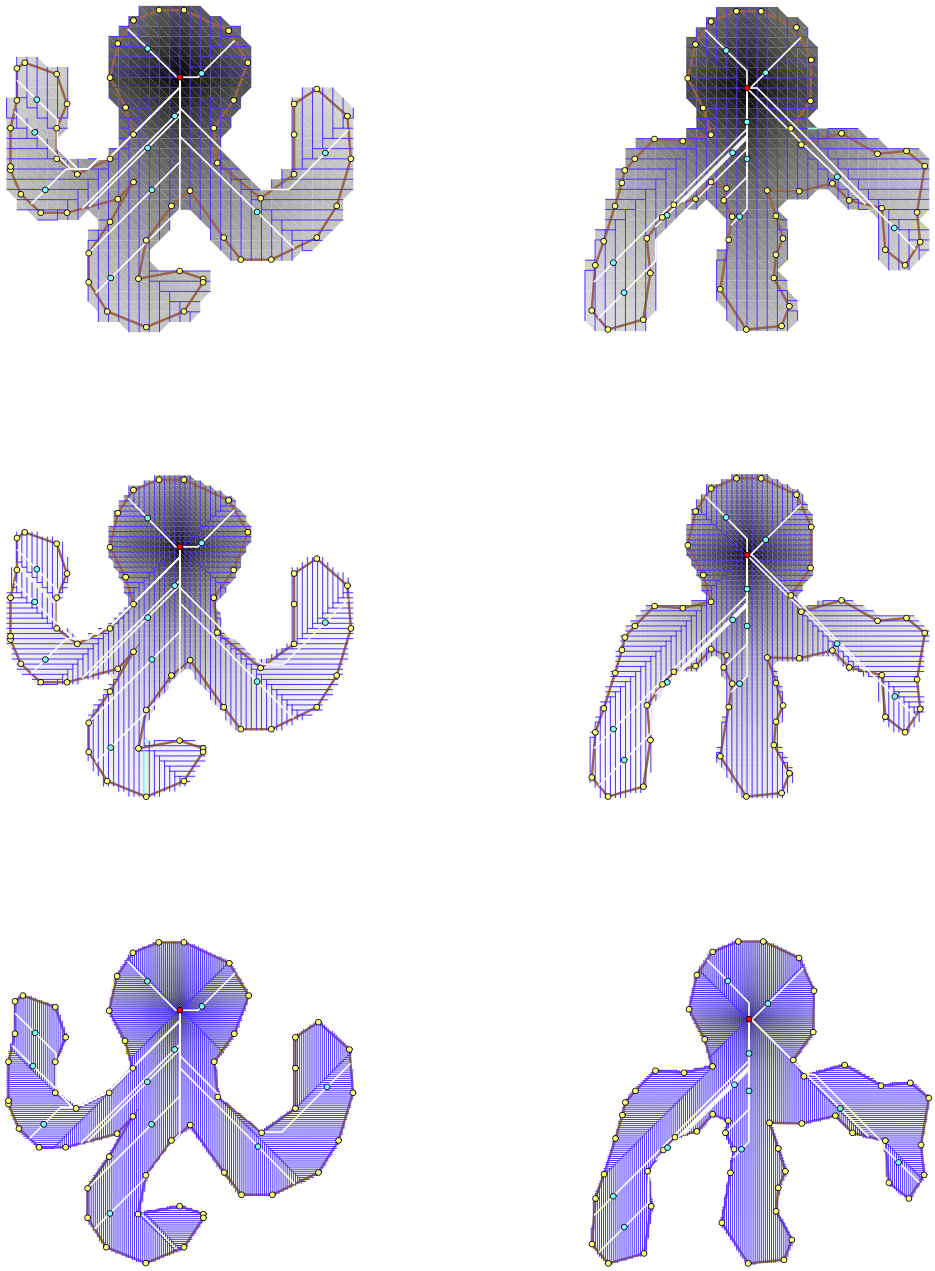


Figure 6.7: The cousin tree, points, and integral lines for a tripus shape undergoing deformation, at 1x, 2x, and 5x resolution.

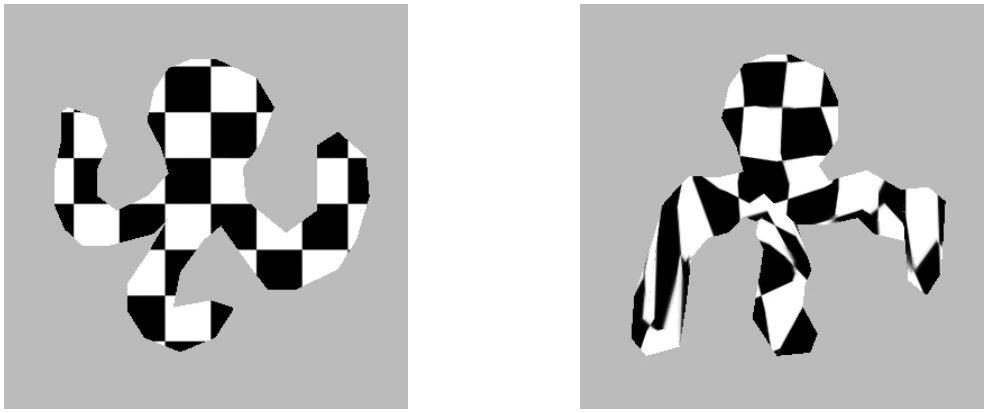


Figure 6.8: A checkerboard pattern applied to the undeformed tripus shape (left) and its deformation (right).

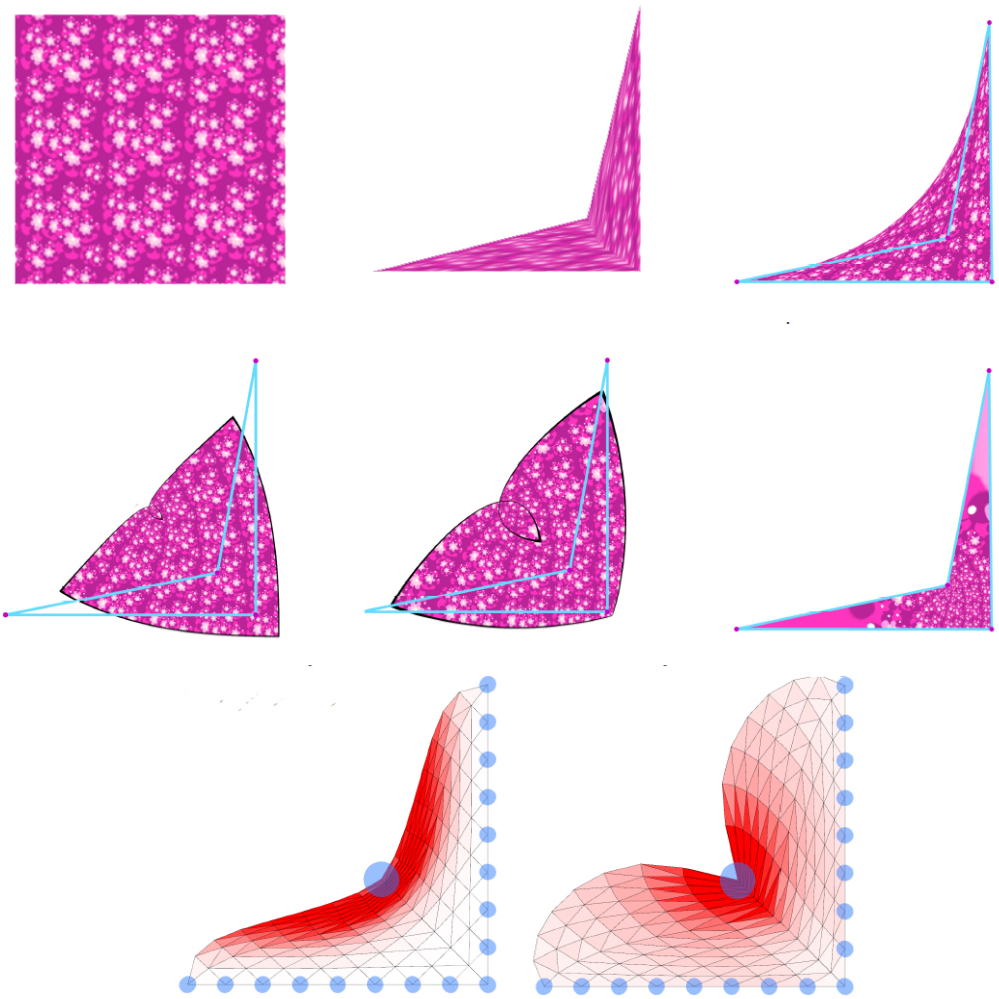


Figure 6.9: Deforming a square comparison
From left to right, top to bottom: the undeformed square shape, Integral Curve Coordinates with a single interior point in the center, Harmonic coordinates [48], Cauchy coordinates, Szego coordinates [5], Controllable Conformal Maps [26], Locally Injective Mapping using Laplacian energy, Locally Injective Mapping using Dirichlet energy [7].

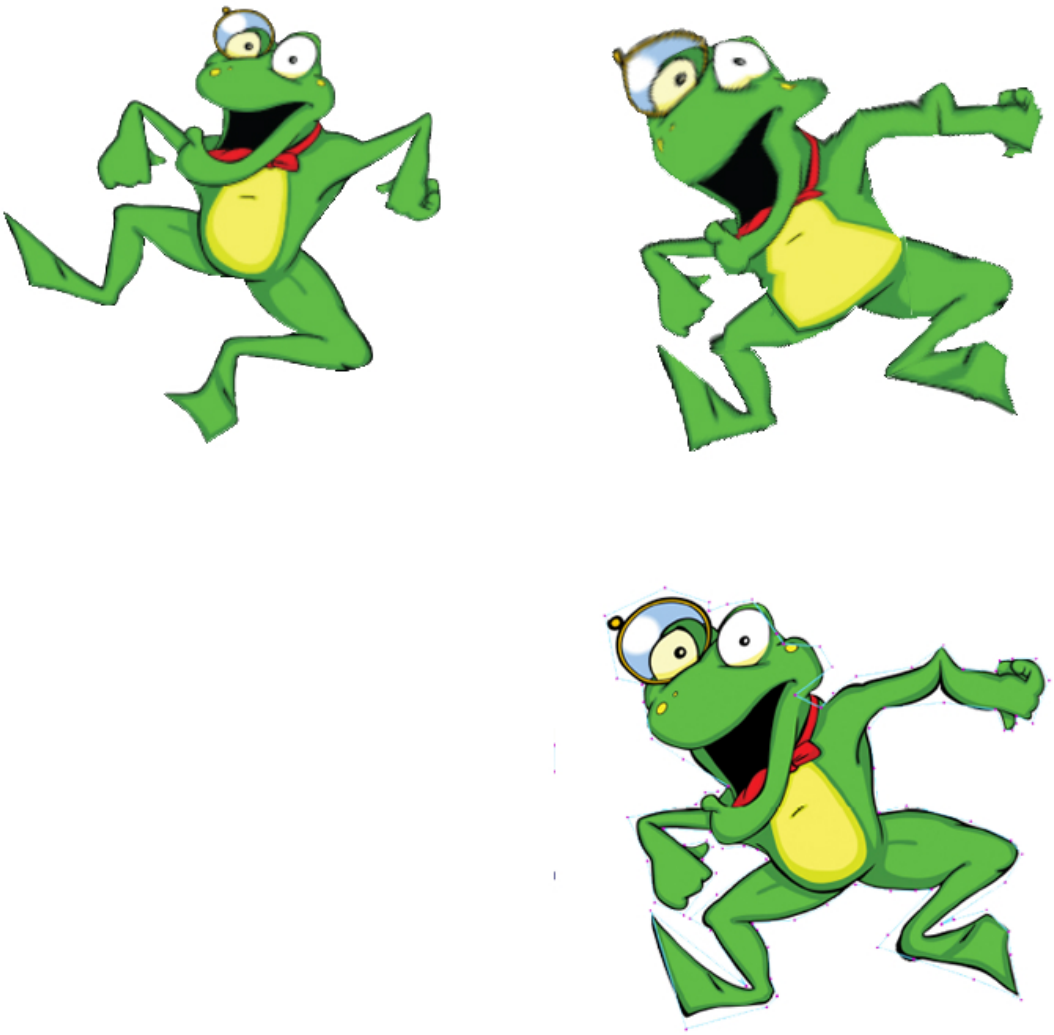


Figure 6.10: Comparison to Complex Barycentric Coordinates
The original figure (upper-left), a deformation computed using Integral Curve Coordinates (upper-right), and a deformation using Complex Barycentric Coordinates [5] (lower-right).

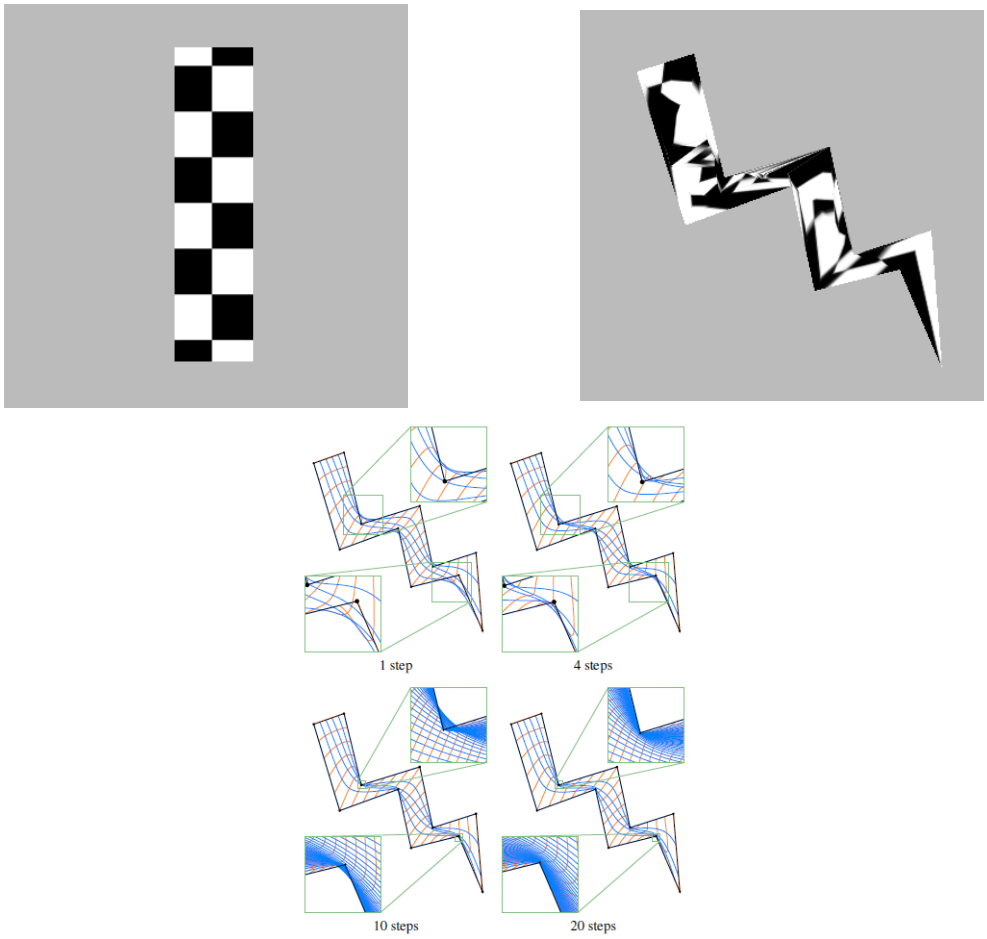


Figure 6.11: Comparison to Composite Mean Value Mapping
From left to right: original figure, deformation using Integral Curve Coordinates, deformation using Composite Mean Value Mapping [8].

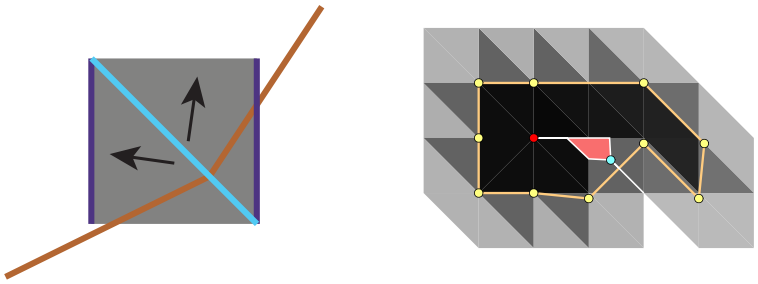


Figure 6.12: Area not covered by integral lines due to an expansion edge.

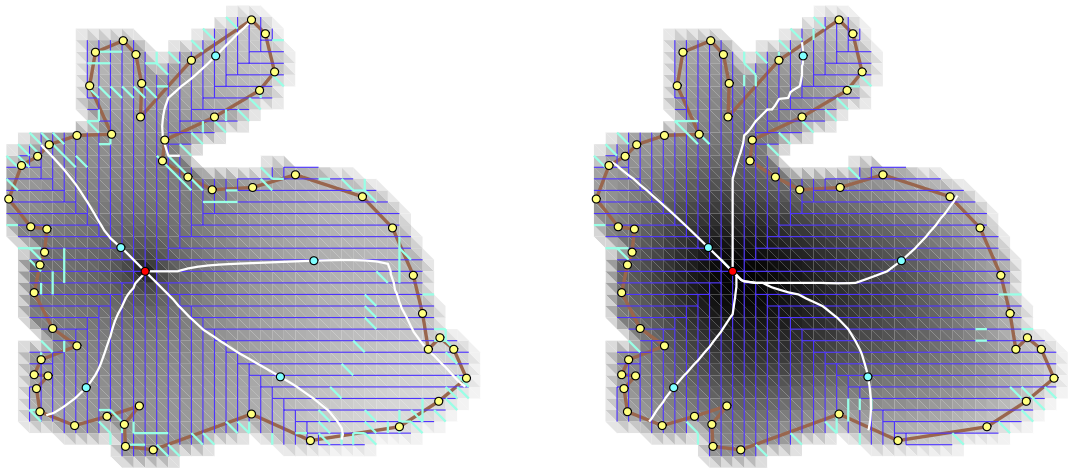


Figure 6.13: Solving the Laplace (left) and bi-Laplace (right) equation without cousin tree constraints.

Chapter 7: Discussion and Conclusion

We have described a new technique to address the problem of bijective shape deformation based on Integral Curve Coordinates. Our current piecewise linear implementation, while preventing inverted elements ($det(M) < 0$ in Equation 2.2), does not prevent collapsed elements ($det(M) = 0$). We would like to address this in the future with C^1 or G^1 monotonic interpolation of functions values or by simulating the infinitessimal separation of integral curves [51] and then perturbing the resulting deformation to correct collapsed elements.

Our approach is restricted to cage-based or boundary deformations. In the future, we would like to extend our approach to these other control structures, such as points and bone skeletons, which are intuitive to manipulate and can have far fewer vertices than a cage. One approach may be to trage integral lines of smoothed distance functions from the control geometry [52], without arc-length reparameterization.

Our grid discretization of the boundary or cage may lead to problematic boundary gradients near sharp angles ($< 45°$). One solution would be to adjust the grid spacing non-uniformly (a so-called "plaid deformation") such that (a) grid vertices are positioned exactly at boundary vertices and (b) sharp angles are non-uniformly scaled and eliminated from the point of view of the grid cell.

In the future, we would like to explore modifications to the constraints computed by the cousin tree. The constraints we compute, while correct, are not the only correct constraints. Thus, we envisage an iterative procedure that updates the constraints and re-runs the function computation portion of our algorithm so as to generate fairer deformations. Jacobson [16] explored such an iterative constraint modification scheme in an analogous setting to good effect. A similar iterative scheme may also remove expansion edges.

Finally, we would like to implement our approach in higher dimensions, applying it to problems such as the animation of volumetric models. As it stands, this work serves as a discrete, 2D implementation of Integral Curve Coordinates for bijective shape deformation.

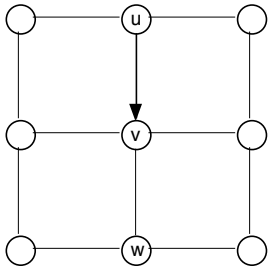
Appendix A: Proof of Correctness

In the following, the domain is a regular n-dimensional grid with edges parallel to the $x, y, z, \ldots$ axes. We call a subdomain *ball-like* if the area enclosed by its cells is locally path-connected even with the addition of imaginary diagonal grid edges in the axis-aligned planes.

Theorem 2. *Let D be a subdomain that is ball-like. Let T be an associated cousin tree. Assume all nodes on the boundary of the subdomain have value greater than all grid neighbors outside the ball (including along imaginary diagonal grid edges). Then by rooting T at the maximum and assigning monotonically decreasing (unique) values to internal nodes of T along the path from the root (preserving the values at the root and the leaf), there can be no grid maximum, minimum, or saddles at any tree node, even if the node has diagonal grid edges.*

Lemma 1. *A node v of cousin tree T on subdomain D with parent in the $+x$ grid direction ($-x, +y, -y, +z, -z, \ldots$ respectively) has a child in the $-x$ grid direction ($+x, -y, +y, -z, +z, \ldots$ resp.), unless the neighbor in the $-x$ grid direction ($+x, -y, +y, -z, +z, \ldots$ resp.) is outside D.*

Proof of Lemma 1. Without loss of generality, assume node v has its tree parent u in the $+y$ grid direction. Let w be the node in the $-y$ grid direction. Assume v is not w's tree parent.



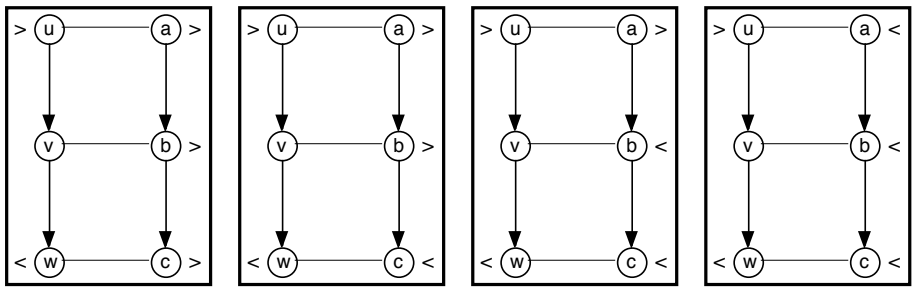
The cousin rule tells us that v's tree parent u and w's tree parent must be grid neighbors. Yet the only grid neighbor of w within 1 grid edge of v's parent of u is v. Thus we have reached a contradiction, and v must be w's parent. □

Lemma 2. *For any non-boundary tree node v, its grid neighbors (even with the addition of diagonal edges in the $x, y, z, \ldots$ planes) with greater value all belong to the same connected component (partitioned according to greater/less than v's value) and its grid neighbors (even with the addition of diagonal edges in the $x, y, z, \ldots$ planes) with lesser value all belong to a second connected component.*

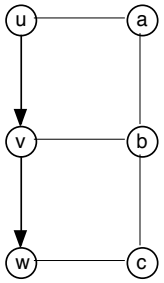
Proof of Lemma 2. Without loss of generality, assume v has parent u in the $+y$ direction. We now consider two cases, v is on the boundary of D and v is not on the boundary of D.

Case v is not on the boundary of D.

It follows from Lemma 1 that v has a child w in the $-y$ direction. As paths are monotonic, $Value(u) > Value(v) > Value(w)$. Since we wish to prove that v will have exactly two connected components partitioned according to greater/less than v's value, we can restrict our examination to grid neighbors of u, v, w in the $+x$ direction, again without loss of generality; the greater value connected component will have to include u and the lesser value connected component will have to include w. We now consider two subcases, v is not the tree parent of its x-direction grid neighbor b and v is the tree parent of b. To clarify, when determining connected components, neighbors along diagonal edges may be considered; however, tree edges are always grid edges, and neighbors in any sense except when computing connected components are only along grid edges.

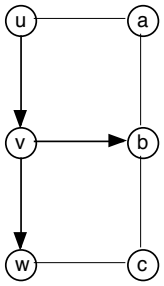


Because of the cousin rule, b's parent must be a, since a is the only grid neighbor of b within 1 grid edge of v's parent of v. Lemma 1 tells us that c must be the tree child of b. Due to the monotonicity of values along tree paths, $Value(u) > Value(v) > Value(w)$ and $Value(a) > Value(b) > Value(c)$. The following diagram depicts all possible greater/less than relationships between a, b, c and v.



As we can see, in all possibilities, there are exactly two connected components partitioned according to greater/less than v's value. The component with values greater obviously contains v's tree parent u, and the component with values lesser obviously contains v's tree child w.

Subcase: v is the tree parent of b.

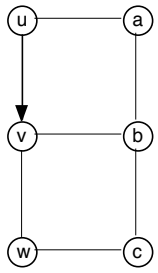


The cousin tree imposes the following restrictions on a and c. a cannot be the tree child of b as u and a would not be tree cousins. Since b cannot be a's tree parent or its tree child, b and a must be tree cousins. Therefore a's tree parent must be a grid neighbor of v. The only possibility is u. Since c cannot be the tree parent of b (resp. w), it must be the tree child or cousin of b (resp. w). Therefore c must be the tree child of one and

the tree cousin of the other. In either case, the monotonicity of values along tree paths implies $Value(v) > Value(c)$ as well as $Value(v) > Value(b)$, $Value(v) > Value(w)$, and $Value(u) > Value(v)$. The value of a may be greater than or less than v, but in both cases there are exactly two connected components (partitioned according to greater/less than v's value). As in the earlier subcase, the component with values greater obviously contains v's tree parent u, and the component with values lesser obviously contains v's tree child w.

Case v is on the boundary of D

If v's grid neighbor w in the $-y$ direction is in D, then Lemma 1 applies and v must be the tree parent of w and hence $Value(v) > Value(w)$. Otherwise, w is not in D, but by assumption $Value(v) > Value(w)$. Thus v's neighbor in the $+y$ direction has value greater than v, and v's neighbor in the $-y$ direction has value less than v. Following the same argument as in Case v is not on the boundary of D, we can again restrict our examination to grid neighbors of u, v, w in the $+x$ direction (without loss of generality).



There are 33 valid cousin tree configurations among the 2^4 possible boundary conditions given a, b, c, w can each be outside D. They are presented in Figure A.1.

In every case there are exactly two connected components partitioned according to greater/less than v's value. The component with values greater obviously contains v's tree parent u, and the component with values lesser obviously w. □

Proof of Theorem 2. Lemma 2 tells us that any internal tree node v has one connected component of nodes with value greater than v's and another connected component with

values lesser (0 "folds"). It follows directly then that v cannot be a minimum, maximum, or saddle. □

Theorem 3. *A breadth-first search (BFS) in a ball-like domain D where at each BFS generation all x then all y then all z ... grid edges are explored in order constructs a cousin tree.*

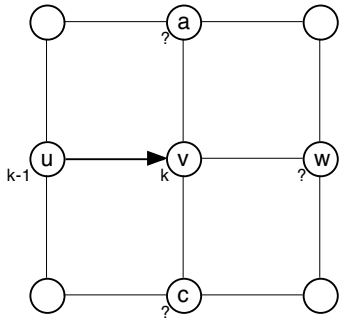
Proof of Theorem 3. By induction. Let n represent the number of BFS generations of growth from the root node. After step n, nodes reached at BFS generation $n-1$ have all of their grid neighbors in the BFS tree or outside D; grid neighbors in the BFS tree will be shown to satisfy the cousin tree definition.

After step $n = 1$, the root node has become the parent of all grid neighbors. The root node satisfies the cousin tree constraints by being the BFS parent of all grid neighbors.

Assume true for $n = k$. Nodes reached at BFS generation $k-1$ have all grid neighbors also in the BFS tree with cousin tree definition satisfied or outside D for BFS generation $\leq k-1$ nodes.

We wish to show that after step $n = k+1$ all nodes reached at BFS generation k now also have all their grid neighbors in the BFS tree satisfying the cousin tree definition or outside D.

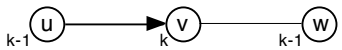
Consider a grid node v reached at BFS generation k. Let u be the BFS parent of v. u was thus reached at BFS generation $k-1$. Consider the following arbitrary axis-aligned figure of v:



After step $n = k + 1$ nodes a, c, w are also in the BFS tree. We aim to show that a, c, w are either the BFS children of v, tree cousins of v in the BFS tree, or outside D.

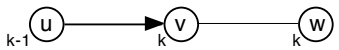
If w (or a, c) is outside D, then it is of no concern to us. If w (or a, c) is inside D, then it must have been reached at step $k - 1$, k, or $k + 1$. w (or a, c) cannot have been reached before step $k - 1$ since it would have been able to reach v at step $k - 1$. w (or a, c) must be reached before step $k + 2$ since v can reach it at step $k + 1$.

Suppose node w was reached at generation $k - 1$.



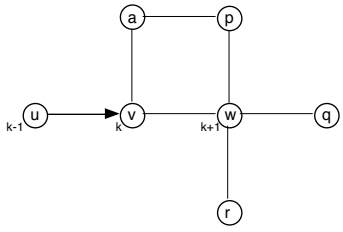
The inductive hypothesis tells us that w has cousin tree relationships to all its grid neighbors, including v. Yet w is not the BFS parent, BFS child, or BFS cousin of v (since the BFS parent of w cannot be a grid neighbor of u). This contradicts w having been reached at BFS generation $k - 1$.

Suppose node w was reached at generation k.



This too is impossible since v, w are neighbors and cannot have the same taxicab distance to the BFS root in our ball-like domain D.

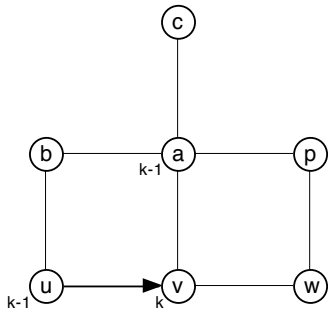
This leaves us with w reached at generation $k + 1$.



We first consider p as the BFS parent of w. Then p must have been reached at BFS generation k. Furthermore, if p, v were both reached at BFS generation k and w was reached at BFS generation $k+1$, then a must be in D and have generation $k-1$, since D is ball-like

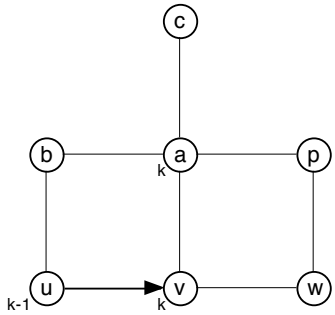
and distance is taxicab. But p as the BFS parent of w contradicts the BFS $x/y/z/\ldots$ growth ordering implied by u as the BFS parent of v, since u is the BFS parent of v instead of a. The same argument prevents r as the BFS parent of w. Consider q as the BFS parent of w. Then q and v were reached at the same BFS generation while w between them was reached at a later BFS generation. This is impossible in our ball-like domain D since BFS generations correspond to minimal taxicab distance. The only remaining possibility is v as the BFS parent of w. This is a valid cousin tree relationship.

Suppose node a was reached at generation $k-1$.



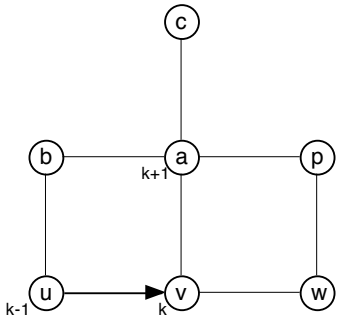
The inductive hypothesis tells us that a has cousin tree relationships to all its grid neighbors, including v.

Suppose node a was reached at generation k.

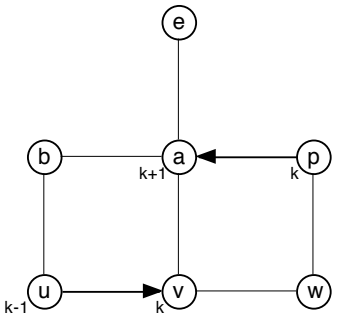


This too is impossible since v, a are neighbors and cannot have the same taxicab distance to the BFS root in our ball-like domain D.

Suppose node a was reached at generation $k+1$.



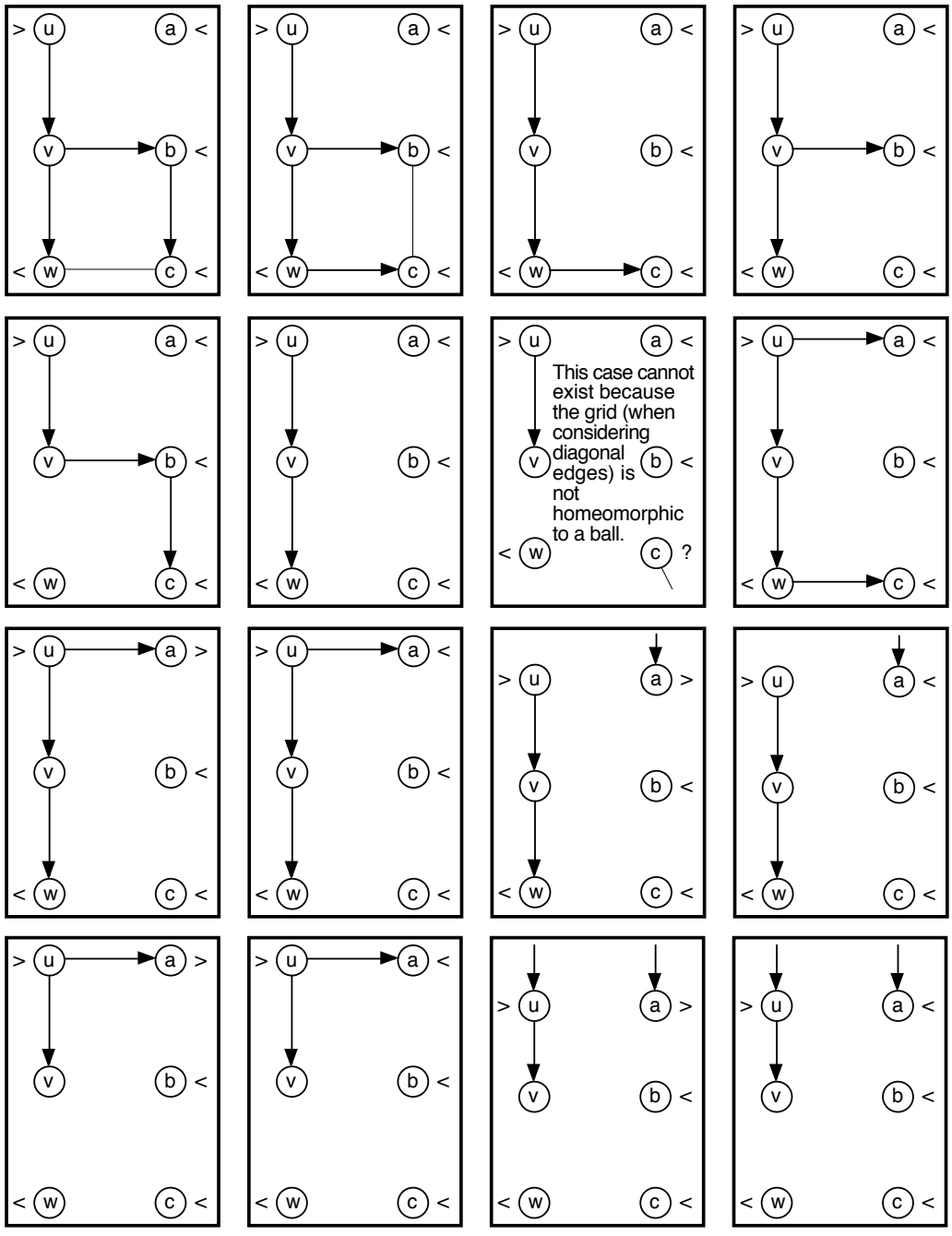
If node p is the BFS parent of a, then p was reached at BFS generation k. We then have the following diagram.

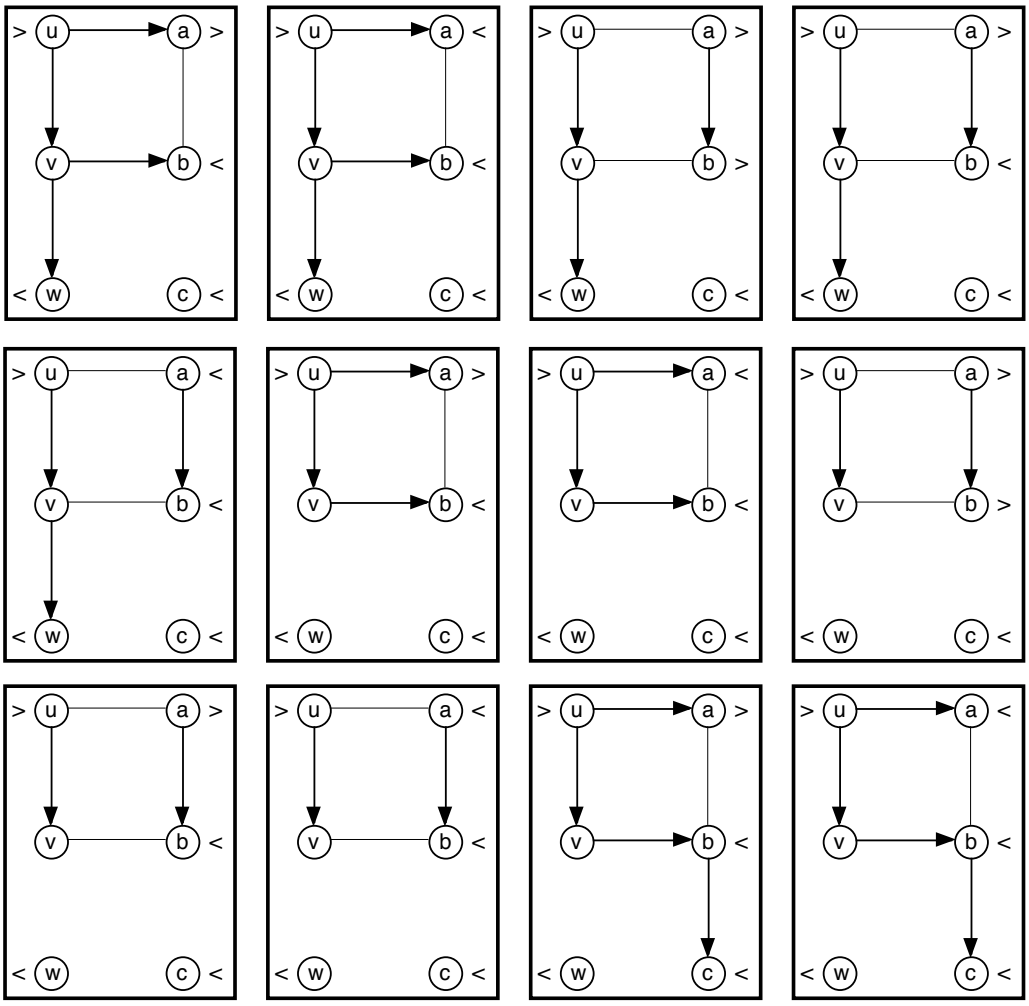


But nodes v and p cannot have the same taxicab distance (as evidenced by their BFS generations) to the BFS root in ball-like D if nodes a and u differ in taxicab distance to the BFS root by 2. The same argument prevents e from being the BFS parent of a. The only remaining possibilities are v as the BFS parent of a and b as the BFS parent of a, both of which have valid cousin tree relationships with v.

The analysis of the relationship between node v and node c follows exactly the same argument as between nodes v and a.

□


This case cannot exist because the grid (when considering diagonal edges) is not homeomorphic to a ball.
?


> u a >
v b <
< w c <
> u a <
v b <
< w c <
> u a >
v b >
< w c <
> u a >
v b <
< w c <
> u a <
v b <
< w c <
> u a >
v b <
< w c <
> u a <
v b <
< w c <
> u a >
v b >
< w c <
> u a >
v b <
< w c <
> u a <
v b <
< w c <
> u a >
v b <
< w c <
> u a <
v b <
< w c <

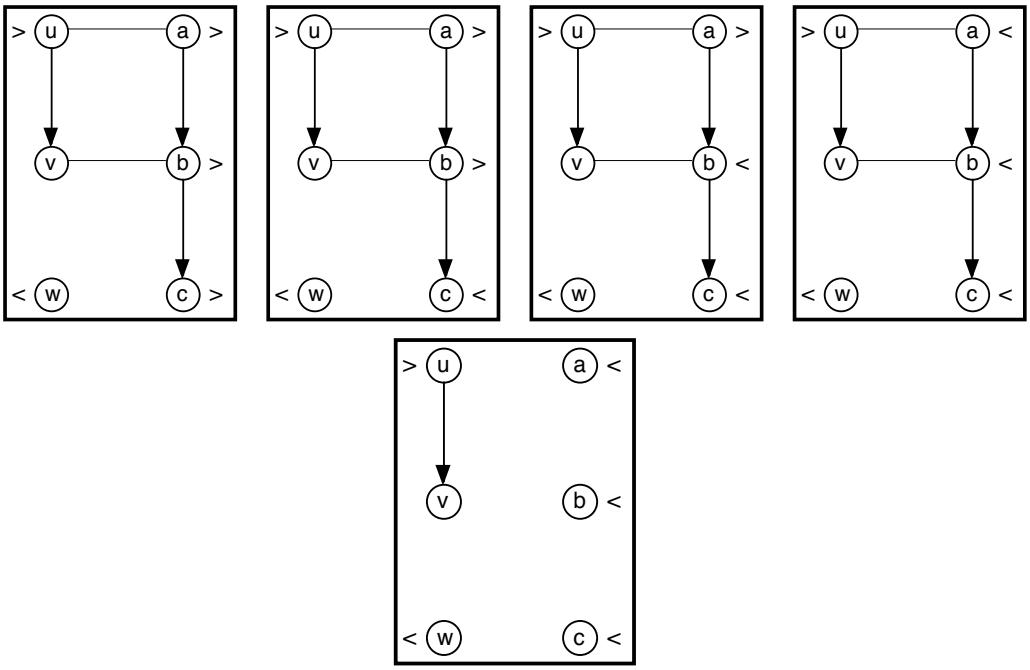


Figure A.1: Cousin tree configurations for a node on the boundary: a outside (2 cases); a, b outside; a, c outside; a, w outside; a, b, c outside; a, b, w outside (violates assumption); b outside; b, c outside (4 cases); b, c, w outside (4 cases); c outside (5 cases); c, w outside (5 cases); w outside (6 cases); a, b, c, w outside.

Bibliography

Bibliography

[1] T. W. Sederberg and S. R. Parry, "Free-form deformation of solid geometric models," in *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '86. New York, NY, USA: ACM, 1986, pp. 151–160. [Online]. Available: http://doi.acm.org/10.1145/15922.15903

[2] M. Alexa, D. Cohen-Or, and D. Levin, "As-rigid-as-possible shape interpolation," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 157–164. [Online]. Available: http://dx.doi.org/10.1145/344779.344859

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

[4] B. Allen, B. Curless, and Z. Popović, "The space of human body shapes: reconstruction and parameterization from range scans," *ACM Transactions on Graphics*, pp. 587–594, 2003.

[5] O. Weber, M. Ben-Chen, and C. Gotsman, "Complex barycentric coordinates with applications to planar shape deformation," in *Computer Graphics Forum*, vol. 28, no. 2. Wiley Online Library, 2009, pp. 587–597.

[6] Y. Xu, R. Chen, C. Gotsman, and L. Liu, "Embedding a triangular graph within a given boundary," *Comput. Aided Geom. Des.*, vol. 28, no. 6, pp. 349–356, Aug. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.cagd.2011.07.001

[7] C. Schüller, L. Kavan, D. Panozzo, and O. Sorkine-Hornung, "Locally injective mappings," in *Computer Graphics Forum*, vol. 32, no. 5. Wiley Online Library, 2013, pp. 125–135.

[8] T. Schneider, K. Hormann, and M. S. Floater, "Bijective composite mean value mappings," in *Computer Graphics Forum*, vol. 32, no. 5. Wiley Online Library, 2013, pp. 137–146.

[9] N. Aigerman and Y. Lipman, "Injective and bounded distortion mappings in 3d," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 106, 2013.

[10] E. L. Wachspress, *A rational finite element basis*. Academic Press New York, 1975, vol. 114.

[11] M. S. Floater, "Mean value coordinates," *Computer Aided Geometric Design*, vol. 20, no. 1, pp. 19–27, 2003.

[12] T. Ju, S. Schaefer, and J. Warren, "Mean value coordinates for closed triangular meshes," in *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3. ACM, 2005, pp. 561–566.

[13] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki, "Harmonic coordinates for character articulation," in *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3. ACM, 2007, p. 71.

[14] K. Hormann and N. Sukumar, "Maximum entropy coordinates for arbitrary polytopes," in *Computer Graphics Forum*, vol. 27, no. 5. Wiley Online Library, 2008, pp. 1513–1520.

[15] J. Manson and S. Schaefer, "Moving least squares coordinates," in *Computer Graphics Forum*, vol. 29, no. 5. Wiley Online Library, 2010, pp. 1517–1524.

[16] A. Jacobson, "Algorithms and interfaces for real-time deformation of 2d and 3d shapes," Ph.D. dissertation, Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 21189, 2013.

[17] J. W. Milnor, *Morse theory*. Princeton university press, 1963, no. 51.

[18] R. Forman, "A users guide to discrete morse theory," *Sém. Lothar. Combin*, vol. 48, p. B48c, 2002.

[19] H. Edelsbrunner, J. Harer, and A. Zomorodian, "Hierarchical morse complexes for piecewise linear 2-manifolds," in *Proceedings of the seventeenth annual symposium on Computational geometry*. ACM, 2001, pp. 70–79.

[20] A. J. Zomorodian, *Computing and comprehending topology: Persistence and hierarchical morse complexes*. Citeseer, 2001.

[21] R. MacCracken and K. I. Joy, "Free-form deformations with lattices of arbitrary topology," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96. New York, NY, USA: ACM, 1996, pp. 181–188. [Online]. Available: http://doi.acm.org/10.1145/237170.237247

[22] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann, "Joint-dependent local deformations for hand animation and object grasping," in *Proceedings of Graphics Interface*, 1988, pp. 26–33. [Online]. Available: http://dl.acm.org/citation.cfm?id=102313.102317

[23] T. Igarashi, T. Moscovich, and J. F. Hughes, "As-rigid-as-possible shape manipulation," in *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3. ACM, 2005, pp. 1134–1141.

[24] Y. Lipman, D. Levin, and D. Cohen-Or, "Green coordinates," in *ACM Transactions on Graphics (TOG)*, vol. 27, no. 3. ACM, 2008, p. 78.

[25] M. Ben-Chen, O. Weber, and C. Gotsman, "Variational harmonic maps for space deformation," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3, pp. 34:1–34:11, 2009.

[26] O. Weber and C. Gotsman, "Controllable conformal maps for shape deformation and interpolation," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4, p. 78, 2010.

[27] J. R. Nieto and A. Susín, "Cage based deformations: a survey," in *Deformation Models*. Springer, 2013, pp. 75–99.

[28] R. W. Sumner, J. Schmid, and M. Pauly, "Embedded deformation for shape manipulation," in *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3. ACM, 2007.

[29] A. Jacobson, I. Baran, J. Popović, and O. Sorkine, "Bounded biharmonic weights for real-time deformation," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 78:1–78:8, Jul. 2011. [Online]. Available: http://doi.acm.org/10.1145/2010324.1964973

[30] D. L. James and D. K. Pai, "Artdefo: accurate real time deformable objects," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1999, pp. 65–72.

[31] F. L. Bookstein, "Principal warps: Thin-plate splines and the decomposition of deformations," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 11, no. 6, pp. 567–585, 1989.

[32] A. Belyaev, "On transfinite barycentric coordinates," in *Proceedings of the fourth Eurographics symposium on Geometry processing*. Eurographics Association, 2006, pp. 89–99.

[33] J. Warren, S. Schaefer, A. N. Hirani, and M. Desbrun, "Barycentric coordinates for convex sets," *Advances in computational mathematics*, vol. 27, no. 3, pp. 319–338, 2007.

[34] O. Weber, M. Ben-Chen, C. Gotsman, and K. Hormann, "A complex view of barycentric mappings," in *Computer Graphics Forum*, vol. 30, no. 5. Wiley Online Library, 2011, pp. 1533–1542.

[35] V. Kraevoy, A. Sheffer, and C. Gotsman, "Matchmaker: constructing constrained texture maps," in *ACM Transactions on Graphics (TOG)*, vol. 22, no. 3. ACM, 2003, pp. 326–333.

[36] H. Seo and F. Cordier, "Constrained texture mapping using image warping," in *Computer Graphics Forum*, vol. 29, no. 1. Wiley Online Library, 2010, pp. 160–174.

[37] T.-Y. Lee, S.-W. Yen, and I.-C. Yeh, "Texture mapping with hard constraints using warping scheme," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 14, no. 2, pp. 382–395, 2008.

[38] Y. Lipman, "Bounded distortion mapping spaces for triangular meshes," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 108, 2012.

[39] T. Weinkauf, Y. Gingold, and O. Sorkine, "Topology-based smoothing of 2D scalar fields with C1-continuity," *Computer Graphics Forum (proceedings of EuroVis)*, vol. 29, no. 3, pp. 1221–1230, 2010. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2009.01702.x

[40] A. Jacobson, T. Weinkauf, and O. Sorkine, "Smooth shape-aware functions with controlled extrema," *Computer Graphics Forum (Proc. SGP)*, vol. 31, no. 5, pp. 1577–1586, July 2012. [Online]. Available: http://tinoweinkauf.net/publications/absjacobson12a.html

[41] A. Zomorodian, "Computing and comprehending topology: Persistence and hierarchical Morse complexes," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2001.

[42] H. Blum, "A transformation for extracting new descriptors of shape," in *Models for the Perception of Speech and Visual Form. Proceedings of a Symposium*, W. Wathen-Dunn, Ed. Cambridge MA: MIT Press, Nov 1967, pp. 362–380.

[43] L. Liu, E. W. Chambers, D. Letscher, and T. Ju, "Extended grassfire transform on medial axes of 2d shapes," *Comput. Aided Des.*, vol. 43, no. 11, pp. 1496–1505, Nov. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.cad.2011.09.002

[44] H. Edelsbrunner and E. P. Mücke, "Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms," *ACM Transactions on Graphics (TOG)*, vol. 9, no. 1, pp. 66–104, 1990.

[45] C.-K. Yap, "A geometric consistency theorem for a symbolic perturbation scheme," *J. Comput. Systems Sci.*, vol. 40, pp. 2–18, 1990.

[46] ——, "Symbolic treatment of geometric degeneracies," *J. Symbolic Comput.*, vol. 10, pp. 349–370, 1990.

[47] J. R. Shewchuk, "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator," in *Applied Computational Geometry: Towards Geometric Engineering*, ser. Lecture Notes in Computer Science, M. C. Lin and D. Manocha, Eds. Springer-Verlag, May 1996, vol. 1148, pp. 203–222, from the First ACM Workshop on Applied Computational Geometry.

[48] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki, "Harmonic coordinates for character articulation," *ACM Trans. Graph.*, vol. 26, no. 3, Jul. 2007. [Online]. Available: http://doi.acm.org/10.1145/1276377.1276466

[49] M. S. Floater and J. M. Peña, "Tensor-product monotonicity preservation," *Advances in Computational Mathematics*, vol. 9, no. 3-4, pp. 353–362, 1998.

[50] R. Beatson and Z. Ziegler, "Monotonicity preserving surface interpolation," *SIAM journal on numerical analysis*, vol. 22, no. 2, pp. 401–411, 1985.

[51] H. Edelsbrunner, J. Harer, and A. Zomorodian, "Hierarchical Morse complexes for piecewise linear 2-manifolds," in *Proc. 17th Ann. ACM Sympos. Comput. Geom.*, 2001, pp. 70–79.

[52] J. Peng, D. Kristjansson, and D. Zorin, "Interactive modeling of topologically complex geometric detail," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 635–643, 2004.

Curriculum Vitae

EDUCATION:

B.S. Bachelor of Science, May 2012

Major: Computer Science

Minor: Mathematics

George Mason University, Fairfax, VA

EXPERIENCE WITH:

Adobe InDesign, Illustrator, Photoshop

HTML and CSS web pages

Java, Javascript, C, C++, Python, Lisp and Oracle/SQL

WORK EXPERIENCE:

Software Engineer (June 2012-Present)
Software Engineering Intern (Summer, Winter 2010-2012)
Metron Aviation, Dulles, VA

Student Research Assistant (September 2012-May 2013)
George Mason University, Fairfax, VA

MASC Member (February 2011-May 2011)
Motion and Shape Computing Group, George Mason University, Fairfax, VA

Resident Technician (August 2009-May 2011)
Information Technology Unit, George Mason University, Fairfax, VA

College Student Tech Specialist (March 2010-June 2010)
Lockheed Martin, Manassas, VA