Hybrid Methods for Interactive Shape Manipulation

Ofir Weber

Hybrid Methods for Interactive Shape Manipulation

Research Thesis

In Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

Ofir Weber

Submitted to the Senate of the
Technion - Israel Institute of Technology

Av 5770 Haifa July 2010

This Research Thesis was done under the supervision of Prof. Craig Gotsman in the faculty of Computer Science.

The generous financial help of the Technion is gratefully acknowledged.

Contents

List of Figures

Abstract

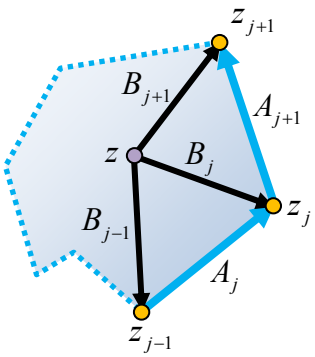
Manipulating 2D and 3D shapes interactively is an important task in computer graphics. The challenge is to be able to induce a global change to the shape while preserving its local structure. A deformation tool accepts as an input a source shape as well as some user specified constraints that can be manipulated interactively. The amount of constraints provided by the user should be kept to minimum in order to make the tool intuitive to control. The required output is a shape that satisfies the imposed constraints, yet strives to preserve the character and the fine geometric details of the original shape.

In this work, we explore several techniques to achieve detail preserving shape deformation. We first provide an algorithm that combines an intrinsic representation of a surface (using differential coordinates) with a data-driven approach. The realism of the deformation is increased by incorporating example shapes that put the deformation into context, demonstrating characteristic deformations of the shape, such as bulging of muscles and appearance of folds for human or animal shapes.

We then turn over to a different approach which is fundamentally a space deformation technique. Space deformation deforms the ambient space rather than directly deforming the object and any object that is embedded in that space deforms accordingly as a byproduct. The main advantage of space deformation is that it is not limited to a particular geometric representation such as triangle meshes. A popular way to perform space deformation is to use barycentric coordinates, however, deformation with barycentric coordinates essentially destroys fine details of the shape. We extend the notion of barycentric coordinates in two dimensions to complex numbers. This generalization results in a hybrid approach that provides us the ability to obtain shape preservation with a space deformation framework.

Notations

Symbol	Meaning
L	The 3D Laplace Beltrami operator applied on a 2-manifold
div	The divergence operator: $\frac{\partial}{\partial x}+\frac{\partial}{\partial y}+\frac{\partial}{\partial z}$
Ω	A simply connected closed domain in R^2
$\partial\Omega$	The boundary of a simply connected domain in R^2
z	An internal point
z_j	The j^{th} vertex of the source polygon cage represented as a complex number
$B_j(z)$	$z_j - z$
A_j	$z_j - z_{j-1}$
$g(z)$	The Cauchy transform
$\mathcal{H}(z)$	The Hilbert transform
$C_j(z)$	The j^{th} Cauchy coordinate basis function evaluated at a point z
$\varphi_j(z)$	The real part of the j^{th} Cauchy coordinate basis function evaluated at a point z
$\psi_j(z)$	The imaginary part of the j^{th} Cauchy coordinate basis function evaluated at a point z
ϕ	The conformal factor
θ	The angular factor



1 Introduction

In this work, we address the problem of deformations for 2D and 3D shapes. We start with an algorithm to deform complex articulated 3D shapes. By articulated shape we mean one whose general set of poses (or kinematics) can be described using a simple subspace representation, such as a skeleton, as in character animation of humans, animals and similar creatures. Given an arbitrary pose, our goal is to design a compelling, naturally-looking deformation of the rest shape in this pose.

There are three important properties which a deformation method should possess: (i) the deformed shape should preserve local details present in the rest shape, such as fine-scale geometric skin characteristics (e.g., bumps or wrinkles); (ii) the deformed shape should conform to the *characteristic shapes* of the specific character being animated: for example, muscles should bulge or cloth should crease; (iii) the deformation should be a smooth function of the pose, such that a plausible and aesthetically-pleasing animation of the shape is obtained as the deformation controls continuously modify the pose.

In the first part of our work (chapter 2) we exploit recent developments in differential surface representations and detail-preserving surface deformation for the purpose of articulated character animation. These techniques are capable of intuitively and efficiently deforming complex surfaces while preserving local details at multiple scales. Employing these deformation techniques addresses the first and third criteria listed above, namely, local detail preservation and smooth dependency on the pose. However, pure geometric deformation methods cannot be aware of the characteristic shapes of the deformed object, and this information must come from an external source. We provide this *contextual* information in the form of *examples* of the animated object in a small number of poses. Our key observation is that the clear separation of characteristic shape from intrinsic surface detail allows effective interpolation *and* extrapolation of the example shapes. Combining these two independent components, we describe a deformation method that can handle shapes of high geometric complexity and produce natural skin deformation and animation, in an effective and space-efficient manner.

A major drawback of all surface-based techniques is that it can be applied only to a limited amount of shape representations. On the other hand, space deformation techniques operate on the ambient space rather than directly on the embedded object, hence they can be applied to a large variety of geometric representations. In the second part of our work (chapters 3 and 4), we construct several space deformation algorithms based on barycentric coordinates and apply them successfully to planar shape and image deformation and interpolation.

Barycentric coordinates are a very useful mathematical tool for computer graphics applications. Since they allow to infer continuous data over a domain from discrete or continuous values on the boundary of the domain, barycentric coordinates are used in a wide range of applications - from shading, interpolation [JSW05], and parameterization [DMA02, SAPH04] to, more recently, space deformations [JMD*07, LKCOL07, LLCO08].

Traditionally, barycentric coordinates in $\mathbb{R}^n$ are defined as the real coefficients of an affine combination of vectors in $\mathbb{R}^n$. As such, they operate identically on each coordinate. When working in the plane, barycentric coordinates in $\mathbb{R}^2$ can also be considered as an affine combination of *complex* numbers with real coefficients, thus we propose to consider also the case where the coefficients are themselves allowed to be complex. This new point of view has a few advantages: First, it allows the definition of *complex barycentric coordinates*, permitting a *different* linear operation for each of the two coordinates, through which new effects can be achieved. Second, it unleashes the rich theory of complex analysis, simplifying the underlying theory considerably.

Complex barycentric coordinates are especially useful for 2D shape and image deformation. In a typical application scenario, the user defines a *source* contour, usually a polygon, and deforms it to a *target* contour by moving its vertices. This indicates to the application that the region within the source contour should be deformed in some natural way to the region within the target contour, such that the per-edge correspondence is respected. As this operation is fundamental for 2D animation applications, many algorithms have been proposed for it, a large number of them based on barycentric coordinates [JMD*07, LKCOL07]. Using barycentric coordinates means that the coordinates of a point x in the source polygon are expressed as an affine combination of the coordinates of the source polygon vertices, and x is then mapped to $f(x)$ – the same affine combination of the coordinates of the vertices of the

target polygon. Recently, observing that traditional (real) barycentric coordinates *by definition* reproduce affine transformations, hence not ideal for shape-preserving deformations because they may introduce shears, Lipman et al. [LLC08] generalized the concept of barycentric coordinates to be the standard linear combination of the coordinates of the vertices of the polygon *plus a linear combination of the normals to the edges of the polygon*. They showed how to use these coordinates in order to generate a shape preserving mapping of the interior of the source polygon.

We show that Lipman's Green coordinates are a special case of complex barycentric coordinates, and provide a very simple analytic formula for them. In addition, we propose new complex barycentric coordinates for 2D shape deformation, which are shown to improve the Green coordinates, as the deformation better fits the user's specifications, without losing any of the properties of the Green coordinates. Finally, we give simple analytic formulae for the *derivatives* of the Green coordinates. This allows us to define constraints on the derivative of the deformation, in addition to positional constraints, thus allows replacing the source-target polygon contour ("cage") user interface with a more intuitive and user-friendly point-to-point user interface.

The main challenge of planar shape deformation is to be able to create high quality deformations of a given 2D shape with as little user input as possible without losing control over the result. In other words, an ideal deformation system should allow the user intervention when it is required but infer all the missing data automatically. Once the user fixes a small set of constraints, the system should find the *best* deformed shape that satisfies these constraints. What is considered best depends, of course, on the application. Research in recent years has focused on finding deformations that best preserve the fine details of the source shape. This means that the deformation should be smooth, avoid unnecessary variations, and locally should resemble only rotation and possibly uniform scale. Shear and non-uniform scale should be avoided, and "foldovers" of the image should not be allowed under any circumstances. These desirable properties are precisely those of conformal maps - injective harmonic maps whose two components satisfy the Cauchy-Riemann equations (which mean that their gradients are perpendicular to each other and have the same magnitude). They preserve the angles between intersecting curves and also orientation,

having strictly positive Jacobian (determinant) throughout the domain, thus are ideal for shape deformation.

Conformal maps are central to complex function theory [Ahl79] and have been the subject of intense study for centuries. Most well-behaved complex functions are holomorphic, and with the additional requirement that their derivative does not vanish, they become conformal. At first glance, this would seem to imply that there are plenty of conformal maps, so it should not be too difficult to generate one satisfying some reasonable user-supplied constraints. In fact, the celebrated Riemann mapping theorem [Ahl79, Ch. 6] guarantees that there exists a (unique up to a few degrees of freedom) conformal map between any two simply-connected regions of the plane. However, the reality of working with conformal maps is quite different. It is notoriously difficult to generate the Riemann maps, and when the boundaries of the two regions are significantly different, the conformal map between the two can be surprisingly complex, with enormous differences in scale being quite common. It is even more difficult to generate conformal maps when more sophisticated constraints are present. For example, it is impossible to impose a correspondence between more than three pairs of points along the two respective boundaries. Thus controlling conformal maps in a natural way to achieve desired results is quite difficult. This is the main problem we address in chapter 4.

A common approach to generating deformations in practice is to search for the solution within some meaningful subspace of the deformation space. Finding the most suitable subspace is the main challenge. A very popular approach is to use a fixed set of basis functions (depending only on the source shape), also known as *barycentric coordinates*, and then the deformation is expressed as a linear combination of these basis functions. Possible choices are mean-value coordinates [Flo03] or harmonic coordinates [JMD*07]. In chapter 3 we suggested using holomorphic basis functions since they have the potential of producing conformal maps if the first complex derivative of the function does not vanish. In chapter 4, we provide a concise way to guarantee that the derivative will actually not vanish at any point inside the domain. Deformation methods based on barycentric coordinates usually lead to efficient algorithms since the basis functions can be computed in a preprocess step. This is the approach we use.

The input to our algorithm is a polygonal boundary curve enclosing a simply connected planar region and a small set of *corner* points along the boundary. During the animation

session, the user controls the target shape by specifying the orientation of the tangent vector at any boundary point. At a corner point, two distinct tangents should be specified, giving the user the freedom to change the corner angle as well as its orientation. This gives the user a large amount of control over the behavior of the deformation. Moreover, the resulting deformation is always a conformal map which satisfies the user constraints exactly.

Despite it not being possible, in general, to generate a conformal map that maps one planar region to another with an exact prescription of the boundary behavior, as we show in chapter 4, there always exists a conformal map that follows an exact angle prescription along the boundary. We provide the theory and the exact recipe to efficiently compute this conformal map using a new type of barycentric coordinates, which we call Hilbert coordinates.

2 Context-Aware Skeletal Shape Deformation

In this chapter, we describe a system for the animation of a skeleton-controlled articulated object that preserves the fine geometric details of the object skin and conforms to the characteristic shapes of the object specified through a set of examples. The system provides the animator with an intuitive user interface and produces compelling results even when presented with a very small set of examples. In addition it is able to generalize well by extrapolating far beyond the examples.

2.1 Background

The skin-skeleton paradigm is very popular in the animation industry and is the method of choice in most commercial modeling/animation packages. Its popularity stems from the intuitive manipulation, the ability to quickly solve inverse-kinematics on a small subspace (the skeleton) and efficient mapping of the animation mechanisms onto the graphics hardware. The difficult part is obtaining high-quality deformations of the skin given a pose of the skeleton. The most wide-spread skin deformation (or, in short, skinning) technique is the so-called Skeletal Subspace Deformation (SSD) method (see [LCF00] for a description), which transforms each vertex of the skin by a weighted linear blend of transformations associated with the skeletal joints influencing that vertex. These blending weights are typically set and tweaked by the animator; they make SSD notoriously difficult to control: the linear nature of the transformation blending causes the candy-wrapping and elbow-collapse artifacts. Consequently, the joint influence weights must be tweaked so that *every* possible pose looks reasonable, an extremely tedious, unintuitive and time-consuming chore. Lastly, SSD (like any other skinning technique) cannot achieve credible contextual movement, because the algorithm lacks any knowledge of these movements. A bend of an arm will not cause the muscle to bulge since all reachable shapes are limited to the linear subspace of transformations, none of which contain this information.

To overcome the problem of context freedom, the Pose-Space Deformation (PSD) technique [LCF00, SCFRC01] augments SSD by morphing with *displacements* taken from a set of user-supplied example skins. Usually the animator designs a few key poses of the skin+skeleton, manually deforming the skin such that it attains the correct shape for the character in those poses. Alternatively, the example skins can be obtained by 3D acquisition [ACP03,

ASK*05]; skeleton rigging and precise registration and correspondence of all skins is then required. The interpolation can be performed on a per example basis, assigning the same weight to all the skin vertices of the example (as in [LCF00, SCFRC01]), or on a per-vertex basis, assigning each vertex of the example a different weight ([KM04, RLN06]). The latter is more expensive, yet much better suited for situations where the examples are sparse. Kry et al. [KJP02] perform principle component analysis on the displacements learned from examples to compress their representation. Mohr and Gleicher [MG03] use examples to automatically find additional skeleton joints and weights which can reduce some of the undesirable effects of SSD.

One of the underlying problems with SSD-type deformation is lack of treatment of the skin surface as a connected manifold, because each vertex is transformed independently of its neighborhood on the surface. Therefore, self-intersections may easily occur. When using a differential surface representation such as we advocate, the deformed skin is obtained as the result of a global optimization process that accounts for the geodesic relationships between the skin vertices, resulting in a graceful and detail-preserving deformation. Recent skin deformation methods have adopted variants of this approach [ASK*05, SZGP05, DSP06, HSL*06, SYBF06, YHM06], yet the full potential of differential deformations, coupled with examples, has not yet been explored.

Surface deformation techniques have been recently shown to be effective in animation applications based on motion-capture data [BPGK06, HSL*06, KS06, SYBF06, YHM06]; most of these methods rely on non-linear global optimization and are context-free. The quality of the deformations they produce is typically much higher than that of SSD, including complete avoidance of candy-wrapping and joint collapse; however, this comes at a significantly higher computational price.

The MeshIK system [SZGP05, DSP06] supports example-based posing of a mesh (without a skeleton) by non-linear optimization in the space of the examples. While performing well in a dense space of examples, the technique has difficulty to extrapolate or operate with a sparse example set. Another problem with MeshIK is its strong dependence on the example shapes at run-time: the non-linear optimization procedure incorporates the complete set of examples within the global non-linear solution, which becomes very expensive as the number of examples increases. This problem was alleviated in [DSP06] by operating in a meaningful

subspace consisting of a set of “bones”, which, although one step back towards a skeletal structure, do not necessarily fit together to a well-formed skeleton, but rather define a set of rigidly-moving regions of the mesh.

Anguelov et al. [ASK*05] present a comprehensive framework, called SCAPE, for human shape acquisition and animation. SCAPE combines example-based deformation with linear differential deformations. More specifically, SCAPE assumes that each rest-pose triangle is first rigidly transformed by exactly one joint, thus propagating the skeleton transformations to the skin. Each triangle's transformation is additionally *corrected* by another transformation, learned from the example skins by linear regression methods. The correction transforms, learned per triangle, are functions of the pose configuration of the two closest influencing joints; up to 63 parameters per triangle needed to be stored in memory. The connected skin surface can then be reconstructed using Poisson stitching [YZX*04]. Due to the linear regression used, when the correction transformations contain rotational components, the result deformation will suffer from visual artifacts similar to those of SSD. Extending SCAPE to use non-linear regression can be expensive.

Our approach is similar to SCAPE [ASK*05] in that we also use an underlying linear deformation approach; however, we allow interactive marking of the joint influences, we do not restrict the influence of each skin element to one or two joints, and we propagate the skeleton transformation further using harmonic interpolation [ZRKS05]. In our framework the underlying deformation is already detail-preserving, thus local surface texture deformation need not be captured or corrected by the examples. When all the example data is present, our method will interpolate the example skins precisely when provided with the example skeleton configuration. This is not possible with SCAPE. In addition, we develop a compact representation of the example data, such that just a small number of localized skin elements need to contain augmented example information, with minimal impact on deformation quality.

Using example meshes as an input for a deformation method can be problematic for resource-demanding applications such as computer games. Typical games use a large number of different characters; many of them may be present in a single game scene. The amount of memory allocated for deformation purposes is usually small compared to textures, scene geometry, AI, physics, etc. Using examples to guide a deformation may easily lead to an

increase in the amount of memory allocated for character geometry by few orders of magnitude. Our algorithm needs fewer examples than PSD [LCF00, SCFRC01] in order to produce high-quality results. In addition, the examples themselves are represented in a compact manner.

In parallel to us, Wang et al. [WPP07] developed an approach similar to ours. However, contrary to our method, their technique relies on regression methods and needs many examples in order to learn the deformation model. An important contribution of their work is the formulation of an approximation to the Poisson system, which improves performance.

2.2 Outline and contributions

Our deformation framework enables interactive posing and animation, and is based on the skin-skeleton paradigm familiar to artists. We allow intuitive control over the skin animation by painting a rough influence field of each joint on the skin. Given a desired skeleton pose, the skin is first deformed using the detail-preserving shape editing approach. This already gives reasonable results; however, if contextual data is present, the result is also influenced by the characteristic shapes extracted from the examples via a differential "morphing" process that enables correct extrapolation (Section 2.3). The characteristic shapes are compactly represented, such that essentially only a fraction of the skin elements (called anchors) contains example information; the correction for the entire shape is performed in real-time by smooth interpolation across the surface (Section 2.4). Our main contributions in this chapter are:

- An example-based skin deformation framework that works well with a very sparse example set and enables interpolation as well as meaningful extrapolation of example data.

- A framework based on the clear separation of intrinsic surface detail information from pose-dependent characteristic shape information.

- Compact representation of the example data, enabled by the generally low-frequency nature of the characteristic shapes. This leads to space- and time-efficient deformation with an intuitive tradeoff of memory consumption and performance for result quality.

2.3 Shape deformation framework

We describe the setup and interface of our deformation framework first from the user's point of view, and then detail the core algorithms present in the system.

2.3.1 User setup and notations

Our system requires minimal user effort to setup for skeletal shape deformation. The input is a manifold triangle mesh, which we call the *rest shape* $\mathcal{S}_0 = (\mathcal{V}_0,\mathcal{F})$ and its associated skeleton structure having joints $\mathcal{B} = \{b_1,...,b_K\}$ and links between the joints. In general, we denote the mesh geometry (vertex positions) by $\mathcal{V} = \{v_1,...,v_N\}$ and the connectivity (the set of faces in the mesh) by $\mathcal{F} = \{f_1,...,f_M\}$. In addition to providing the skeleton, the user is required to establish a basic correspondence between the skeleton and the rest shape: for each joint b_k, the vertices of $\mathcal{S}_0$ associated with b_k are specified, meaning that the user marks regions $H_k \subset \mathcal{V}$ of the rest shape surface that should completely follow the movement of each joint. We call these regions *joint handles*. As we will see later, when no example shapes are provided, the joint handles are transformed exclusively by the corresponding joint transformation; when examples are present this behavior is modified by mimicking the examples. Note that the handles should be disjoint, but need not cover the entire mesh surface. Their marking can be performed via a simple binary painting interface (see Figure 2.1 and the video that accompanies the work of [WSLG07]). No numerical input of influence weights is required; in a sense our system automatically deduces the joint influences on the entire surface from the painted regions.

This basic setup is enough to perform detail-preserving skeletal deformation with the algorithm described next. However, this will typically not capture the "characteristic behavior" of the object – its "context". To increase the realism of the deformation, the system incorporates example shapes that put the deformation into context, demonstrating characteristic deformations of the shape, such as bulging of muscles and appearance of folds for human or animal shapes. Our system accepts such examples in the form of D additional mesh geometries $\mathcal{V}_1,...,\mathcal{V}_D$ (the connectivity $\mathcal{F}$ is shared by all the shapes), coupled with corresponding skeleton poses, specified by joint transformations $P_j = \{\mathbf{R}_{j,1}, \mathbf{R}_{j,2}, ..., \mathbf{R}_{j,K}\}$ for

each example $\mathcal{V}_j$. Only a small number of examples are needed to produce highly realistic deformations; an example shape can exhibit several characteristic behaviors simultaneously (for instance, all the limbs of the character may bend). In addition to providing the example shapes, the user can specify the desired tradeoff between efficiency (i.e., space and time complexity) and quality of deformation by controlling the compactness of the example shape representation; this is described in more detail in Section 2.4.

2.3.2 Basic skeletal surface deformation

Once the joint handles H_k are specified, the surface of the rest shape $\mathcal{S}_0$ can be deformed to conform to any given skeleton pose. For this purpose we associate a continuous scalar field $\mathbf{w}_k$ with each joint b_k, which assigns an influence value $\mathbf{w}_k(v)$ to each mesh vertex v. The vertices that "belong" to the joint handle are assigned the value $\mathbf{w}_k(v) = 1$, while vertices belonging to other joint handles are assigned $\mathbf{w}_k(v) = 0$. All other vertices are assigned values $\mathbf{w}_k(v) \in [0,1]$ obtained as discrete harmonic functions over the mesh. We compute these values for each mesh *vertex* by solving the Laplace equation:

$$\mathbf{L}\mathbf{w}_k = 0 \qquad 2.1$$

subject to the Dirichlet boundary conditions $\mathbf{w}_k(v) = 1$ for $v \in H_k$ and $\mathbf{w}_k(v) = 0$ for $v \in H_l$ where $l \neq k$. The operator $\mathbf{L}$ is the Laplace-Beltrami operator associated with the rest shape $\mathcal{S}_0$, discretized using the cotangent weights [PP93]. After computing the values of $\mathbf{w}_k$ for the mesh vertices, we compute corresponding scalar fields, $\mathbf{h}_1,...,\mathbf{h}_K$ for the mesh triangles by averaging the values at the three vertices of each triangle. Since the joint handles are mutually exclusive, $\mathbf{w}_1,...,\mathbf{w}_K$ are a partition of unity over the mesh. As a result, $\mathbf{h}_1,...,\mathbf{h}_K$ sum to unity on each face.

The influence fields $\mathbf{h}_k$ are computed once and can then be used to perform arbitrary deformations of the shape: they serve as blending weights to distribute the transformation of the joints to the skin mesh. Given an arbitrary skeletal pose $P = \{\mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_K\}$, where $\mathbf{R}_k$ are relative joint transformations with respect to the rest pose, each mesh face f is assigned a blended transformation, expressed as $\mathbf{R}(f) = \mathbf{h}_1(f)\mathbf{R}_1 \oplus \mathbf{h}_2(f)\mathbf{R}_2 \oplus ... \oplus \mathbf{h}_K(f)\mathbf{R}_K$. We assume

that the transformations $\mathbf{R}_k$ are rotations, as this is a common case in skeletal animation (although arbitrary transformations can be easily handled via polar decomposition).

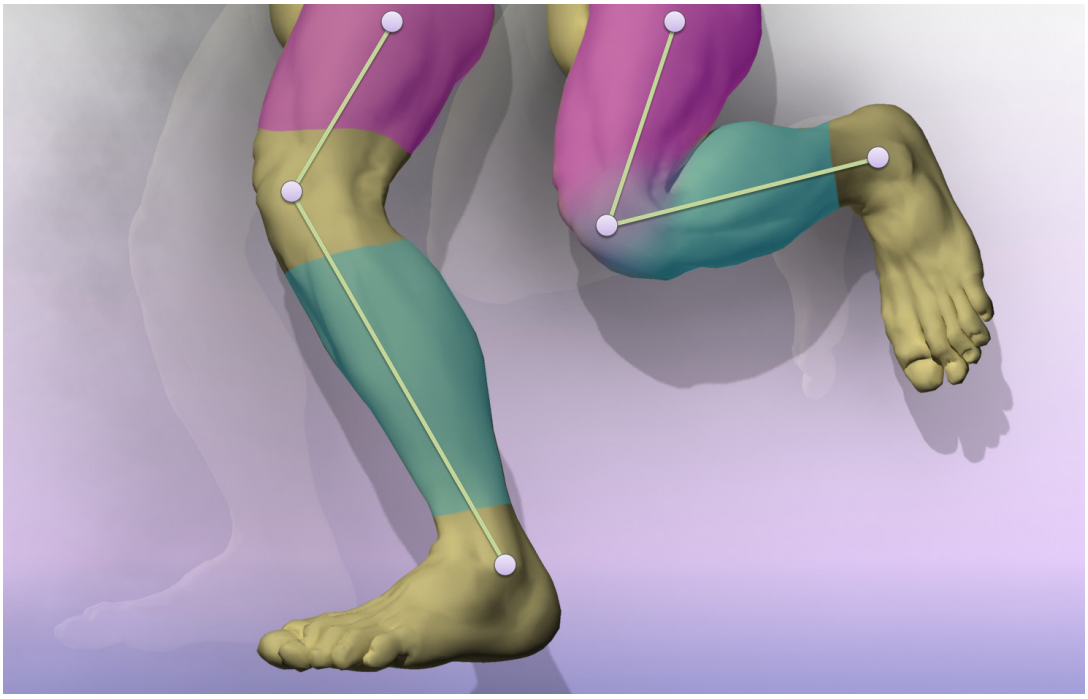


Figure 2.1: Painting joint influences and deforming the skin. (left) The colored regions designate the joint handles that are under the influence of corresponding joints (the knee and the thigh), as painted by the user. (right) The detail-preserving deformation uses these regions as modeling constraints that move rigidly with the joints; the rest of the skin deforms according to the optimization process.

To correctly blend between rotations, the operator $\oplus$ cannot be a simple addition – this would lead to the well-known artifacts of linear blend skinning (as in SSD). Instead, we represent the rotations using log-quaternions [Gra98]; these are 3-vectors whose direction is the axis of rotation and the magnitude is the rotation angle. Linear combinations of log-quaternions still represent rotations, and are therefore able to efficiently blend between more than two rotations. Note that in general, when linearly interpolating log-quaternions, the interpolation path is not a geodesic on the sphere of unit quaternions. Thus, interpolation of log-quaternions is sensitive to the initial orientation of the quaternions involved. This means that the mesh is not invariant to a common rotation of all skeleton joints. In addition, in order to avoid the singularities of the quaternion's logarithm, we prefer to use log vectors with smaller magnitude. Therefore, when blending several rotations $\mathbf{R}_1, ..., \mathbf{R}_K$, we think of one of them

(say, $\mathbf{R}_1$) as the identity and represent the other rotations relative to it. The result is a new set of rotations: $\mathbf{R'}_1, ..., \mathbf{R'}_K = \mathbf{I}, \mathbf{R}_1^{-1}\mathbf{R}_2, ..., \mathbf{R}_1^{-1}\mathbf{R}_K$. After blending these relative representations, we multiply back by $\mathbf{R}_1$ to obtain the absolute result. Such relative blending of rotations is insensitive to the skeleton orientation. When blending between the identity and another rotation, linear combination of the log-quaternion representation gives the same effect as spherical linear interpolation (slerp). This means that when only 2 joints are involved, the resulting interpolation technique is equivalent to slerp.

Once the blended rotations $\mathbf{R}(f)$ are computed per face, they are applied to the rest shape triangles, transforming each of them independently. To obtain the final connected mesh, we apply Poisson stitching to these transformed triangles [SP04, YZX*04], namely, we compute the gradients of the transformed triangles and solve the Poisson equation for the deformed mesh vertices:

$$\mathbf{L}[\mathbf{x}\,\mathbf{y}\,\mathbf{z}] = \mathrm{div}[\mathbf{g}_x\ \mathbf{g}_y\ \mathbf{g}_z] \qquad 2.2$$

where **x, y, z** are the deformed mesh coordinate functions and $\mathbf{g}_x$ $\mathbf{g}_y$ $\mathbf{g}_z$ are the 3×3 stacked gradient matrices of the transformed triangles. **L** is the same Laplace operator used in Equation (2.1). Note that the Poisson equation requires boundary conditions because the gradients are translation-invariant; we use the positions of the vertices associated with the root joint of the skeleton as Dirichlet boundary conditions. Botsch et al. [BSPG06] recently showed that instead of plugging the triangles gradients to the right-hand side of Equation (2.2), it is sufficient to use the so-called deformation gradients [SP04]. This will lead to the following, equivalent, linear system:

$$\mathbf{L}[\mathbf{x}\,\mathbf{y}\,\mathbf{z}] = \mathrm{div}[\mathbf{S}] \qquad 2.3$$

We simply treat the triangle rotations, $\mathbf{R}(f)$ as 3×3 deformation gradient **S** matrices and plug them into the right-hand side of Equation (2.3). This eliminates the need to constantly compute the gradients during deformation. For details on the Poisson system setup and the equivalence of the two linear systems, see [BSPG06]. Figure 2.2 and Figure 2.3 compare what we get vs. SSD skinning.

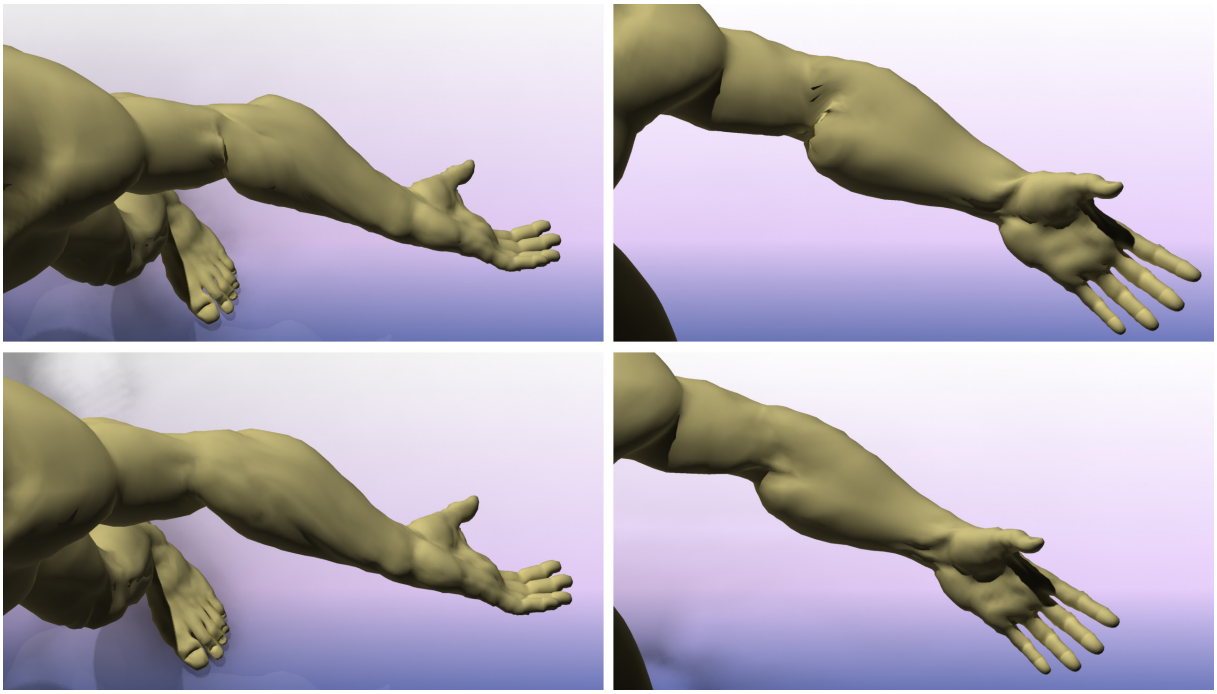


Figure 2.2: Skinning using SSD and our method without examples. ***(top row)*** *SSD. Notice the artifacts in the elbow area.* ***(bottom row)*** *Our method.*

The deformation technique described above is a variant of the method proposed by Zayer et al. [ZRKS05] and Lipman et al. [LCOGL07], adapted to the skeletal deformation setup. Zayer et al. [ZRKS05] did not explicitly discuss the limitations of transformation blending when more than two handle regions are involved; Lipman et al. [LCOGL07] used a non-linear system to correctly blend the transformations. A somewhat similar approach to skeletal deformation was demonstrated by Shi et al. [SYBF06]; however, they rely on a manual assignment of skin vertices to bones and joints and require an augmented volumetric mesh structure, which increases the complexity of the solution.

2.3.3 Using context: example shapes

When example shapes are provided, our system incorporates the additional information about the shape's characteristic behavior, as contained in the examples. Since the geometric deformation method described in the previous section is detail-preserving, and thus faithful to the local geometric texture of the shape, the difference between an example shape and the rest shape deformed into the pose of the example, is typically smooth and varies slowly across the surface. We use a differential representation of this difference, on a per-face basis, and enable

its blending for arbitrary poses. The low-frequency nature of this additional deformation information also allows compact representation, as described in the next section.

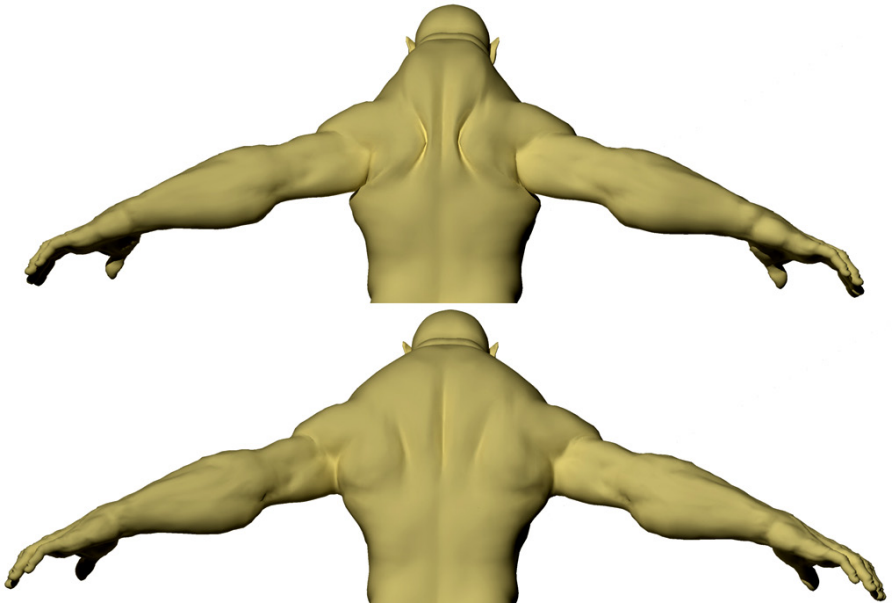


Figure 2.3: Skinning using SSD and our method without examples. ***(top)*** *SSD. Notice the artifacts in the shoulder area.* ***(bottom)*** *Our method.*

To extract the deformation nuances from an example shape $\mathcal{V}_j$, we first compute a rotation for each triangle of the example. The rotations are computed according to the algorithm described in Section 2.3.2, using the example pose P_j as input. We apply the inverse rotation to each of the example's triangles. This can be thought of as a process of transforming the example into the pose of the rest shape. The result is a set of new normalized examples, all in a unified coordinate system. We then compute the relative transformation between the rest shape triangle and the corresponding triangle in the *normalized* example (we use two triangle edge vectors and the normal to disambiguate the transformation). We obtain the so-called deformation gradient $\mathbf{T}_{j,m}$ (for the j'th shape and m'th triangle) which encodes the example shape in a rotation-invariant manner, because the rotation due to pose has been factored out. The full representation of the characteristic behavior present in the j'th example shape is, therefore, its skeleton pose P_j and the set of relative transformations $\{\mathbf{T}_{j,1},...,\mathbf{T}_{j,M}\}$.

When an arbitrary pose P is given, we would like to deform the shape in such a way that it benefits from *all* the examples, proportionally to the similarity between the skeleton poses. In other words, we would like to create meaningful interpolations and extrapolations of the

examples in the parameter space of skeletal joint transformations. The similarity metric should be local, in the sense that when only part of P (say, the elbow configuration) is close to an example pose, then the corresponding part of the shape is significantly influenced by the example, whereas the rest of the shape is not. This allows the use of a very small number of example shapes that exhibit several characteristic behaviors of different limbs simultaneously.

We employ the weighted pose-space deformation (WPSD) ideas [KM04] to compute the similarity metric between poses and the blending weights. In a nutshell, WPSD accepts a set of poses $\{P_j\}$ described by the Euler angles of the joint transformations, and K scalar fields defined on the skin mesh elements, defining the influence of each joint on each mesh element. Traditionally, these joint influences were manually painted and tweaked by the animator; we replace them by the harmonic influence fields $\mathbf{h}_k$ described earlier, significantly simplifying the process. Then, for a given pose P and triangle f_m, WPSD computes D weights $a_{j,m}$ that describe how close that element is to its counterpart in each of the examples. The weights $a_{j,m}$ are computed using RBF interpolation on the example poses; in addition, we incorporated the linear extrapolation technique proposed by [SCFRC01] for simple pose-space deformation (PSD) and adapted it to WPSD.

The final construction of the deformed shape is performed by computing the total deformation gradient of each triangle, applying the geometric deformation method $\mathbf{R}(f_m)$ to the triangle augmented by the blend of the example representations. Since $\mathbf{T}_{j,m}$ may contain rotational components, linear blending may lead to visual artifacts. For a precise combination, we compute the polar decomposition of each relative example transformation $\mathbf{T}_{j,m} = \mathbf{Q}_{j,m}\mathbf{S}_{j,m}$ and separately combine the rotations $\mathbf{Q}_{j,m}$ and the skew components $\mathbf{S}_{j,m}$. The final transformation for the m'th triangle is then given by:

$$\begin{aligned} \mathbf{T'}_m &= \mathbf{R}(f_m)\mathbf{A}_m \\ \mathbf{A}_m &= \left[\bigoplus_{j=0}^{D} a_{j,m}\mathbf{Q}_{j,m}\right]\left[\sum_{j=0}^{D} a_{j,m}\mathbf{S}_{j,m}\right] \end{aligned} \qquad 2.4$$

We again use log-quaternions to represent and combine the rotations $\mathbf{Q}_{j,m}$. By applying the transformations $\mathbf{T'}_m$ to the right-hand side of Equation (2.3) and solving the Poisson stitching,

we arrive at the final deformed shape. Note that only one Poisson stitching process is needed, since we do not actually solve for the intermediate shape obtained by purely geometric deformation, but only use its transformations $\mathbf{R}(f_m)$ obtained by combining the pre-computed harmonic fields $\mathbf{h}_k$. See Figure 2.4 for an example.

2.4 Compact representation of examples

As mentioned above, the characteristic deformation behavior of a shape tends to have a low-frequency nature, meaning that the difference between a basic skeletal deformation and the given example shape is a smooth function over the surface. Thus there is significant coherence within the representation of an example mesh, i.e. between $\{\mathbf{T}_{j,m}\}$, the relative transformations of the individual mesh triangles and also between $\{\mathbf{A}_m\}$. We exploit this fact to "compress" the example representation, expressing each $\mathbf{A}_m$ as a combination of a small number of basis functions, centered around particular mesh elements called *anchors*. This way only the relative transformations of the anchors need to be stored, instead of the entire set of $D \cdot M$ matrices $\mathbf{T}_{j,m}$. A conceptually similar approach was used in [SCOIT05] for geometry compression; here we apply it to deformation compression.

Assume that instead of storing all the matrices $\mathbf{T}_{j,m}$ we choose a subset of triangles $\{f_i\}$, $i \in \mathcal{C} = \{c_1,...,c_{M'}\}$, $M' << M$ for which we store this information. At run time, we can approximate the entire set of $\mathbf{A}_m$'s by smoothly interpolating this subset across the mesh. Since the $\mathbf{A}_m$'s contain both a rotational component and a skew component, we could have used the polar decomposition again in order to perform the interpolation on the two components separately. In practice, if the subset of anchors is nicely distributed over the mesh, linear interpolation will suffice. In total we have 9 scalar values to interpolate over the mesh. Denote by $(d_1,d_2,...,d_M)^T = \mathbf{d}$ one of these scalar fields; the values d_i, $i \in \mathcal{C}$ are known. We can approximate the rest of $\mathbf{d}$ by solving a Laplace problem for $\mathbf{d'}$:

$$\mathbf{L}\mathbf{d'} = 0 \qquad 2.5$$

with the boundary conditions $d'_i = d_i$ on the anchors. The operator $\mathbf{L}$ is the triangle-based Laplacian operator, defined on the graph dual to the mesh. For simplicity, we define $\mathbf{L}$ with uniform weights. The quality of approximating $\mathbf{d}$ by $\mathbf{d'}$ depends of course on the number and

position of the anchors $\mathcal{C}$. We employ a greedy algorithm to choose these anchors in the preprocess stage, as proposed in [SCOIT05]. We use the rotation-skew decomposed components of the $\mathbf{T}_{j,m}$'s separately in an attempt to optimize the approximation of both components of the $\mathbf{T}_{j,m}$'s. In the beginning, we initialize $\mathcal{C}$ with a small number of randomly distributed anchors. Then we solve (2.5) for the 3 rotational (log-quaternion) scalar fields and find the triangle whose entry in **d'** has the maximal error with respect to its counterpart in **d**. The error is defined as the angle of the relative rotation between the element in **d'** and the element in **d**. This triangle is added to $\mathcal{C}$ and the process repeats. After the desired number of rotational anchors is obtained, we proceed to compute the skew anchors in the same fashion. Since the skew matrix is symmetric, we have 6 independent scalar fields to solve for. The error metric for the skew component is the Frobenius matrix norm.

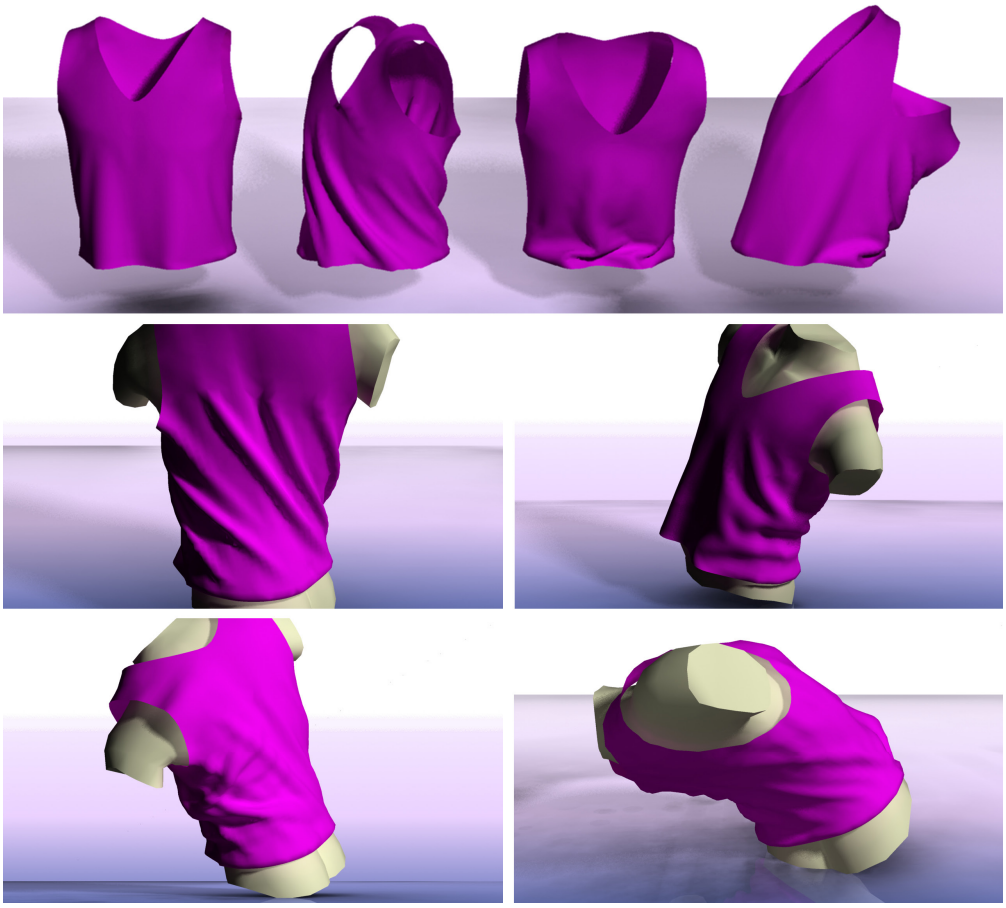


Figure 2.4: Deformation using characteristic shapes. ***(top row)*** *Rest shape and 3 examples.* ***(bottom two rows)*** *Deformations of the rest shape in arbitrary poses.*

The process is then repeated for all example meshes. The total number of anchors M' is limited by the space consumption constraints provided by the user (we need to store $9 \cdot D \cdot M'$ scalars to sparsely represent all the examples).

At run-time, the D sets of anchor values for the rotational and the skew components are blended using the WPSD weights, as described in Section 2.3.3, resulting in a single set of blended anchors. The transformations $\mathbf{A}_m$ are then computed for the entire mesh by solving (2.5) with the blended anchor boundary constraints.

Our experiments show that for natural shape examples, only a small fraction of the mesh triangles need to be designated as anchors; typically up to 5% is enough to obtain an excellent approximation; a significantly reduced number of anchors leads to smoothed incorporation of example behavior since the harmonic interpolation over-smoothes the relative transformations. The advantage of such a sparse representation is two-fold: it drastically reduces the memory consumed by the examples by at least a factor of 20, and reduces the time spent on computing WPSD weights and evaluating Equation (2.4) at run-time, since the computation is performed only on the anchors. The price paid is the solution of an additional linear system; however, this can be significantly accelerated, as explained in the next section.

2.5 Implementation issues

The main computational bottleneck in our approach is the solution of sparse linear systems. In the preprocess stage we compute the harmonic fields $\mathbf{w}_k$ (Equation (2.1)), as well as the greedy choice of the anchors for sparse example representation (solving Equation (2.5) with varying constraints). At run-time, given a skeleton pose, solving a sparse system is required in order to reconstruct the blended examples information (solving Equation (2.5) for 9 right-hand sides) and in the final Poisson stitching (Equation (2.3)). Naturally, it is desirable to optimize the run-time performance, even at the expense of a longer pre-processing stage.

We use a direct sparse solver [Tol03] to precompute the Cholesky factorization of the Laplacian matrix involved in the Poisson stitching; since the system matrix does not change at run-time, the factorization can be reused to solve for multiple right-hand sides by back substitution, which is very fast compared to standard iterative solvers [BBK05]. Solving the linear system (2.5) for the anchors can be avoided altogether at run-time by representing the

solution **d'** as an affine combination of M' harmonic basis functions $\mathbf{u}_i$, $i \in \mathcal{C}$, constructed as in Section 2.3.2:

$$\mathbf{L}\mathbf{u}_i = 0,$$

subject to $\mathbf{u}_i(c_j)=\delta_{ij}$. Then:

$$\mathbf{d'} = \sum_{i \in \mathcal{C}} d_i \mathbf{u}_i$$

We only need to pre-compute the $\mathbf{u}_i$'s and combine them at run-time, which is cheaper than back-substitution. For additional speedup and space conservation, we selectively reduce the amount of $\mathbf{u}_i$ values stored per triangle: we sort them in descending order and keep only the larger values (e.g., such that they sum to 0.95; we then rescale them to sum to 1). The cut-off parameter may be exposed to the user for better quality control. Typically most of the original values are nearly zero because each triangle is mostly affected by the closest anchors. We also use the same process to reduce the amount of joint influence weights $\mathbf{h}_k$ stored per triangle. Hence, at run-time only one equation (Equation (2.3)) needs to be solved (using back-substitution only).

2.6 Experimental results

Our mesh deformation system has been fully implemented as a plugin to the Maya® commercial animation system. The code is not optimized, although it does use the acceleration methods described in Section 2.5.

Our system can generate deformations without the use of any example meshes, thus we can compare our results to traditional linear weighted blending of transformations (SSD). As Figure 2.1, Figure 2.2 and Figure 2.3 demonstrate, our method produces much smoother and more natural results on a humanoid model. The muscles contract naturally although only the shoulder joints rotate. Notice the self intersections near the scapula region and the over-stretching of the shoulder area when SSD is applied (Figure 2.3).

When presented with example shapes, the deformation captures the context illustrated by the examples and reproduces it. Moreover, it has significant extrapolation capabilities, as

illustrated in Figure 2.5. When the example shows how a muscle bulges slightly and the fingers bend slightly as the arm and wrist are bent, the same muscle continues to bulge more dramatically as the arm is bent further and the fingers continue to rotate as the wrist is bent further. Note that there is no skeletal structure whatsoever in the fingers! The conventional PSD method [LCF00, SCFRC01], which is based on displacement vectors, cannot handle the large rotation of the fingers correctly. SCAPE [ASK*05] will also fail on this data since its regression model is linear.

Cloth deformations that do not exhibit strong dynamic effects can be modeled with our system. We use three examples to guide the deformation of a shirt, controlled by the spine joints. Figure 2.4 shows the application of our deformation technique to the shirt. This exhibits quite complex deformations with large local rotations.

Another scenario is depicted in Figure 2.6. Here the rest mesh is a flat mesh with sinusoidal waves on it. These waves are the fine details. An example is presented which shows the mesh folded over in a circular shape by approximately 180°. This example suffices to indicate what the mesh should look like when it is bent by 360°. The result is exactly as we would expect – a cylindrical form with no distortion of the fine detailed sinusoidal waves.

The example meshes which supply the context can occupy a large amount of memory. This can be a serious problem for many applications. We represent our examples in a compact manner using a small subset of the mesh triangles – so-called anchors. Even when a very small amount of anchors are used, the fine geometric details of the mesh are preserved. The flat mesh in Figure 2.6 is a classic example of a very smooth and uniform deformation. This leads to extreme compression capabilities where only 30 anchors among 24,000 triangles were used.

Figure 2.7 shows what happens when we bend the Armadillo leg by rotating the knee joint. On the left is an example mesh. When we ignore it, the deformation preserves the fine details of the mesh but does not deform as expected and the muscles do not bulge. When we use the example, the results are improved. The same figure shows the deformation achieved when all or just some of the triangles of the example are used as anchors. Using fewer anchors leads to less memory consumption and better performance. Note that the 2% anchor image is almost indistinguishable from the 100% anchor image, even though only 290 anchors were used

instead of 14,606. Even when only 73 anchors (0.5%) are used, the fine scale details of the original mesh are not harmed.

2.6.1 The video

The video that accompanies the work of [WSLG07] (which can be found at www.cs.technion.ac.il/~gotsman/shape/EG07.zip) depicts an interactive session with our system and shows how the results described in the previous section were obtained. The interaction is easy and natural. Influence regions of the various joints may be painted onto the mesh. Then the mesh is deformed by posing its skeleton. After a pre-processing step, real-time performance is obtained during the deformation process. The video shows a comparison between using SSD and our method to deform a humanoid figure without examples, the deformation of the arm model using our method with and without examples, and the deformation of a shirt and a flat mesh using our method based on a small number of examples.

2.6.2 Complexity

A main component in our computation is solving a linear system involving the sparse mesh Laplacian matrix. We do this by factoring the matrix by a Cholesky decomposition in a pre-process, which allows to solve the system faster during interaction by back-substitution only. Cholesky factorization takes less than a second on the arm mesh containing 10,000 triangles, and back-substitution can be done in real-time for meshes containing up to 60,000 triangles. We are optimistic that significant speedups can still be obtained by performing much of these computations on the GPU (see next section).

Our anchor selection process is greedy and quite slow, since it checks the quality of each selection by solving the equation associated with it. This, of course, is linear in the size of the example meshes and their number. Despite using the fast factorization update scheme [SCOIT05], it can take up to a minute per example mesh, but since this is also done in a pre-process, it is not critical. We are also sure that a more efficient procedure is possible.

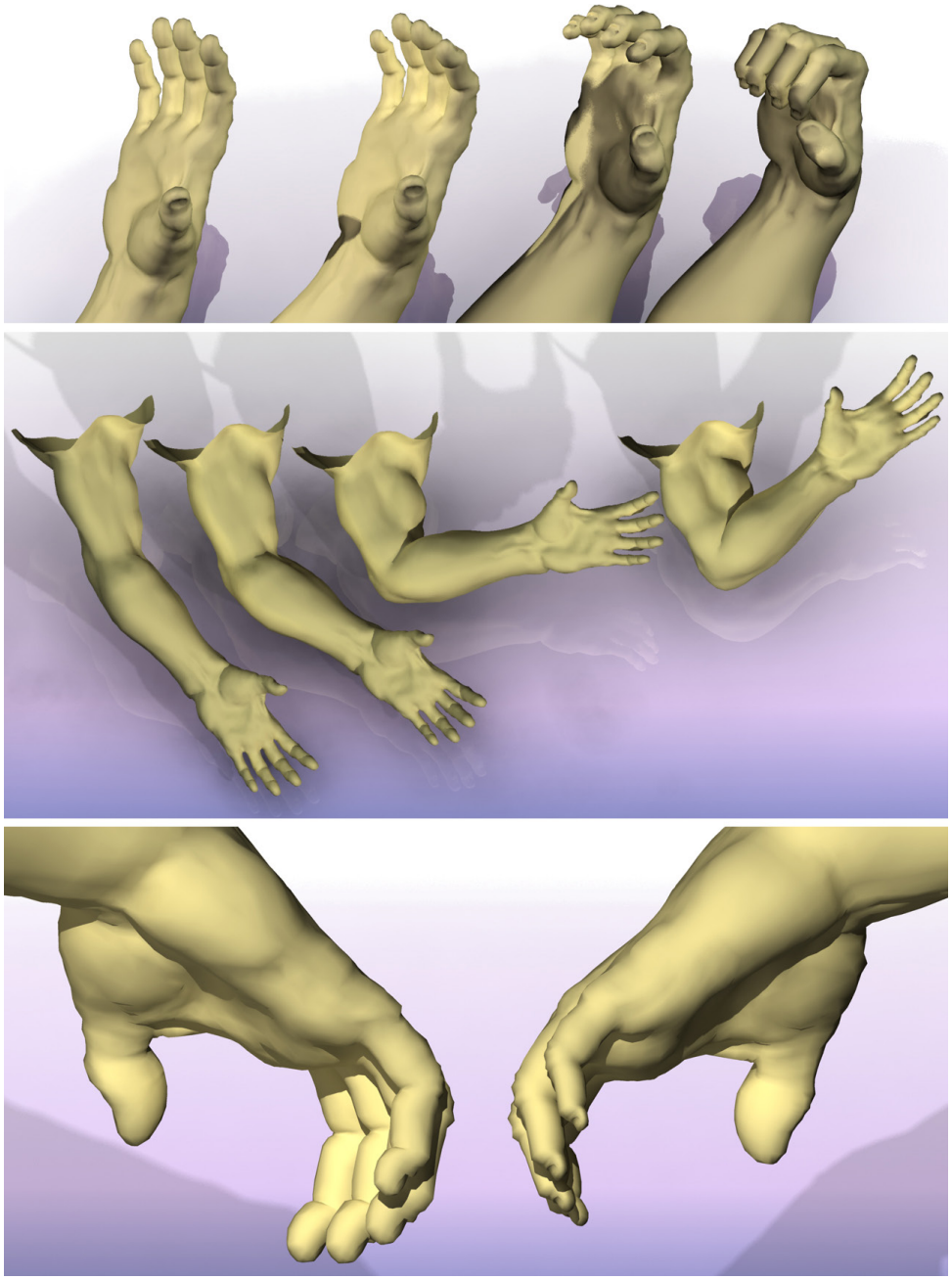


Figure 2.5: Deformation using characteristic shapes. (a) ***left to right -*** *Rest shape; one example with slight bend of fingers; two deformations of the rest shape based on the example. Note the significant bend of the fingers. There are no skeleton joints in the fingers! (b)* ***left to right -*** *Rest shape; one example with slight muscle bulge; two deformations of the rest shape based on the example. Note the significant and natural bulge of the muscle. (c) Comparison of* ***(left)*** *our deformation and* ***(right)*** *that of [LCF00, SCFRC01]. Note the shrinkage in the fingers.*

2.7 Discussion and future work

We have presented a system for skeletal shape deformation that operates within the context of given deformation examples – so-called characteristic shapes. The system has a very simple set-up compared to traditional skinning techniques, and it is robust: small variations in the specified joint handle regions lead to only small changes in the deformation results. Our system uses a differential detail-preserving deformation technique as its underlying deformation core, and thus produces plausible results even when the desired skeleton pose is very different from the provided example poses. The differential encoding of the characteristic behavior exhibited in the examples enables high-quality interpolation and extrapolation for arbitrary poses. Moreover, only a small number of example meshes is sufficient thanks to the WPSD-based interpolation which takes advantage of partial matching between poses. The computational cost of WPSD is significantly reduced by a compact representation of the example deformations, since it is applied only to representative triangles (anchors) which are a small fraction of the entire mesh.

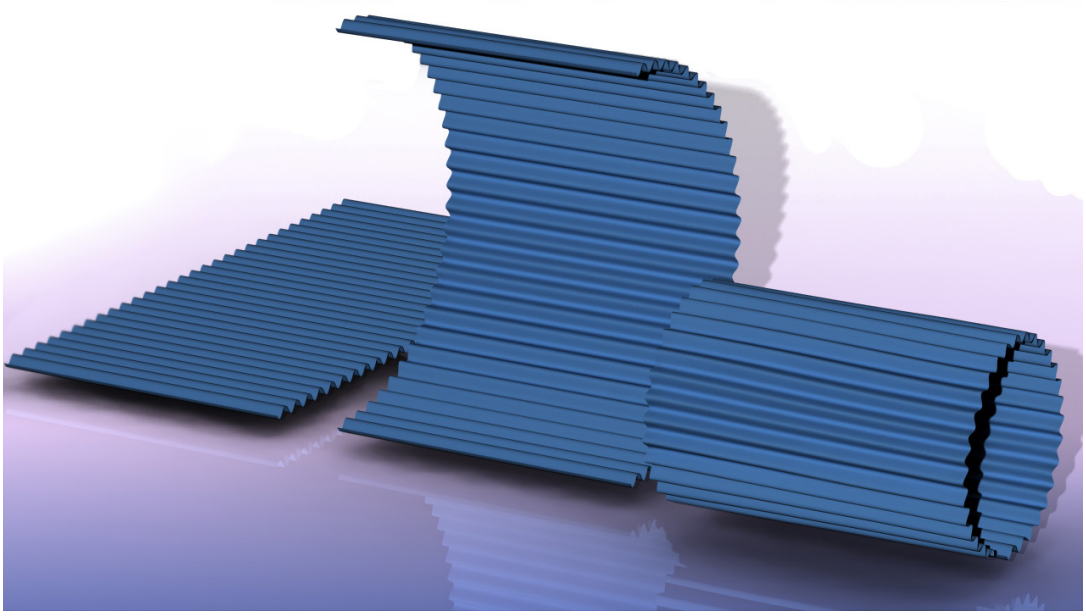


Figure 2.6: Deformation using characteristic shape. ***(left to right)*** *Rest shape of flat mesh with waves; one example with bend; deformation of the rest shape based on the example. Note the significant extrapolation of the bend. Only 30 anchors out of 24,000 triangles were used.*

The compact representation of examples allows the saving of precious storage space and also offers a viable trade-off between deformation quality and space- and time-efficiency. We use a simple greedy algorithm for the anchors selection which is far from being optimal and also quite slow, requiring re-solving of Equation (2.5) with an increasing number of constraints. Although not the bottleneck in our computations, this aspect of our system could definitely be improved.

The first stage in our deformation algorithm is a detail preserving gradient-based technique. We approximate the gradients field of the deformed mesh using harmonic functions guided by the skeleton joints rotations. In some cases, this could lead to a translational gap between a limb and its corresponding joint (see Figure 2.7 – NO EXAMPLES image). A possible solution is to constrain the positions of the handle's vertices, found in the H_k's, to the joints by adding additional boundary conditions to Equation (2.3). Since gradients are translation-insensitive, using positional constraints will not alter the gradients any further and may lead to visual artifacts. We chose not to use positional constraints in all the examples being presented in the manuscript and the video. A more sophisticated solution may use a deformation mechanism that handles translations correctly; this will probably lead to a non-linear system and will complicate the solution. The main reason that we were satisfied with the linear system behavior is the fact that, as soon as examples are added, the approximation of the gradients, achieved by the first stage, is dramatically improved and naturally eliminates the artifact (see Figure 2.7 and the video).

A possible future direction is to implement the algorithm on the GPU. In fact, apart from the linear system solver, the entire algorithm maps perfectly to the GPU. For the linear solver, we currently use a Cholesky factorization and perform back-substitution during interaction. Back-substitution is a sequential process, hence it does not map well to the GPU. A solver based on multigrid methods could be used, and since multigrid can be easily parallelized, this is suitable for GPU implementation [BFGS03, GWL*03]. However, typical multigrid methods assume that the mesh is structured, which is not the case for arbitrary mesh data, such as the inputs we deal with, so this aspect of the problem requires some additional research.

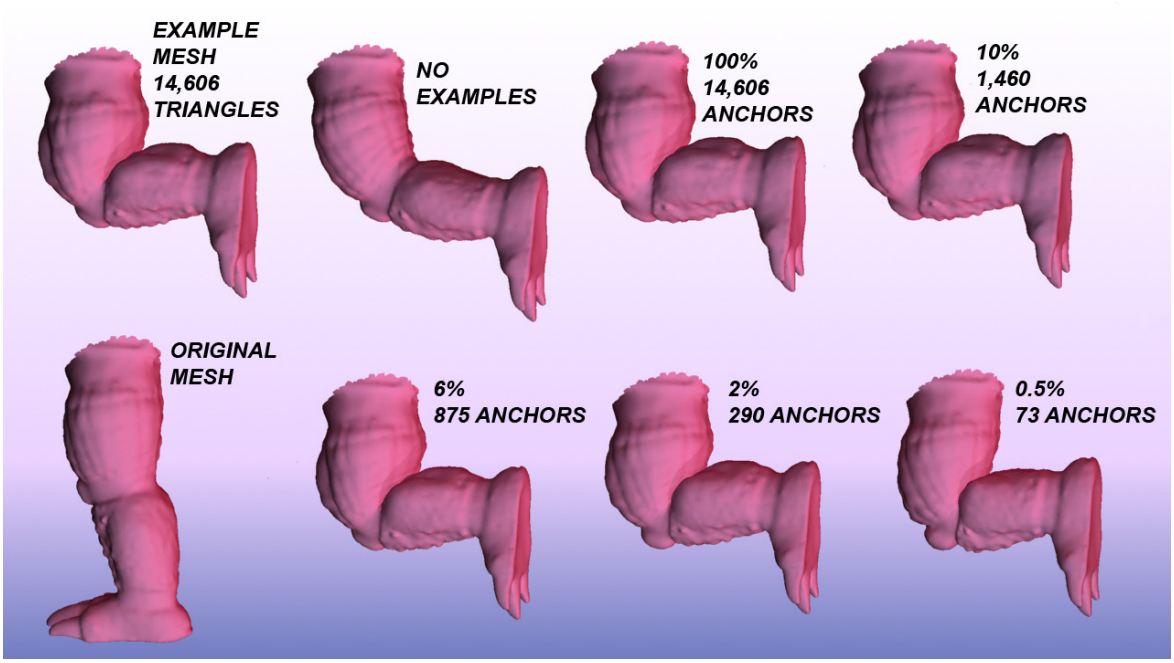


Figure 2.7: The effect of using different amount of anchors when deforming the original mesh using one example. Using just 2% of the triangles as anchors produces a muscle bulging effect almost indistinguishable from that produced with 100% anchors.

3 Complex Barycentric Coordinates with Applications to Planar Shape Deformation

Barycentric coordinates are heavily used in computer graphics applications to generalize a set of given data values. Traditionally, the coordinates are required to satisfy a number of key properties, the first being that they are real and positive. In this chapter we relax this requirement, allowing the barycentric coordinates to be complex numbers. This allows us to generate new families of barycentric coordinates, which have some powerful advantages over traditional ones. Applying complex barycentric coordinates to data which is itself complex-valued allows to manipulate functions from the complex plane to itself, which may be interpreted as planar mappings. These mappings are useful in shape and image deformation applications. We use Cauchy's theorem from complex analysis to construct complex barycentric coordinates on (not necessarily convex) polygons, which are shown to be equivalent to planar Green coordinates. These generate holomorphic mappings from a given source region to a given target region, such that the image of the source region is close to the target region. We then show how to improve the Green coordinates in two ways. The first provides a much better fit to the polygonal target region, and the second allows to generate deformations based on positional constraints, which provide a more intuitive user interface than the conventional cage-based approach. These define two new types of complex barycentric coordinates, which are shown to be very effective in interactive deformation and animation scenarios.

3.1 Previous work

The simplest real barycentric coordinates in the plane are defined on a basis of three points forming a triangle. These unique coordinates are just ratios between various areas. Real

barycentric coordinates satisfy a number of important properties, such as non-negativity, constant precision, linear precision, interpolation and smoothness. The main challenge in developing new recipes for barycentric coordinates is to find coordinates which can be applied to a wider range of bases – such as convex polygons, general polygons, continuous contours, and complexes in higher dimensions – while maintaining the attractive properties of the simple barycentric coordinates on a triangle. Many generalizations have been developed in recent years, and we will mention only those most relevant to our work.

One of the most commonly-used barycentric coordinates are the mean-value coordinates, first introduced by Floater [Flo03]. These coordinates have the nice property that they are positive on convex polygons, and can be generalized to R^3 [JSW05, FKR05, LBS06] and to the exterior of polygons [HF06]. In addition, they can also be generalized to be positive on non-convex polygons [LKCOL07]. Mean-value coordinates are derived from the mean-value theorem for harmonic functions, applied to a piecewise linear contour – a polygon. As we will show in the next sections, Green coordinates [LLCO08] can be derived by applying Cauchy's integral formula – which is the complex equivalent of the mean-value theorem for holomorphic functions – to a polygon. In this sense, the Green coordinates are conceptually a generalization of mean-value coordinates to complex functions.

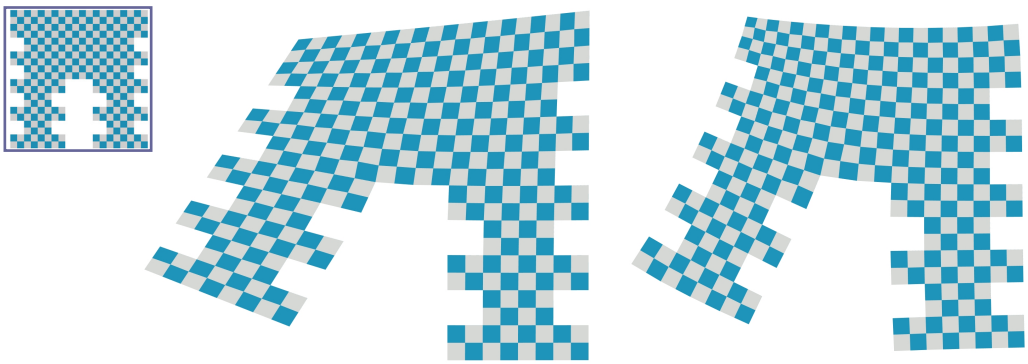
Floater et al. [FHK06] have showed that mean-value coordinates are members of a large family of barycentric coordinates, known as "three-point coordinates". We show how to derive in a similar way a family of complex three-point coordinates, and show that the Green coordinates are a member of this family.

In recent works [Bel06, WSHD07], barycentric coordinates were developed for continuous planar contours. The main challenge here is to find a barycentric coordinate function – or *kernel* – such that the resulting transform will have linear precision. We show that the Green coordinates on a polygon originate in a simple kernel on a continuous contour, which easily achieves linear precision and conformality.

One of the applications of planar barycentric coordinates is planar shape deformation, for example, for image warping. This is also a highly active research area, and we will only mention a few recent works, which are closest in spirit to ours. There are two major approaches to planar deformation – the first requires discretizing the shape into finite

elements, so that the deformation is performed by solving an optimization problem on this discretization, see [BS08] for a survey of recent linear methods. The biggest disadvantage of this approach is that the computation time is dominated by the complexity of the discretization, and not by the intrinsic complexity of the shape itself. The second approach is more analytic and does not require a discretization of the domain. An example of this approach is Moving Least Squares (MLS) [SMW06] which allows for affine, similar and rigid deformations. However, the MLS approach deforms the entire plane, and does not take into account the underlying shape of the deformed object.

In order to allow shape-aware deformations, Ju *et al.* [JSW05] suggested to define a "cage" around the shape and control the deformation by manipulating the cage. Most of the cage-based deformations use barycentric coordinates, for example [LKCOL07, JMD*07]. A notable set of barycentric coordinates are the so-called *harmonic* coordinates [JMD*07]., named that way since they are the unique solution to the Laplace equation with Dirichlet boundary conditions on the cage. In contrast to other barycentric coordinates on polygons, no analytic expression is available for harmonic coordinates. They are quite difficult to compute, and are typically approximated using a discretization of the interior of the cage. Despite the usefulness of barycentric coordinates in data interpolation applications, Lipman et al. [LLCO08] have observed that the affine invariance property of barycentric coordinates is in fact *harmful* for shape deformation, as it introduces undesirable shears, see for example Figure 3.1. This is probably inevitable if the source region is forced to deform to fit precisely to the target region. Relaxing this constraint, Lipman et al. propose to work with *two* sets of barycentric coordinates, one for the vertices of the cage, and the second for the normals of its edges. The resulting deformation is shown to be conformal, although not interpolating. We show how to derive these coordinates as simple *complex* barycentric coordinates, and, in addition, show how to improve them.



Harmonic **Green**

Figure 3.1: Affine invariance of harmonic coordinates causes shearing, whereas Green coordinates generate a conformal map, better preserving the details.

In some scenarios, it is easier to control a deformation by using control points instead of a cage. The smaller number of control points leaves extra degrees of freedom, which may be used to impose constraints on the *derivative* of the deformation. We develop the derivatives of the Green coordinates, and show how to exploit them in a more user-friendly deformation setting.

The rest of this chapter is organized as follows. In the next section we introduce complex barycentric coordinates, and show examples of such coordinates – the discrete and continuous Cauchy-Green coordinates, and their generalization to a family of complex three-point coordinates. In Section 3.3, we modify the Cauchy-Green coordinates to be *variational* in the sense that the deformation attempts to minimize a given objective function, and introduce the Szegö and Point-to-Point Cauchy coordinates. Section 3.4 presents experimental results which show how these new coordinates out-perform current state-of-the-art methods for planar deformation. We conclude with a discussion of open issues in Section 3.5.

3.2 Complex barycentric coordinates

3.2.1 Definitions

Let $S = \{v_1, v_2, \ldots, v_n\} \subset R^2$ be the vertices of a simply connected planar polygon, oriented in the counter clockwise direction, $v_j = (x_j, y_j)$. Let $z_j = x_j + iy_j$ be the representation of the

vertices as complex numbers, with $i = \sqrt{-1}$, $z_j \in \mathbb{C}$. Denote by Ω the interior of S. Given a point $v = (x,y) \in \Omega$, define $z = x + iy$ and consider the following *complex* linear combination:

$$\sum_{j=1}^{n} k_j(z) z_j$$

where $k_j(z):\Omega \to \mathbb{C}$.

We say that the functions $k_j(z)$ are *complex barycentric coordinates* with respect to S if the following two properties hold for all $z \in \Omega$:

Constant precision:

$$\sum_{j=1}^{n} k_j(z) = 1 \qquad 3.1$$

Note that this implies that the real part of the coordinates sums to 1, and the imaginary part sums to 0.

Linear precision:

$$\sum_{j=1}^{n} k_j(z) z_j = z \qquad 3.2$$

Constant precision is sometimes called "reproduction of unity", and linear precision called "reproduction of the identity". Given complex barycentric coordinates $k_j(z)$ for S, we may consider the complex function $g_{S,F}(z)$ which results from applying the complex barycentric coordinates to the vertices of a target polygon $F = \{f_1, f_2, \ldots, f_n\} \subset \mathbb{C}$, as in Figure 3.2:

$$g_{S,F}(z) = \sum_{j=1}^{n} k_j(z) f_j \qquad 3.3$$

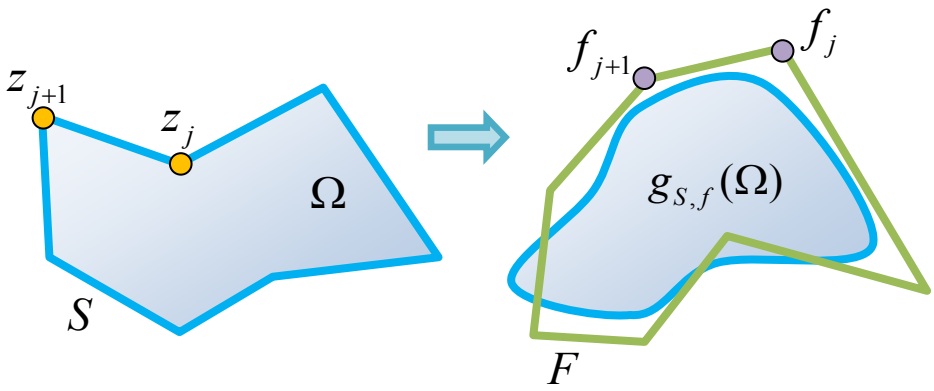


Figure 3.2: Planar mapping from Ω *- the interior of the polygon S, to* $g_{S,F}(\Omega)$ *using complex barycentric coordinates* $k_j(\Omega)$.

Note that F should also be simply connected and oriented counter-clockwise. The complex function g can be viewed as a planar mapping from Ω - the interior of S - to its image $g(\Omega)$. Note that $g(\Omega)$ will usually *not* be the interior of the polygon F. However, since the coordinates reproduce unity and the identity, they will reproduce any linear function of z. Recalling that a linear function of a single complex variable is equivalent to a similarity 2D transformation in the plane, we have:

Theorem 1: Complex barycentric coordinates reproduce similarity transformations. For a proof, see Appendix A.

This means that if $f_j = f(z_j)$ for some similarity transformation f, then $g_{S,F} = f$.

Note that although all complex barycentric coordinates reproduce similarity transformations, not all reproduce affine transformations. It is quite straightforward to see that:

Theorem 2 Complex barycentric coordinates $k_j(z)$ reproduce affine transformations if and only if the complex conjugates of the coordinates $k_j(z)$ also have linear precision (see Appendix A):

$$\sum_{j=1}^{n} \overline{k}_j(z) z_j = z$$

In particular, all real barycentric coordinates reproduce affine transformations. Somewhat counter-intuitively, the fact that most complex barycentric coordinates do not reproduce affine transformations is an advantage, as, in many applications, non-uniform scale is undesirable.

On top of not reproducing affine transformations, we depart from another standard property of real barycentric coordinates: we do not require the barycentric coordinates to be interpolating (sometimes called the *Lagrange* property of the coordinates). We expect the image $g(\Omega)$ to be close in some sense to the target polygon F, but do not impose the strict interpolation requirement: $g(z_j)=f_j$. In many cases this allows for more natural mappings.

Complex barycentric coordinates can be easily generalized to continuous contours in the following way. Let Ω be a simply connected open planar region with a smooth boundary S. Given $z \in \Omega$ and $w \in S$, consider the complex function $k(w,z):S\times\Omega\to\mathbb{C}$. Analogously to the discrete case, we say that $k(w,z)$ is a barycentric coordinate function if it satisfies the following properties for all $z\in\Omega$:

<u>*Constant precision:*</u>

$$\oint_S k(w,z)dw = 1 \tag{3.4}$$

<u>*Linear precision:*</u>

$$\oint_S k(w,z)wdw = z \tag{3.5}$$

The function $k(w,z)$ is sometimes called a *kernel* function. The main difference between this and the continuous definition of real barycentric coordinates [Bel06, WSHD07] is that here the integral over S is a *complex* integral, where $dw = T(w)ds$. $T(w)$ is the unit-length tangent vector to S at w, and ds is the usual arc-length differential element. See Figure 3.3.

Analogously to (3.3), given a continuous complex function $f(S)$: $S \to \mathbb{C}$, we can define a planar mapping $g_{S,f}(\Omega)$ as follows:

$$g_{S,f}(z) = \oint_S k(w,z) f(w) dw$$

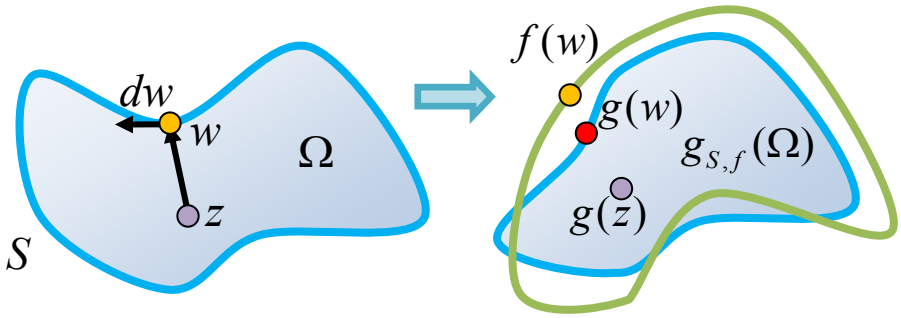


Figure 3.3: Continuous planar mapping from Ω - the interior of the closed curve S - to gS,f(Ω) using the complex barycentric kernel k(S×Ω).

As in the case of real barycentric coordinates, the main challenge is to find kernels $k(w,z)$, or, in the discrete case, coordinate functions $k_j(z)$, which satisfy the required properties. Next we will show how a simple complex kernel can be used both in the continuous and discrete settings to obtain useful barycentric coordinates, and how to generalize this kernel to a family of discrete complex three-point-coordinates.

3.2.2 Continuous Cauchy coordinates

Consider the complex function:

$$C(w,z) = \frac{1}{2\pi i} \frac{1}{w-z}$$

C is well known from complex analysis, where it is called the *Cauchy kernel* [Bel92]. C satisfies the two properties (3.4) and (3.5), namely:

$$\frac{1}{2\pi i} \oint_S \frac{1}{w-z} dw = 1; \qquad \frac{1}{2\pi i} \oint_S \frac{w}{w-z} dw = z; \;\; z \in \Omega$$

because these two identities are special cases ($h(w)=1$ and $h(w)=w$) of Cauchy's integral formula [Ahl79], which asserts that the values of a function on the boundary of a simply-connected region determine its value at every point inside the region:

$$\frac{1}{2\pi i}\oint_S \frac{h(w)}{w-z}dw = h(z) \qquad 3.6$$

Cauchy's integral formula holds for the class of complex functions known as *holomorphic functions*. Such functions are the linear subspace of "well behaved" complex functions, and also have a geometric interpretation – a holomorphic function whose first derivative does not vanish is a conformal mapping. See Ahlfors [Ahl79] for a detailed introduction to holomorphic functions. So, in fact, the Cauchy kernel reproduces *all holomorphic functions*. We call the resulting coordinates *Cauchy coordinates.*

Applying the Cauchy coordinates to a target contour $f(S)$ defines the following mapping:

$$g_{S,f}(z) = \frac{1}{2\pi i}\oint_S \frac{f(w)}{w-z}dw \qquad 3.7$$

Note, that equations (3.6) and (3.7) are very different, as h in (3.6) is a holomorphic function defined on S and on Ω, and f in (3.7) is a function defined *only* on S.

The mapping $g(\Omega)$ in (3.7) is sometimes called the *Cauchy transform* of f [Bel92]. It has various interesting properties, one of which being that if f is continuous on S, then g is always holomorphic on Ω [Bel92, Theorem 3.1]. Hence, if we apply these coordinates in the context of planar shape deformation, the deformation is guaranteed to be conformal (if the derivatives do not vanish). In addition, since holomorphic functions are infinitely differentiable, the mapping will be smooth.

3.2.3 Discrete Cauchy-Green coordinates

In a practical shape deformation scenario, the contour S is usually a polygon (sometimes called "cage") which the user deforms to a new polygon F, as in Figure 3.2. Let us now consider what (3.7) reduces to when S is a polygon $S = \{z_1, z_2, \ldots, z_n\}$. Although S does not have a tangent vector at z_i, (3.6) is still valid, by applying it to each edge $e_j = (z_{j-1}, z_j)$ separately:

$$g_{S,f}(z) = \frac{1}{2\pi i} \sum_{j=1}^{n} \int_{e_j} \frac{f(w)}{w - z} dw$$

Since F is also a polygon, f maps each edge of S linearly to an edge of F. Hence, for $w \in (z_{j-1}, z_j)$:

$$f(w) = f_{j-1} + \frac{(f_j - f_{j-1})(w - z_{j-1})}{(z_j - z_{j-1})}$$

Computing the integral on a single edge e_j:

$$\int_{e_j} \frac{f(w)}{w - z} dw = \log \frac{B_j(z)}{B_{j-1}(z)} \left(f_{j-1} \frac{B_j(z)}{A_j} - f_j \frac{B_{j-1}(z)}{A_j} \right) + f_j - f_{j-1}$$

where $B_j(z) = z_j - z$ and $A_j = z_j - z_{j-1}$, as in Figure 3.4. Summing over all edges, and rearranging the terms yields:

$$g_{S,f}(z) = \sum_{j=1}^{n} C_j(z) f_j$$

$$C_j(z) = \frac{1}{2\pi i} \left(\frac{B_{j+1}(z)}{A_{j+1}} \log \left(\frac{B_{j+1}(z)}{B_j(z)} \right) - \frac{B_{j-1}(z)}{A_j} \log \left(\frac{B_j(z)}{B_{j-1}(z)} \right) \right) \qquad 3.8$$

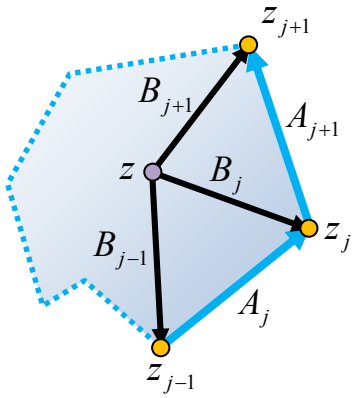


Figure 3.4: Notations for the Cauchy-Green coordinates.

We call $C_j(z)$ *discrete Cauchy coordinates* and g the *discrete Cauchy transform* of f. Note that we have defined $C_j(z)$ only on the interior of the polygon, since on the boundary the

expression may be singular. See Figure 3.5 for an example of a mapping of a polygon generated by discrete Cauchy coordinates. Note that the image of the polygon is not a polygon. The mapping is just a linear combination of the n holomorphic coordinate functions $C_j(z)$, one of which is visualized in the figure.

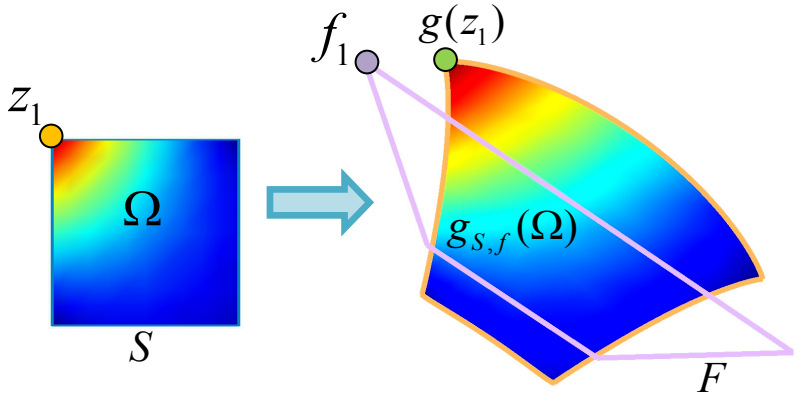


Figure 3.5: Example of a mapping of source polygon S, guided by target polygon F, using the discrete Cauchy coordinates. The result is the region $g_{S,F}$. The real part of $C_1(z)$ is visualized using color-coding both on the source polygon and on $g(\Omega)$.

The discrete Cauchy transform has a number of desirable properties. First, like the continuous Cauchy transform, the resulting function g is holomorphic, and infinitely differentiable. Hence, the mapping from Ω to $g(\Omega)$ is conformal (if the derivative of g does not vanish). In fact, not only that the discrete Cauchy transform possesses a closed form, expressions for the derivatives of the transform are easily obtained (see Appendix C, Appendix D and Appendix E for the exact formulas of the first, second and third derivatives). This demonstrates the advantage of using complex numbers formulation over real numbers.

As it turns out, the discrete Cauchy coordinates have already been discovered, albeit in a different guise.

Theorem 3: Lipman's 2D Green coordinates [LLCO08] are identical to discrete Cauchy coordinates.

Proof: The 2D Green coordinates are defined in the following way:

$$g(\eta) = \sum_{k \in V} \phi_k(\eta) v_k + \sum_{j \in V} \psi_j(\eta) n(t_j)$$

where v_k and t_j are the vertices and edges of the cage, respectively, and $n(t_j)$ are the un-normalized normals to the edges. The coordinate functions ϕ_k and ψ_j are the closed form integrals present in Green's third identity, and have somewhat complicated expressions. Denoting by z_k the complex representation of the cage vertices v_k, we get:

$$g(\eta) = \sum_{k \in V} \phi_k(\eta) z_k + \sum_{j \in V} \psi_j(\eta) i (z_{j+1} - z_j)$$

since the un-normalized normal to an edge is just the edge rotated by $\pi/2$, which is equivalent to multiplication by i in the complex plane. Rearranging terms we have:

$$g(\eta) = \sum_{j \in V} \left(\phi_j(\eta) + i \left(\psi_{j-1}(\eta) - \psi_j(\eta) \right) \right) z_j$$

At this point, we can plug in the formulas for ϕ_j and ψ_j given in [LLCO08] and simplify the resulting expression. The derivation is extremely technical, but following it through results in (3.8) – the discrete Cauchy coordinates. ♦

It should actually not come as a surprise that the discrete Cauchy coordinates and the Green coordinates are equivalent. One is derived from Cauchy's integral formula, and the second from Green's third identity. Since these are known to be equivalent (in the sense that one can be derived from the other [Ahl79]), the connection between the two is inevitable. Thus, in the sequel, we will refer to these coordinates as the *Cauchy-Green* coordinates.

In the next section, we will consider a general family of complex barycentric coordinates analogous to the "three-point coordinates" family defined in [FHK06].

3.2.4 Complex three-point coordinates

Floater *et al.* [FHK06] described a family of barycentric coordinates such that the coordinate k_j depends only on the points v_{j-1}, v_j, v_{j+1}, and showed that known planar barycentric coordinates, such as the mean-value, Wachpress and discrete harmonic coordinates are all members of this family. It turns out that a similar classification can be made for complex barycentric coordinates, and that the Cauchy-Green coordinates are a member of this family.

Theorem 4: Let $m_j(z):\Omega\to\mathbb{C}$, $j = 1,..,n$ be complex functions, and let $B_j(z) = z_j - z$ and $A_j = z_j - z_{j-1}$, as in Figure 3.4. Then the functions:

$$k_j(z) = m_j(z)\frac{B_{j+1}(z)}{A_{j+1}} - m_{j-1}(z)\frac{B_{j-1}(z)}{A_j}$$

satisfy:

$$\sum_{j=1}^{n} k_j(z)(z_j - z) = 0$$

Proof: The key to the proof, and one of the differences between complex and real coordinates, is the fact that an arbitrary complex number z can be expressed uniquely using a complex affine combination of only *two* other points, while an arbitrary 2D vector requires an affine combination of *three* points. Specifically, given a complex number z, and two other complex numbers z_j and z_{j+1}, there exist complex numbers $\alpha_j(z)$ and $\beta_j(z)$ such that:

$$z = z_j\beta_j(z) + z_{j+1}\alpha_j(z)$$
$$\alpha_j(z) + \beta_j(z) = 1$$

Finding $\alpha_j(z)$ and $\beta_j(z)$ is a matter of solving these two linear complex equations in two variables.

It is easy to check that the solutions are:

$$\alpha_j(z) = \frac{z - z_j}{z_{j+1} - z_j}, \quad \beta_j(z) = \frac{z_{j+1} - z}{z_{j+1} - z_j}$$

Following Floater *et al.*, [FHK06], we can write:

$$D_j(z) = \beta_j(z)(z_j - z) + \alpha_j(z)(z_{j+1} - z) \equiv 0$$

Now, any linear combination of $D_j(z)$ vanishes, hence:

$$\sum_{j=1}^{n} m_j(z)D_j(z) = 0$$

Plugging back $\alpha_j(z)$ and $\beta_j(z)$, and rearranging the terms:

$$\sum_{j=1}^{n}\left(m_j(z)\frac{B_{j+1}(z)}{A_{j+1}} - m_{j-1}(z)\frac{B_{j-1}(z)}{A_j}\right)(z_j - z) = 0 \qquad 3.9$$

which concludes the proof.

Given a set of functions $k_j(z)$ as in Theorem 4, whose sum is non-zero for all z, it is straightforward to see that the functions $w_j(z) = k_j(z)/\sum k_j(z)$ have the constant precision and linear precision properties, and hence are by definition complex barycentric coordinates.

In fact, using a proof which is almost identical to the one in [FHK06], we can show that the converse of Theorem 4 is also true:

Theorem 5: Any set of complex functions $k_j(z)$ which satisfy

$$\sum_{j=1}^{n} k_j(z)(z_j - z) = 0$$

can be expressed in the form (3.9). For a proof, see Appendix A.

The family of complex three point coordinates is generated by restricting $m_j(z)$ to be a complex function of only $B_j(z)$ and $B_{j+1}(z)$.

Comparing the expression for the discrete Cauchy-Green coordinates from (3.8) to the expression in (3.9), we can see that the discrete Cauchy-Green coordinates are members of the complex three-point coordinate family, with:

$$m_j(z) = \frac{1}{2\pi i}\log\frac{B_{j+1}(z)}{B_j(z)} = \frac{1}{2\pi i}\int_{z_j}^{z_{j+1}}\frac{1}{w-z}\,dw$$

So far, we have introduced complex barycentric coordinates, and showed how to easily derive the planar Cauchy-Green coordinates in this context. In the next section, we will demonstrate that the complex point of view not only gives insight into existing barycentric coordinates, but also allows us to generate new coordinates which perform better than state-of-the-art algorithms for planar shape deformation.

3.3 Cauchy-type coordinates and shape deformation

In planar shape deformation applications, the user deforms a shape, which is usually a planar mesh or an image. In cage-based applications, the user draws a "cage" around the shape of interest, and modifies the shape by deforming the cage. Two main requirements must be met for an algorithm to be practical – the mapping should preserve as much as possible the details of the shape or image, and the deformation should be fast enough to be run interactively.

Since conformal maps preserve angles and the shape of small details, they are good candidates for detail-preserving deformations. Barycentric coordinates are extremely fast to compute – the complexity of the computation for a single point in the deformed domain depends only on the complexity of the cage, which is usually significantly smaller than the complexity of the deformed shape. Thus complex barycentric coordinates which produce conformal maps could provide the best of both worlds and be a very useful tool for planar shape deformation.

Unfortunately, in general, given two arbitrary polygons with corresponding vertices, there does not exist a conformal mapping which maps the corresponding edges linearly to each other. Thus, to achieve conformality we must relax the interpolation requirement. Luckily, the space of conformal mappings from a given source polygon to the region close to another given target polygon is quite large, so we can add an additional requirement – find the complex barycentric coordinate functions which give a conformal mapping *and* minimize a functional of our choice.

As described in Section 3.2.2, the Cauchy transform takes as input a continuous function f on a contour S, and outputs g, a function holomorphic on the interior of S, see Figure 3.3. Thus the Cauchy transform can also be interpreted as a *projection* from the linear subspace of continuous complex functions on $S \cup \Omega$, to the linear subspace of holomorphic functions on $S \cup \Omega$. It is a projection because of its holomorphic function reproduction property – if f is holomorphic on $S \cup \Omega$, then $g=f$ on Ω. An interesting question is whether the Cauchy transform is an *orthogonal projection*, meaning that the resulting holomorphic function g is the *closest* holomorphic function, in some metric, to the given function f. For the metric derived from the following inner product of two complex functions of a complex variable:

$$< f, g >= \oint_S (f \cdot \bar{g}) ds$$

namely, the metric that measures goodness of fit of the image of the source contour $g(S)$ to the target contour $F = f(S)$, the answer to this question is negative – the Cauchy projection is an *oblique projection*, and given an arbitrary function f there exists a holomorphic function whose boundary values are closer to f than $g=Cf$. This, in general, is not good news – it means the function generated by the Cauchy transform is indeed holomorphic, but it is not as close as it could be to the user's specification. But there are also good news: the orthogonal projection of f, also known as the *Szegö projection* [Bel92], onto the space of holomorphic functions, is relatively well-known, and computable.

Since we have the closed form (3.8) for the discrete Cauchy-Green coordinates, we would like to stay within the n-dimensional subspace of holomorphic functions spanned by these coordinate functions, which we call the *Cauchy-Green subspace*, yet produce a better-quality deformation. More formally:

Problem 1: Given a source polygon S with n vertices and interior Ω, and a functional $E_S(g)$ defined on holomorphic functions g: $\Omega \cup S \rightarrow \mathbb{C}$, find complex numbers $u_1,..,u_n$ such that:

$$g_u(z) = \sum_{j=1}^{n} C_j(z) u_j$$

minimizes $E_S(g)$ among all possible choices of u.

We can think of solving Problem 1 as finding a *virtual target polygon u*, whose discrete Cauchy transform minimizes $E_S(g)$.

Next, we describe two choices for such a functional E, that can be applied to achieve useful effects in the context of planar shape deformation.

3.3.1 Szegö coordinates

As mentioned in the previous section, the main disadvantage of the Cauchy-Green coordinates is that the resulting deformation might be far from the target cage, see for

example Figure 3.5. An obvious improvement would result if we could find a better fit to the target cage within the Cauchy-Green subspace. Inspired by the Szegö projection, we define the following functional, which will help us generate a better fit.

Let f be a continuous complex function on the source polygon S. Define:

$$E_S^{Szegö}(g) = \oint_S \left| g(w) - f(w) \right|^2 ds \qquad 3.10$$

We would like to minimize (3.10) within the Cauchy-Green subspace, which means, in other words, finding the virtual polygon u of Problem 1. However, so far we have only defined the Cauchy-Green coordinate functions on the *interior* of S, and not on S itself, where they are singular. Since we are now interested in the boundary values of the transformation image, we define the values of the coordinate functions on S to be their limit when approaching the boundary from the interior Ω:

$$C_j(z) = \lim_{z^{in} \to z} C_j(z^{in}), \quad z^{in} \in \Omega,\ z \in S$$

These limits always exist. We refer to Appendix A for a proof. The resulting formulae can be found in Appendix B.

Having defined C_j also on S, we may now rewrite (3.10) as

$$E_S^{Szegö}(g) = \oint_S \left| g(w) - f(w) \right|^2 ds = \oint_S \left| \sum_{j=1}^{n} C_j(w) u_j - f(w) \right|^2 ds$$

To solve the problem in practice, we approximate the integral as a sum over a k-sampling of S. This sample may be expressed as a product of the n-vector z of the vertices of S with a $k \times n$ sampling matrix H, such that $w = Hz$ is a complex k-vector of points sampled on the polygon S. Now $C_j(w)$, $f(w)$ are also complex k-vectors (by evaluating the respective function at the entries of w), and we can express the functional in matrix form:

$$E_S^{Szegö}(g) = \left\| Cu - f_s \right\|_2^2$$

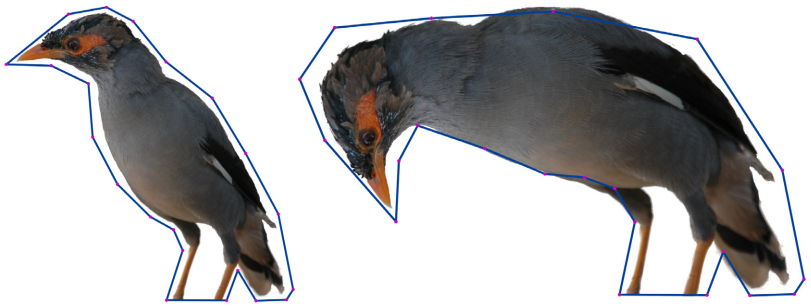
C is the complex $k \times n$ matrix whose columns are $C_j(w)$, namely, the values of the Cauchy-Green coordinate function on the sampled boundary, u is a complex n-vector and f_s a complex k-vector whose entries are $f(w)$. This is a simple linear least-squares problem over the complex numbers, and its solution is known to be [Bjö96]:

$$u^{Szegö} = C^{+} f_s = (C^{*}C)^{-1} C^{*} f_s \qquad 3.11$$

where C^{+} is the pseudo-inverse of C and C^{*} is the conjugate transpose of C. Note that the size of the matrix $C^{*}C$ is $n \times n$, where n is the number of the vertices of the polygon S, hence this computation involves the inversion of a very modest sized matrix. Now that we have the virtual polygon $u^{Szegö}$, we define the deformation of an interior point $z \in \Omega$ to be:

$$g^{Szegö}(z) = \sum_{j=1}^{n} C_j(z) u_j^{Szegö} \qquad 3.12$$

Figure 3.6 shows an example of applying the Szegö coordinates for image deformation. As is evident from the image, the mapping is detail preserving (conformal), and the deformed shape remains close to the target cage.



Source **Szegö**

Figure 3.6: Deformation of a bird using Szegö coordinates. The cage has 21 vertices.

When working with deformation applications, time complexity is an issue, and we would like to avoid computing the virtual polygon $u^{Szegö}$ every time the user modifies the target function f. Fortunately, if the target function f is itself a polygon F, the deformation (3.12) can be made more compact by formulating it with barycentric coordinates applied to F.

Let $F = \{f_1, f_2, \ldots, f_n\}$ be the target polygon, then its sampled version f_s can also be obtained using the sampling matrix H: $f_s=HF$. Then:

$$u^{Szegö} = C^{+}HF$$

Thus the deformation $g_{S,f}$ is defined in terms of the *discrete Szegö coordinates* $G_j(z)$ of an interior point $z \in \Omega$:

$$g_{S,f}(z) = \sum_{j=1}^{n} G_j(z) f_j$$

$$G_j(z) = \sum_{k=1}^{n} C_k(z) M_{k,j} \quad , \qquad M = C^{+}H$$

M is an $n \times n$ matrix, called the *Szegö correction matrix*. It depends *only* on the source polygon S, thus may be computed once. Figure 3.7 shows the color coding of the real and imaginary parts of the one of the Szegö coordinates from Figure 3.6.

It is relatively easy to see that M has a number of useful properties, such as its rows sum to unity and $Mz=z$ (where z is a complex n-vector of the vertices of S), which imply that the Szegö coordinates also have constant and linear precision (for a detailed proof see Appendix A). Note that each Szegö coordinate function is a linear combination of *all* the Cauchy-Green coordinate functions. Thus they depend on *all* the vertices of the polygon S, so cannot be local three-point coordinates like the Cauchy-Green coordinates.

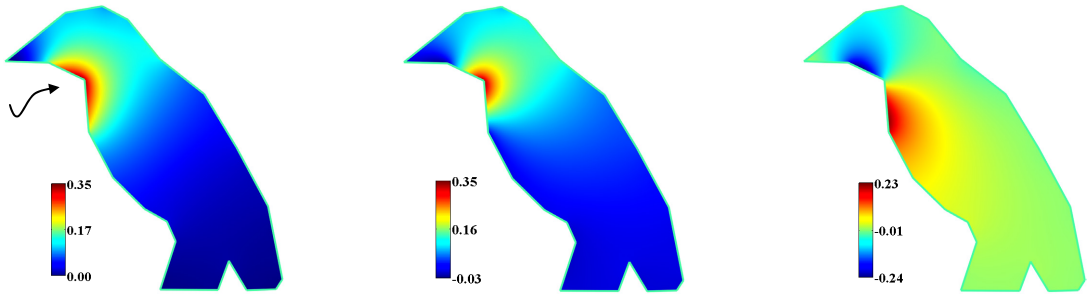


Figure 3.7: Absolute, real and imaginary component values of the Szegö coordinate function of the marked point. Note that the real part is centered at the vertex and the imaginary part at the two adjacent edges.

Next, we consider a different deformation paradigm which leads to another easily-minimized functional E.

3.3.2 Point-to-point Cauchy coordinates

The complex barycentric coordinates that we have discussed so far are "cage-based" deformations. The user modifies the location of the target cage vertices and thus controls the deformation. Unfortunately, in practice, the cage of a complicated shape can contain hundreds of vertices, and modifying each vertex independently to form the new cage is a time-consuming and unintuitive operation.

A much more intuitive user interface for the modeler will be manipulating a small number of positional constraints – control points - not necessarily on the shape boundary. This way the tedious task of manually positioning the cage vertices can be replaced by an efficient optimization process which derives suitable cage vertex positions automatically.

We now show how to easily adopt such an interface, while maintaining all the nice deformation properties that the Cauchy-Green coordinates provide. As the number of control points is typically much smaller than the number of cage vertices, it is possible to use the extra degrees of freedom to *regularize* the deformation. This means minimizing some aggregate differential quantity. Luckily, the Cauchy-Green coordinates have very simple derivatives and, like the transform itself, will also be a complex linear combination of the cage vertices. Hence we can easily minimize a functional combining both positional constraints inside the cage, and derivatives on the boundary of the cage.

Let f be a mapping from a set of p points $r_1, r_2, \ldots, r_p \in \Omega$, to the complex plane C, such that $f(r_k) = f_k$, and let:

$$E_S^{Smooth}(g) = \int_S \left| g''(w) \right|^2 ds \quad , \quad E_S^{Pts}(g) = \sum_{k=1}^{p} \left| g(r_k) - f_k \right|^2$$

Minimizing the first functional requires the mapping g to be as smooth as possible on the boundary of the cage. Minimizing the second functional imposes a finite set of positional constraints on the deformation in the interior of S.

Define the following combined weighted functional:

$$E_S^{PtoP}(g) = E_S^{Pts}(g) + \lambda^2 E_S^{Smooth}(g)$$

for some real λ. We will now attempt to solve Problem 1 for this functional.

Using the usual rules of linearity, the second derivative of the discrete Cauchy-Green transform (3.8) is:

$$g''(z) = \sum_{j=1}^{n} d_j(z) z_j$$

where

$$d_j(z) = \frac{1}{2\pi i}\left(\frac{1}{B_{j-1}(z)B_j(z)} - \frac{1}{B_j(z)B_{j+1}(z)} \right) \qquad 3.13$$

As was the case for the discrete Szegö coordinates, E_S^{Smooth} is defined on the boundary of Ω, where d_j is singular when z is on the edge $e_j = (z_{j-1}, z_j)$, or on the edge $e_{j+1} = (z_j, z_{j+1})$. So here too, we need to use the limits of d_j for these cases:

$$d_j(z) = \lim_{z^{in} \to z} d_j(z^{in}),\ z^{in} \in \Omega,\ z \in S$$

It turns out that the limits everywhere except at the vertices are equal to $d_j(z)$, hence we can use (3.13) as is:

$$E_S^{smooth}(g) = \oint_S \left| g''(w) \right|^2 ds = \int_S \left| \sum_{j=1}^{n} d_j(w) u_j \right|^2 ds$$

As with the Szegö coordinates, we can rewrite the combined functional in matrix form:

$$E_S^{PtoP}(g) = \left\| Cu - f \right\|_2^2 + \lambda^2 \left\| Du \right\|_2^2$$

where D is a complex $k \times n$ matrix whose columns are $d_j(w)$. As in Section 3.3.1, C is the $p \times n$ matrix whose (i,j) entry is $C_j(r_i)$, u is a complex n-vector and f a complex k-vector whose entries are the positional constraints f_k. Some algebra leads to the Point-to-Point Cauchy-Green barycentric coordinates:

$$g_{S,f}(z) = \sum_{j=1}^{p} P_j(z) f_j$$

$$P_j(z) = \sum_{k=1}^{n} C_k(z) N_{k,j} \quad , \quad N = \left(A^{+} = \begin{pmatrix} C \\ \lambda D \end{pmatrix}^{+} \right)_{1..p}$$

We now have only p barycentric coordinate functions - P_j. $A^{*}A$ is an $n \times n$ matrix which can be easily inverted. N is the $n \times p$ matrix consisting of the first p columns of A^{+}. Using properties of N, it is relatively easy to show that the point-to-point Cauchy-Green coordinates have constant and linear precision (we refer the reader to Appendix A for a detailed proof).

Figure 3.8 shows an example of a deformation produced by the P2P Cauchy-Green coordinates. As promised, the deformation is conformal, yet much easier to control since the user needs to manipulate only a small number of control points.

The number of coordinate functions P_j is now p - the number of control points, rather than n - the number of cage vertices (even though an n-vertex cage implicitly participates in the process). These functions are centered around the control points. Figure 3.9 shows the color-coded values of one of the coordinates functions of the shape of Figure 3.8. As with the Szegö coordinates, the influence of a coordinate function associated with a specific control points is local, in the sense that it affects only the areas of the cage which are geodesically close to it. This property guarantees that modifying one part of the shape will not affect another part, even if the *Euclidean* distance between them is small.

Figure 3.8: A giraffe, and its deformation using Point-to-point Cauchy-Green coordinates, with 16 control points. The cage has 113 vertices.

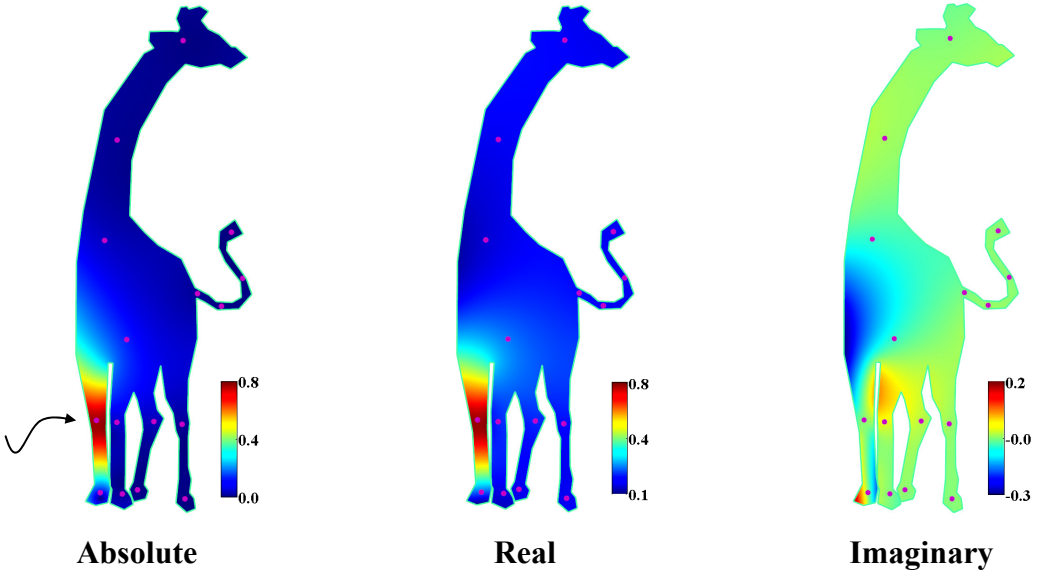


Figure 3.9: P2P coordinate function of the marked point.

3.4 Experimental Results

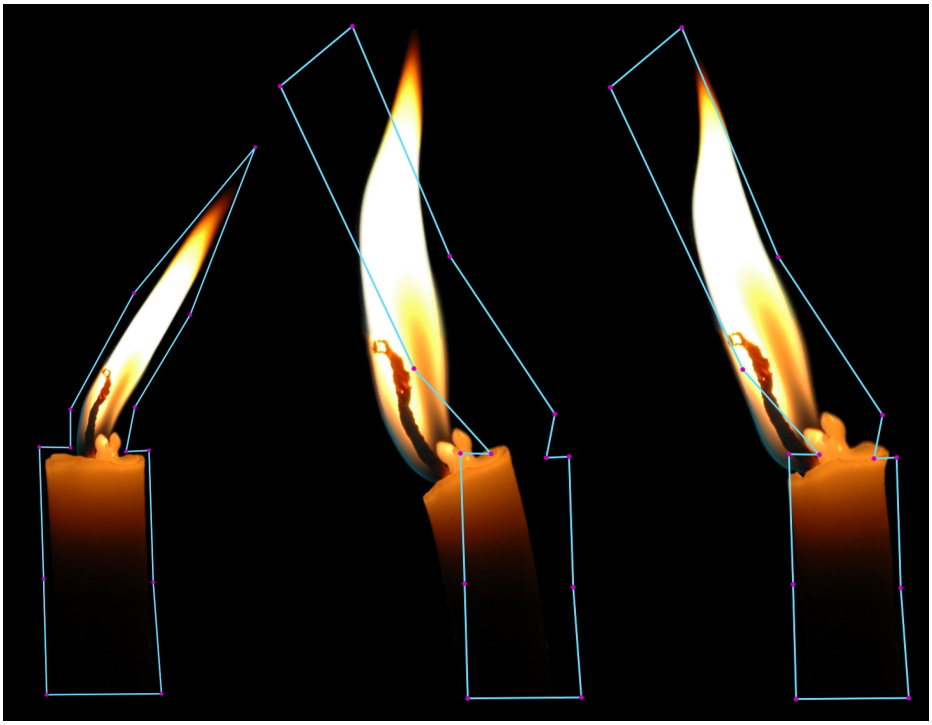
We have implemented an image deformation system using the discrete Szegö coordinates and the point-to-point Cauchy-Green coordinates as plugins to the Maya® commercial modeling

and animation system. We compared their performance to that of existing state of the art planar deformation algorithms – the original cage-based Cauchy-Green coordinates [LLCO08] and control-point-based MLS [SMW06]. The image was represented as a texture map on a triangulation of the 2D domain, containing m vertices. Each of the n barycentric coordinates were pre-computed on these m vertices and stored in a dense complex $m \times n$ matrix B. A typical cage contains about n=150 vertices and m=15,000 interior vertices. The pre-process time to compute B is less than 10 seconds. The serial runtime complexity of a subsequent deformation operation is $O(mn)$ – the time required to multiply B by a complex n-vector. However, this multiplication was implemented in the GPU using Nvidia's CUDA programming language on an Nvidia Geforce 8800 GTX graphics card, resulting in a very significant speedup. For example, a single deformation of the "lady with whip" image (Figure 3.12), which has m=12,000, n=272 and p=26 control points, takes approximately 0.05 milliseconds. Moreover, the pre-process time can be significantly reduced by implementing it also on the GPU, but we have not yet done this.

Before starting our qualitative comparison, we state upfront a few downsides of our coordinates. First and foremost, we currently do not have an extension to 3D deformations, as the original Green coordinates have. In addition, our coordinates generate only conformal deformations, and we do not currently support "as rigid as possible"-type deformations, as the MLS method does. Despite these shortcomings, we believe that our coordinates are useful in many 2D deformation scenarios, as we will now demonstrate.

3.4.1 Szegö vs. Cauchy-Green

As was already mentioned in previous sections, the main disadvantage of the Cauchy-Green coordinates is that the image of the domain, $g(\Omega)$, might be quite far from the target contour F. On the other hand, the Szegö coordinates produce (by definition) the holomorphic function in the Cauchy-Green subspace, whose boundary values are closest to the target polygon.



Source **Cauchy-Green** **Szegö**

Cauchy-Green **Szegö**

Figure 3.10: Comparison of the Cauchy-Green coordinates and the Szegö coordinates. The Szegö coordinates better fit the cage, preventing an undesirable bend of the candle, and an undesirable movement of the rose leaves.

Figure 3.10 compares the deformation of some images using the Cauchy-Green coordinates and the Szegö coordinates. It can easily be seen that the Szegö coordinates provide a better

match to the target contour. See, in addition, the video that accompanies the work of [WBCG09] for a live demo of interactive deformations using both methods. There the Szegö coordinates can also be seen to be more stable during deformation.

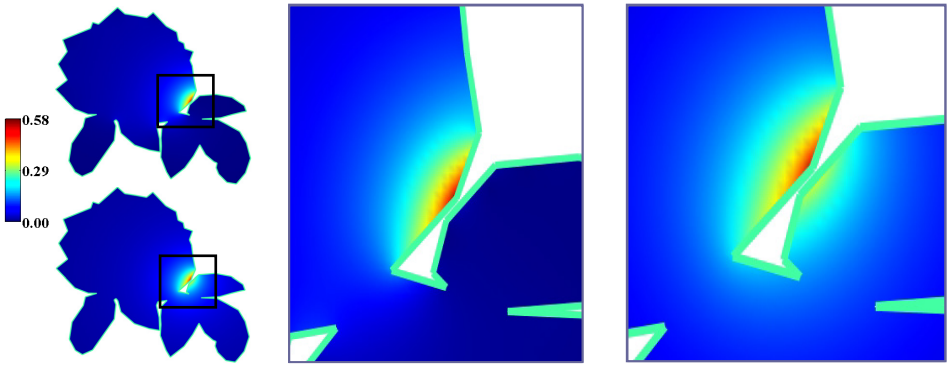


Figure 3.11: Absolute value of the Szegö and Cauchy-Green coordinates. The Cauchy-Green coordinate "spills" into the leaf near it, whereas the Szegö coordinate does not.

In addition to better fitting the contour, the Szegö coordinates have another important property – they are local within the cage. Figure 3.11 shows an example of the absolute value of the discrete Cauchy coordinates vs. the discrete Szegö coordinates on a given polygon. As is obvious from the image, the effect of the Szegö coordinates is more local in comparison to that of the Cauchy coordinates, for which deformation of one part of the shape may influence other parts which are geodesically far away.

The computation of the correction matrix M is done in the preprocess step, after the source cage is defined but before the actual deformation. Thus the runtime complexity of the preprocess step depends on k – the source contour sample density, but the runtime complexity of the actual deformation is exactly the same as that of the Cauchy-Green coordinates.

3.4.2 Point to point Cauchy-Green vs. MLS

In some scenarios cage deformations are not very useful. If the cage is complicated, as is usually the case for real-life shapes, cage-controlled deformations are less intuitive than a small number of simple control points strategically placed in the domain. This interface was also used by Igarashi *et al*. [IMH05] and the MLS system [SMW06].

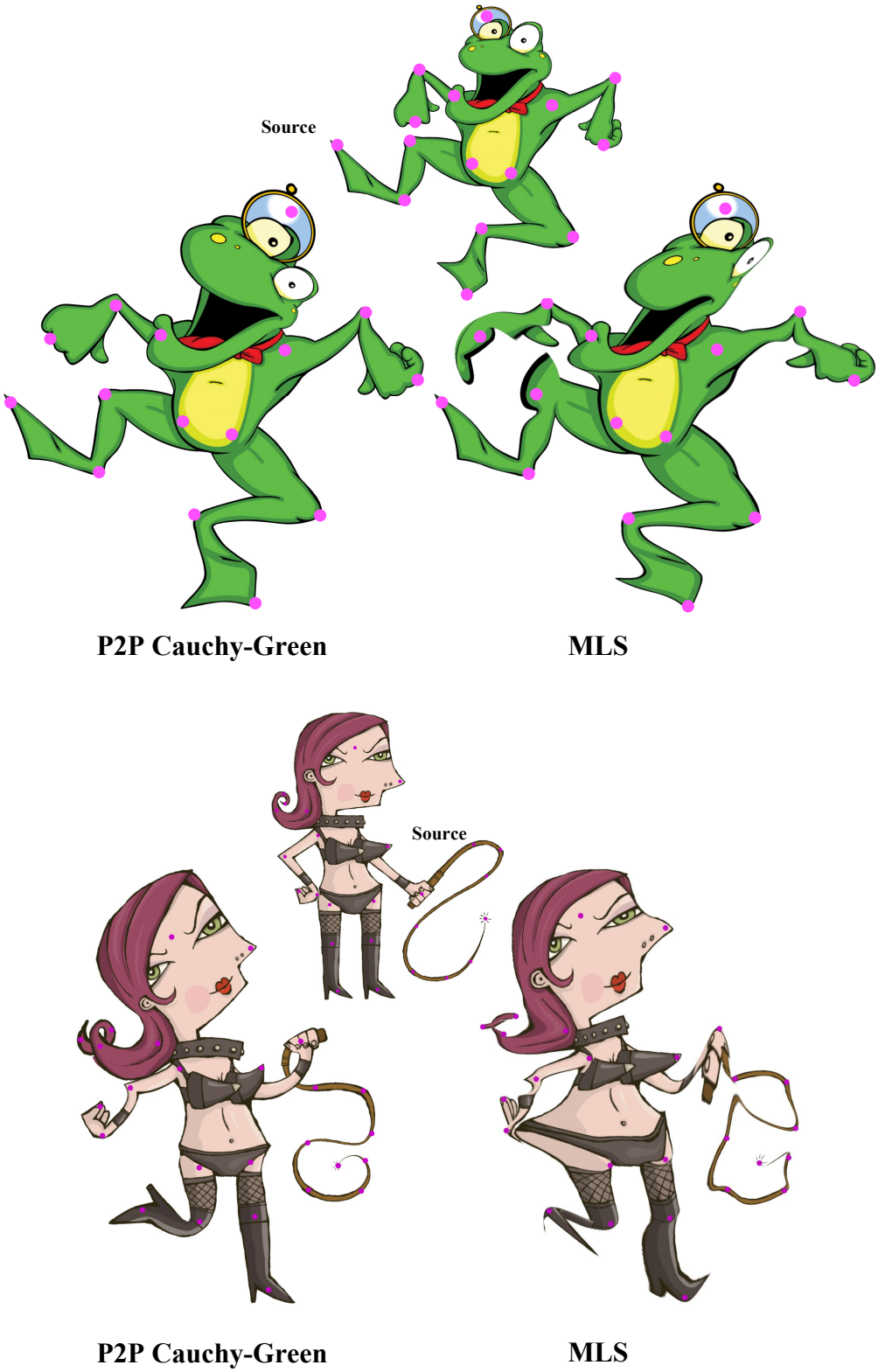


Figure 3.12: Comparison of the P2P Cauchy-Green coordinates and the MLS coordinates. The P2P coordinates better handle control points whose Euclidean distance is small, yet their geodesic distance within the cage is large.

Specifically, the MLS approach allows to define a set of control points and finds a local as-similar-as-possible or as-rigid as-possible deformation which satisfies the positional constraints imposed by the control points. As the point-to-point Cauchy-Green approach produces a holomorphic function by definition, the most relevant comparison is to the similarity version of MLS. Close examination of the MLS equations reveals that there exist complex barycentric coordinates which are equivalent to the similarity version.

Figure 3.12 and the video that accompanies the work of [WG10] compare some deformations generated by both methods with exactly the same constraints. The results show that point-to-point Cauchy-Green coordinates do much better at preserving the geometry of the shape. Using the additional information – the cage – this method is better at separating the extremities – e.g the hand of the frog from its leg. Since the MLS method deforms the entire plane, ignoring the geometry of the shape, separating the hand of the frog from its foot results in serious artifacts. On the other hand, as opposed to the MLS method, the point-to-point Cauchy-Green coordinates do not satisfy the positional constraints imposed by the control points precisely, rather treat them as soft constraints. This can be alleviated, if needed, by reducing the value of λ in the optimization problem of Section 3.3.2, albeit at the expense of the mapping smoothness.

As stated above, the MLS deformation can also be formulated as complex barycentric coordinates centered around the control points (see Appendix A). Obviously, since the MLS has no knowledge of the cage, the coordinates' effect depends on Euclidean distances. This is clearly seen in Figure 3.13 which compares the absolute values of the MLS coordinates and the point-to-point Cauchy-Green coordinates. Of course, it might be possible to modify MLS to use geodesic distances instead of Euclidean distances, but this would require discretization of the interior of the cage, which we wish to avoid.

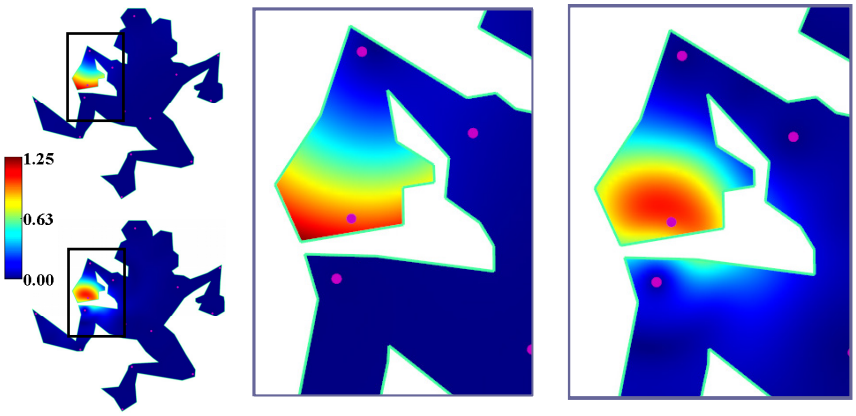


Figure 3.13: Absolute value of the P2P and MLS coordinates of the point on the left hand. The MLS coordinate "spills" into the leg near it, whereas the P2P coordinate does not.

The time complexity of deformation using P2P Cauchy-Green coordinates is very similar to that of the Szegö coordinates, as most of the computation is done in the pre-process step. Here, however, we have an additional benefit – the complexity of the deformation during user interaction is proportional to p - the number of control points, as opposed to n - the number of cage vertices, which is typically much larger.

3.5 Conclusions and Discussion

We have generalized the concept of barycentric coordinates from real numbers to complex numbers, and provided a few examples of known and new coordinates which can be expressed quite simply in this framework. In addition, we have shown how the new coordinates can be used in the context of planar shape deformations to produce results superior to state-of-the-art methods, at a small extra cost in computational complexity, in pre-process time only.

We believe there is still much research to be done on the theory and applications of complex barycentric coordinates. Their most obvious drawback is the fact that they are only defined for planar domains, which would seem to rule out the possibility of using them for 3D shape deformation. However, this should still be possible if the complex numbers are interpreted geometrically and analogs found in higher dimensions.

Another important research direction is relaxing the conformality requirement. As real barycentric coordinates are a special case of complex coordinates, obviously non-conformal complex coordinates exist. We have seen that the MLS coordinates are such. An interesting challenge is to find non-conformal complex coordinates which will generate the more useful quasi-conformal or "as rigid as possible"-type deformations.

There are some connections between our method and other deformation methods. First, the formulation of the Green Coordinates is a special case of the Boundary Element Method formulation, where constant boundary elements are used. Second, the functional minimized for the P2P coordinates has some resemblance to functionals used with Radial Basis Functions. As both methods were previously used for shape warping and deformation [JP99,Boo89], it would be interesting to explore the relationship between these methods and ours.

One theoretical issue which we haven't addressed at all, is the connection between complex barycentric coordinates and the so-called "primal/dual ratio". As Mercat [Mer08] pointed out, complex primal/dual ratios will arise when the primal and dual edges are not orthogonal. We believe more insight into complex barycentric coordinates can be gained by studying more these concepts.

Discrete conformal maps have been used for many years for 3D mesh parameterization, where the domain is a discrete set of triangles. In fact, many of the most common recipes for real barycentric coordinates were motivated by this problem. The Least Squares Conformal Mapping (LSCM) method [LPRM02] for free-boundary mesh parameterization is based on a discretization of the Cauchy-Riemann equations on triangles, achieving an “As-Similar-As-Possible” effect. Adding soft positional constraints to some of the boundary vertices in the linear LSCM system allows extra control so that the deformed mesh is close in some way to some target geometry. The advantage of the Szegö coordinates proposed here is that they achieve a similar effect without any discretization of the plane. It would be interesting to develop the connection between LSCM and Szegö coordinates.

4 Controllable Conformal Maps for Shape Deformation and Interpolation

Figure 4.1: Conformal deformation of a giraffe with sharp bends at neck and legs. Original model (left) and three deformed versions.

Conformal maps are considered very desirable for planar deformation applications, since they allow only local rotations and scale, avoiding shear and other visually disturbing distortions of local detail. Conformal maps are also orientation-preserving C^{∞} diffeomorphisms, meaning they are extremely smooth and prevent unacceptable "foldovers" in the plane. Unfortunately, these maps are also notoriously difficult to control, so working with them in an interactive animation scenario to achieve specific effects is a significant challenge, sometimes even impossible.

We describe a novel 2D shape deformation system which generates conformal maps, yet provides the user a large degree of control over the result. For example, it allows discontinuities at user-specified boundary points, so true "bends" can be introduced into the deformation. It also allows the prescription of angular constraints at corners of the target image. Combining these provides for a very effective user experience. At the heart of our method is a very natural differential shape representation for conformal maps, using so-called "conformal factors" and "angular factors", which allow more intuitive control compared to representation in the usual spatial domain. Beyond deforming a given shape into a new one at each key frame, our method also provides the ability to interpolate between shapes in a very natural way, such that also the intermediate deformations are conformal.

Our method is extremely efficient: it requires only the solution of a small dense linear system at preprocess time and a matrix-vector multiplication during runtime (which can be implemented on a modern GPU), thus the deformations, even on extremely large images, may be performed in real-time.

4.1 Previous Work

In the 2D domain, the deformation problem boils down to bounding some region of the plane (usually a portion of an image) by a polygonal *source* "cage" P, and, using a set of controls, deform this to another *target* planar polygon Q. In the simplest scenario, we would like to generate some well-behaved mapping between the interior of P and the interior of Q. The easiest way to achieve this is through barycentric coordinates, which express the coordinates of any x in the interior of P as a linear combination of the coordinates of the vertices of P. The image of x is then defined to be the same linear combination of the vertices of Q. Over the years, many recipes for barycentric coordinates have been proposed, and we mention here just the most well-known: the three-point family [FHK06] (which includes the cotangent, mean-value and Wachspress coordinates), the harmonic coordinates [JMD*07] and the maximum entropy coordinates [HS08]. Joshi *et al.* [JMD*07] define harmonic coordinates using a discrete triangulation of the interior of P, and generate a discrete harmonic map between the interior of P and Q. All these mappings distort P as much as is needed in order to fit Q exactly, sometimes yielding quite visually-unsatisfying results. If the exact boundary mapping between P and Q is relaxed, the mapping being controlled only by a small set of "positional constraints", it is possible to optimize the mapping to contain less shear effects and as much local similarities or rigid transformations (which are considered less distorting) as possible. In the discrete world, these are sometimes called the "As-Similar-As-Possible" (ASAP) [LPRM02; SMW06; IMH05; KFG09] or "As-Rigid-As-Possible" (ARAP) [KFG09] mappings.

Realizing that conformal mappings are very desirable in the deformation context, more recent work has restricted the mappings generated to be continuous holomorphic mappings. Thus P will typically not be mapped exactly to Q, rather close to it. These mappings may also be generated using barycentric coordinate functions, such as the Green coordinates [LLCO08], and the equivalent Cauchy coordinates presented in the previous chapter, which may be

controlled directly through the target polygon Q, or through a more intuitive "point-to-point" user interface. Unfortunately, while better than the more distorted traditional deformations, they are still quite difficult to control, and foldovers are not eliminated.

4.2 Contributions

Our main contribution is a novel 2D shape deformation system which generates controllable conformal maps. We introduce the notion of singular points to 2D shape deformation which leads to the ability to generate more realistic deformations compared to existing methods. To the best of our knowledge, this is also the first deformation method that *guarantees* that the Jacobian of the map will not vanish, thus preventing foldovers completely. This approach also leads to a simple way to interpolate shapes in a natural way such that the intermediate deformations are also conformal.

From a technical point of view, we derive a new type of complex barycentric coordinates which are a generalization of the previously defined, and show an interesting connection between Cauchy coordinates and harmonic coordinates which, in turn, leads to a an elegant computational method for harmonic coordinates. We describe the Hilbert barycentric coordinates which leads to an efficient computation of the so-called Hilbert transform. This, in turn, allows us to construct conformal maps via their derivatives. The efficiency of our deformation method is comparable to methods based on other barycentric coordinates. This permits a very fast deformation algorithm which can be easily implemented on a GPU.

4.3 Conformal Shape Deformation

Real-valued barycentric coordinates are affine-invariant. Until recently this ability was considered an advantage. This was challenged by Lipman *et al.* [LLCO08] who demonstrated that when barycentric coordinates are used for deformation, affine transformations introduce shear which in turn destroys the fine details of the shape. To avoid this, Lipman *et al.* [LLCO08] suggested blending the cage normals using another set of barycentric coordinates such that the linear combination of vertices augmented with linear combination of normals is no longer affine-invariant. More recently, we extended this idea by allowing the barycentric coordinates to be complex-valued functions. Since holomorphic functions form a linear subspace, complex holomorphic barycentric coordinates leads to holomorphic deformations.

However, a holomorphic deformation does not always mean the map is conformal since the derivative of the function may vanish inside the domain. This leads to local foldovers which may even cause the winding number of the boundary curve to change. Such visual artifacts (see Figure 4.2) are undesirable and in this chapter we provide a concise way to prevent them.

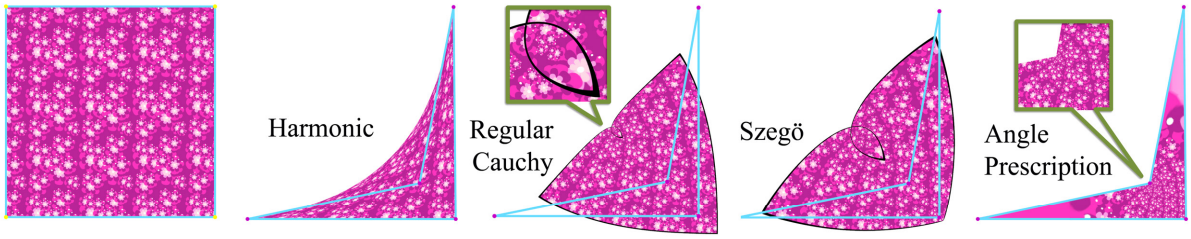


Figure 4.2: Deforming a square. (left to right) Original image enclosed by a cage of 4 vertices and resulting deformations controlled by identical target cages, generated by a number of methods: harmonic coordinates (note how the image "spills" outside the target cage), regular Cauchy coordinates (note how the winding number of the boundary curve has changed from 2π to 4π and the derivative vanishes at a point inside the loop), Szegö coordinates (which tries to map closer to the cage), our deformation using exact angle prescription at the corners results in a bijective conformal map without any foldovers.

Complex barycentric coordinates (when they are holomorphic) have the advantage of being detail preserving, however, this comes with a price. It is impossible to achieve the Lagrange property i.e., to interpolate the target cage, since it is not possible, in general, to prescribe the boundary mapping of a conformal map. This does not contradict Riemann's conformal mapping theorem since that theorem only guarantees that the source domain will be mapped onto the target domain, with no possibility to influence the mapping between the two boundaries. At first glance, the fact that interpolation cannot be achieved using complex barycentric coordinates is disappointing. However, careful examination shows that actually *none* of the existing real-valued barycentric coordinates has the "containment" property as really expected by a user of an interaction deformation tool, namely that $f(\Omega)$ does not "spill out" of $f(\partial\Omega)$.When the target cage is convex (even if the source cage is not), the Rado-Kneser-Choquet theorem [Dur04, Ch. 3] states that the corresponding harmonic map will be a bijection. However, when the target cage is not convex, even harmonic maps cannot guarantee that the Jacobian of the mapping will have positive determinant. This means that even though the target shape interpolates the target cage, some portions of the source region

may “spill” outside the target region. Thus, the outer boundary of the target region does not interpolate the target cage (see Figure 4.2). In fact, we do not know of barycentric coordinates (real or complex) that *guarantee* a bijective mapping (not necessarily harmonic or conformal) between arbitrary source and target cages. Whether it is possible to create such coordinates is an open question.

In light of this unfortunate situation, we now show how to relax the boundary interpolation requirement, in order to be able to construct conformal deformations which are more appealing than deformations that can be achieved using current complex barycentric coordinates. Instead of prescribing the exact boundary mapping at each point along the edges of the cage we will prescribe only the angular change at each point along the boundary. As we will see next, this can be achieved exactly.

The Jacobian of deformations achieved by real-valued barycentric coordinates can easily vanish or even be negative, resulting in local foldovers. Using complex holomorphic barycentric coordinates guarantees that the Jacobian of the mapping (the absolute value of the first complex derivative) is non-negative (since it is a similarity transformation). Our method is superior to that in the sense that it guarantees that the Jacobian is strictly *positive* inside the domain.

The key to success is the use of a representation of conformal maps that always exists and is unique. The representation should be minimal in the sense that it should not contain any redundant information yet still contain all the information required to uniquely reconstruct the deformed shape. The representation we use forms a linear subspace, namely that a linear combination of two valid shape representations is also a valid shape representation. By doing so, we can reduce the optimization problem (of finding a particular conformal map) to finding a particular shape representation within the linear subspace. This leads to an efficient and simple deformation algorithm which has some provable shape- preserving properties.

4.3.1 Shape Representation

Given a complex mapping f, consider the following function:

$$\mathrm{Log}(f') = \mathrm{Log}(|f'|) + i\,\mathrm{Arg}(f')$$

This log derivative of f encodes two local geometric properties of f: 1) The *conformal factor*, $\mathrm{Log}(|f'(z)|)$, describes the local scale change induced by f, and 2) the *angular factor*, $\mathrm{Arg}(f'(z))$, describes the local orientation change induced by f at z. As we will soon see, given one of these two functions (conformal factor or angular factor), satisfying some mild conditions on the boundary of a domain, it is possible to uniquely recover the conformal map having that boundary behavior up to some global transformation. It is not possible to prescribe both functions.

Thus our representation of choice for a conformal map f of a given simply connected 2D domain Ω will be its *angular factor* on $\partial\Omega$ - a piecewise-smooth real-valued scalar function defined on $\partial\Omega$, which we denote by $\theta(t)$ when parameterized by arc length. More specifically, given a conformal map $f(z)$ of a source region Ω, define:

$$\theta(z) = \mathrm{Im}\left(\mathrm{Log}\left(f'\left(z\right)\right)\right) = \mathrm{Arg}(f'(z))$$

With slight abuse of notation, the conformal *shape representation* of f is now defined as the parameterized version of θ on $\partial\Omega$:

$$\theta\left(t\right) = \lim_{z \to w(t)} \theta\left(z\right), \quad z \in \Omega,\ w(t) \in \partial\Omega \tag{4.1}$$

where $w(t)$ is the point on $\partial\Omega$ corresponding to parameter t. $\theta(t)$ has the geometric interpretation as the local change in tangent direction of f on $\partial\Omega$.

The derivative of a holomorphic function f is also holomorphic. Since f is conformal its derivative does not vanish, hence the multivalued $\mathrm{Log}(f'(z))$ function is also holomorphic. Furthermore, since the real and imaginary parts of any holomorphic functions are harmonic functions, it follows that $\theta(z)$ is harmonic on Ω. Hence, the limit (4.1) always exists except at some finite number of points.

4.3.2 Shape Reconstruction

A harmonic function $v(x,y)$ is said to be the (harmonic) conjugate of the harmonic function $u(x,y)$ defined on some open domain $\Omega \subset \mathbb{R}^2$ if and only if u and v satisfy the Cauchy-Riemann equations in Ω. Alternatively, v is the harmonic conjugate of u if and only if

$f(z)=f(x+iy)=u(x,y)+iv(x,y)$ is holomorphic on Ω. Any harmonic function admits a harmonic conjugate whenever its domain is simply connected, and it is unique up to an additive constant.

Given the angular factor $\theta(t)$ defined on the boundary of a simply connected domain Ω, it is always possible to reconstruct a conformal map f which has angular factor θ on $\partial\Omega$, and is unique up to translation and scale, using a two step process. In the first step we construct the derivative f'. In the second step we seek a function f having that derivative.

Existence: Given the function $\theta(t)$ on $\partial\Omega$, it is always possible to solve the Dirichlet problem, extending $\theta(t)$ harmonically into the interior of Ω, thus obtaining $\theta(z)$. Since Ω is simply connected, $\theta(z)$ always admits a harmonic conjugate function, denoted by $\phi(z)$. It is then possible to construct the holomorphic function $g(z)=\phi(z)+i\theta(z)$. Since the complex exponential function is holomorphic, so is $\exp(g(z))$. A consequence of Cauchy's integral theorem is that if a function is holomorphic in a simply connected domain it always has an antiderivative, which means that $\exp(g(z))$ is the derivative of some holomorphic function f. In other words $f'=\exp(g(z))$. Since f' cannot vanish, f is conformal.

Uniqueness: Assume that two conformal maps f_1 and f_2 have the same angular factor $\theta(t)$ on the boundary. Since the derivatives f_1' and f_2' cannot vanish, $\mathrm{Log}(f_1')=\phi_1+i\theta_1$ and $\mathrm{Log}(f_2')=\phi_2+i\theta_2$ are holomorphic. It follows that θ_1 and θ_2 are harmonic functions. The uniqueness of the solution to the Dirichlet problem and the fact that θ_1 and θ_2 coincide on the boundary implies that $\theta_1=\theta_2$. Since both ϕ_1 and ϕ_2 are harmonic conjugate to $\theta_1=\theta_2$, this means that $\phi_1=\phi_2+k$ where k is a real scalar. It follows that $f_1'=\exp(\phi_1+i\theta_1)=\exp(\phi_2+k+i\theta_1)=\exp(k)\exp(\phi_2+i\theta_2)=\exp(k)f_2'$. We conclude that f_1' and f_2' differ only by scale, thus f_1 and f_2 differ only by scale and translation.

This shows that there is a one-to-one correspondence between conformal maps of a given simply connected domain Ω and real-valued functions defined on $\partial\Omega$. This greatly simplifies the task of searching for a particular conformal map since we may restrict our search to the low-dimensional linear subspace of real functions defined on the boundary of the domain.

In order to reconstruct a conformal map from its representation $\theta(t)$ in practice, we will need efficient techniques for finding a harmonic conjugate function and reconstructing a holomorphic function from its derivative. These will be described in the following sections.

4.3.3 Shape Interpolation

The process of animation creation consists of several steps. The animator selects a particular point in time, poses and deforms the objects in the scene, and sets a keyframe. The animator then "plays" the animation, which in turn deforms the objects in the scene by interpolating the shapes between each two consecutive keyframes. The animator then repeats this process in order to correct and refine the animation. This is a tedious process and a good animation tool strives to minimize the manual effort needed.

Our conformal shape deformation framework has an inherent ability to interpolate shapes in a shape-preserving way. Achieving that using our shape representation is very simple. Given two (or more) deformed shapes A and B, we simply blend the shape representations of A and B in a linear manner to obtain a new shape representation:

$$\theta^C(t) = (1-\mu)\theta^A(t) + \mu\theta^B(t)$$

such that $\mu \in [0,1]$ is a parameter that controls the interpolation. The same arguments that allowed the successful reconstruction of a conformal map from $\theta^A(t)$ and $\theta^B(t)$ can be applied to $\theta^C(t)$. This leads to a very simple and efficient algorithm for shape interpolation with some guaranteed properties. The deformations obtained using our method are very natural, avoid local intersections and are conformal everywhere. This is demonstrated in Figure 4.3 but even more so in the video that accompanies the work of [WG10]. In contrast to naïve linear morphing methods, which operate directly in the spatial domain, no shrinkage effects occur and the overall boundary length and shape area is nicely preserved (when it is equal in A and B). Also, as opposed to other interpolation methods which are limited to rotations of up to 180 degrees, our method is capable of interpolating shapes which undergo arbitrarily large rotations (even much larger than 2π) between shapes.

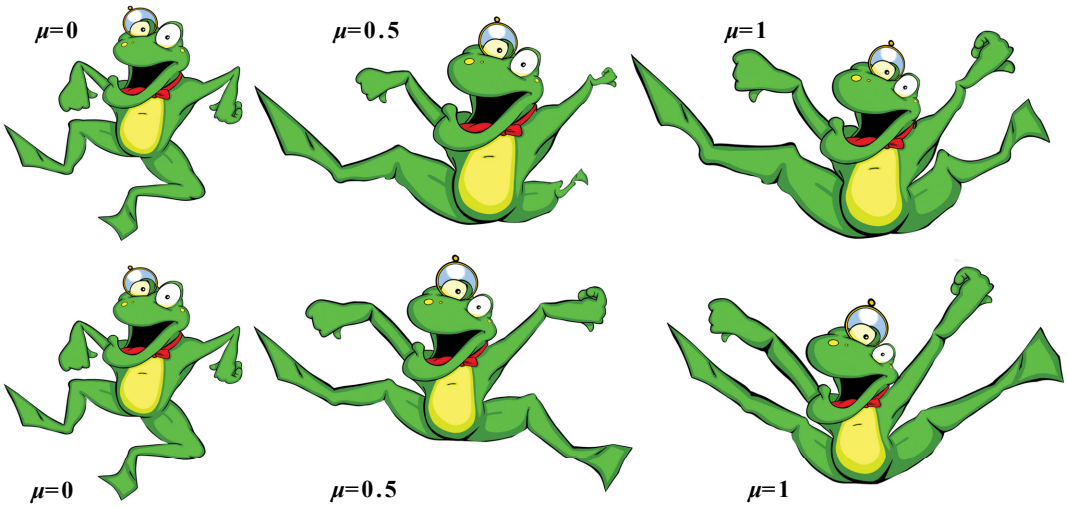


Figure 4.3: Deformation and interpolation: (top row) Regular Cauchy coordinates. Note the shrinkage and rounding of the legs. (bottom row) Our method. Note the sharp bend in elbow and knee.

4.4 Cauchy Coordinates

Since planar deformations may be viewed as a function from the complex plane to itself, it is advantageous to develop the necessary theory in the complex plane. We start by reviewing some basic facts from complex function theory. Consider the following complex function of two variables:

$$C(w,z) = \frac{1}{2\pi i}\frac{1}{w-z}$$

called the *Cauchy kernel*. The *Cauchy transform* of a function f - a piecewise smooth function defined on the boundary of Ω - a region of the complex plane - is defined as the following complex boundary integral [Bel92], producing a new function g on Ω, as illustrated in Figure 4.4:

$$g(z) = \oint_{\partial\Omega} C(w,z) f(w) dw = \frac{1}{2\pi i} \oint_{\partial\Omega} \frac{f(w)}{w-z} dw \qquad 4.2$$

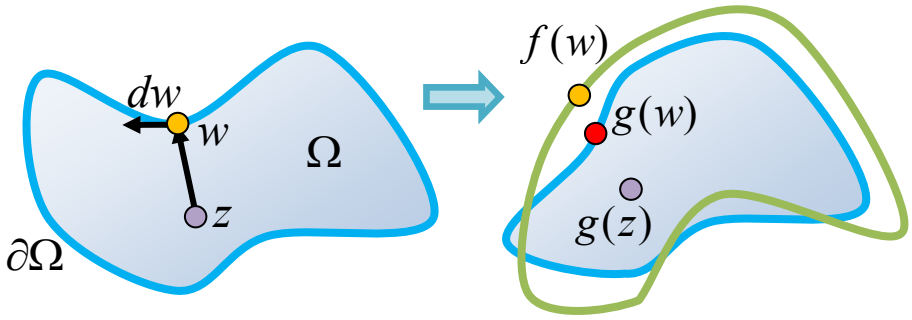


Figure 4.4: Continuous planar mapping from Ω *to* $g(\Omega)$ *generated by the Cauchy transform of* $f(\partial\Omega)$. *If* f *is not holomorphic then* $g(\partial\Omega) \neq f(\partial\Omega)$.

The Cauchy transform has various desirable properties. One is that, under mild assumptions on f, g is always holomorphic on Ω [Bel92, Theorem 3.1]. Hence, when using this transform in the context of planar shape deformation, if the derivative of f does not vanish, the mapping will be conformal. Specifically, the Cauchy transform reproduces any holomorphic function from its boundary values. Unfortunately, as Figure 4.4 illustrates, the Cauchy transform does not possess the interpolation property, namely in general $f(w) \neq g(w)$ on $\partial\Omega$ if f is not holomorphic. This is not surprising since it is well known that (in contrast to harmonic maps), in general, it is impossible to generate a conformal map while prescribing its exact behavior on the boundary.

4.4.1 Regular Cauchy Coordinates

Complex barycentric coordinates were introduced in chapter 3. We allowed barycentric coordinates, which are traditionally real-valued basis functions, to assume complex values. When the basis functions are holomorphic, their linear combinations, using complex coefficients, may be used to generate detail preserving deformations of a 2D region. Chapter 3 describes several recipes for such complex barycentric coordinates, the most fundamental being the Cauchy coordinates, based on the Cauchy transform (4.2) described above, which were proved to be equivalent to the 2D version of the Green coordinates [LLCO08]. The Cauchy coordinates were derived by discretizing $\partial\Omega$ into a set of straight lines, namely a polygon (the so-called *cage*), reducing the integral (4.2) to:

$$g(z) = \frac{1}{2\pi i} \sum_{j=1}^{n} \oint_{e_j} \frac{f(w)}{w - z} dw \qquad 4.3$$

Under the assumption that f is linear on each edge and continuous at the cage vertices, the integral (4.3) has a closed-form expression leading to an elegant formula for the (regular) Cauchy coordinate function $C_j(z)$ at vertex j (see Figure 4.5, left, for the exact notations):

$$g(z) = \sum_{j=1}^{n} C_j(z) f_j \quad C_j(z) = \frac{1}{2\pi i} \left(\frac{B_{j+1}}{A_{j+1}} \log\left(\frac{B_{j+1}}{B_j} \right) - \frac{B_{j-1}}{A_j} \log\left(\frac{B_j}{B_{j-1}} \right) \right)$$

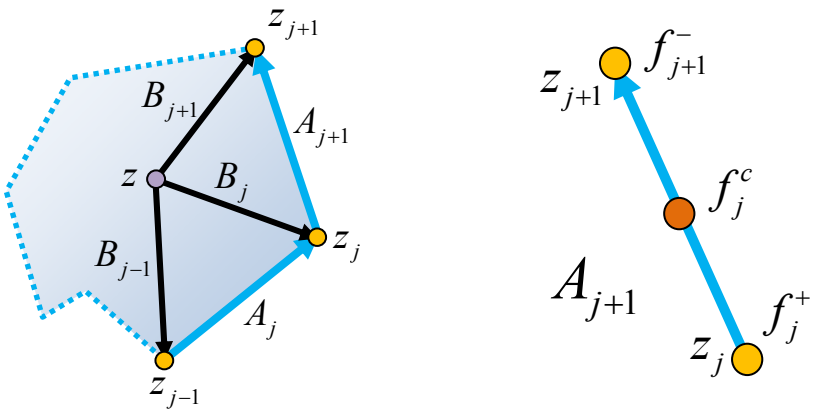


Figure 4.5: Cauchy coordinate notations.

where n is the number of vertices of the cage and f_j is the value of f at the j^{th} vertex. Note that the Cauchy coordinates $C_j(z)$ are not defined on $\partial\Omega$, however the limit, as z approaches $\partial\Omega$, exists and the function is well behaved.

Conformal maps are angle preserving, however, the boundary curve is not considered to be part of the domain. For example, consider a conformal map f of a square to a disk, where it is clear that the boundary angles are changed. Examining $\theta(z)=\text{Arg}(f'(z))$ on the boundary will reveal that the function is not continuous at the corners of the square.

The Cauchy transform spans the entire linear space of holomorphic functions. However, its discretized version, defined in (4.3), which we use in order to approximate the function $\text{Log}(f')$, spans only an n dimensional subspace of the holomorphic functions. It turns out that this subspace is relatively limited, for two main reasons. First, the Cauchy basis functions do

not contain any singularity. They vary smoothly everywhere, even in the vicinity of the cage vertices. Since any linear combination will inherit the properties of the basis functions, it is not possible to properly express near-singular holomorphic functions. In particular, it cannot support changes in angle at the cage vertices. The second reason is due to the implicit assumption that f behaves linearly on the polygonal edges. Most holomorphic functions behave quite differently and a linear approximation is just not good enough.

4.4.2 Generalized Cauchy Coordinates

We now show how to extend the regular Cauchy coordinates in two ways. First, we allow f to have quadratic rather than linear behavior on each edge. Second, and most important, we allow f to have two distinct values at each singular vertex (a subset of the cage vertices), introducing a discontinuity for f at the vertex if the values are different. These two properties are achieved by prescribing three values for f on the edge A_{j+1}: f_j^+, f_j^c and f_{j+1}^- (see Figure 4.5, right) such that:

$$f_j^+ = f(z_j^+), \qquad f_j^c = f\left(\frac{z_j + z_{j+1}}{2}\right), \qquad f_{j+1}^- = f(z_{j+1}^-)$$

Given these, f can be expressed on the edge A_{j+1} as follows:

$$\begin{aligned} f(w) = {} & \frac{f_j^+}{A_{j+1}^2}\left(2w^2 - wz_j - 3wz_{j+1} + z_j z_{j+1} + z_{j+1}^2\right) \\ & + \frac{f_j^c}{A_{j+1}^2}\left(-4w^2 + 4wz_j + 4wz_{j+1} - 4z_j z_{j+1}\right) \\ & + \frac{f_{j+1}^-}{A_{j+1}^2}\left(2w^2 - 3wz_j - wz_{j+1} + z_j^2 + z_j z_{j+1}\right) \end{aligned}$$

The next step is to plug f into (4.3) and evaluate the integral. The derivation is long and technical and is omitted here for brevity. Fortunately, as with the regular Cauchy coordinates, the integral has a closed-form solution which gives rise to the *generalized Cauchy coordinates*:

$$CE_j(z) = \frac{1}{\pi i A_{j+1}^2}\left(A_{j+1}\left(B_{j+1}+B_j\right) - 2B_{j+1}B_j \log\left(\frac{B_{j+1}}{B_j}\right)\right)$$

$$CV_j^-(z) = \frac{B_{j-1}}{2\pi i A_j^2}\left(\left(B_j+B_{j-1}\right)\log\left(\frac{B_j}{B_{j-1}}\right) - 2A_j\right)$$

$$CV_j^+(z) = \frac{B_{j+1}}{2\pi i A_{j+1}^2}\left(\left(B_{j+1}+B_j\right)\log\left(\frac{B_{j+1}}{B_j}\right) - 2A_{j+1}\right)$$

While the number of regular Cauchy coordinate functions is n (one coordinate for each vertex), here we have one coordinate function for each edge (CE) and an additional two coordinate functions (CV^- and CV^+) for each vertex, resulting in $3n$ coordinate functions. Finally, the generalized discrete Cauchy transform is:

$$g(z) = \sum_{j=1}^{n}\left(CE_j(z)f_j^c + CV_j^-(z)f_j^- + CV_j^+(z)f_j^+\right)$$

It is important to note that even though the basis functions are bounded holomorphic functions at any point inside Ω, the limit of $CV_j^-(z)$ and $CV_j^+(z)$ at the vertex z_j does not exist, rather has a logarithmic singularity there. The singularity drastically improves the expressive power of the discrete Cauchy transform, and we refer to this as the *discontinuous case*. If discontinuity is not needed (i.e., we assume that $f_j^+ = f_j^-$), it is possible to reduce the number of basis functions by combining the two coordinates at each vertex:

$$CV_j(z) = CV_j^-(z) + CV_j^+(z)$$

In this case the total number of basis functions is reduced to $2n$ and the transform has the following form, which we refer to as the *continuous case*:

$$g(z) = \sum_{j=1}^{n}\left(CV_j(z)f_j + CE_j(z)f_j^c\right)$$

Note that in this case the limit of $CV_j(z)$ at the vertices exists as in the regular Cauchy coordinates. Furthermore, if the values at the mid-edges equal the average of the values at the edge endpoints, i.e. $f_j^c = (f_{j+1}^- + f_j^+)/2$, the generalized Cauchy coordinates will reduce to regular

Cauchy coordinates. Note also that generalized Cauchy coordinates reproduce both linear and quadratic functions.

Figure 4.6 shows a visualization of the real and imaginary parts of the generalized Cauchy coordinates for a non-convex polygon.

4.5 Dirichlet Problems and Hilbert Transforms

In Section 4.3.2 we explained how to reconstruct a conformal map from its shape representation (prescribed angular factor on the boundary). This required the solution of a Dirichlet problem and the computation of a harmonic conjugate function. In this section, we describe an efficient, elegant and accurate numerical method for such computations.

The operator that takes a function u to its harmonic conjugate function v is the *Hilbert transform*. The classic definition of the Hilbert transform operates on real-valued functions of a single real variable, however, for our applications we prefer to use an alternative definition where u and v are real-valued functions of a single *complex* variable defined on the upper part of the complex plane such that the transform is applied to their restriction to the real axis. This is equivalent since it is always possible to uniquely extend a real-valued function defined on the real axis to be harmonic on the upper complex plane. Once the extended harmonic function is found, it is possible to find its harmonic conjugate and again, its restriction to the real axis is considered as the transformed real-valued function.

The same logic can be used to define the Hilbert transform on more general domains. For a simply connected domain Ω, consider the Hilbert transform [Bel92] as an operator that takes a real-valued function $u(s)$ on $\partial\Omega$ to the holomorphic function $\mathcal{H}(z)=u+\mathrm{i}v$ on Ω. The transform always exists and is unique up to additive imaginary constant. However, in general, it does not possess a closed-form expression and computing it requires some effort.

The Boundary Element Method (BEM) [Kyt95] is a numerical computational method of solving linear partial differential equations such as Laplace's equation. The main idea is to formulate the problem as a boundary integral equation and then discretize the *boundary* into finite elements and approximate the equation as a linear combination of a set of basis functions with unknown coefficients. The coefficients are obtained as a solution to a dense linear system. Once the coefficients are found, the solution can be evaluated at any arbitrary

point inside the domain. In contrast to the Finite Element Methods (FEM), BEM is based on discretization of the boundary alone and avoids discretization of the entire domain. The advantage is that the complexity of the boundary, which dictates the complexity of the solution, is decoupled from the complexity of the domain. The Complex Variable Boundary Element Method (CVBEM) [HL87] is a variant of BEM for solving two dimensional problems based on complex numbers. An advantage of CVBEM over BEM is that the derivation of the boundary integrals is done using complex analysis which greatly simplifies the derivation and leads to closed-form expressions. But more importantly, as we will elaborate on later, this technique is more natural if our ultimate goal is to construct *dual pairs* of harmonic functions (which together form a holomorphic complex function). We now show how CVBEM combined with our generalized Cauchy coordinates leads to an efficient and elegant method for the computation of the Hilbert transform.

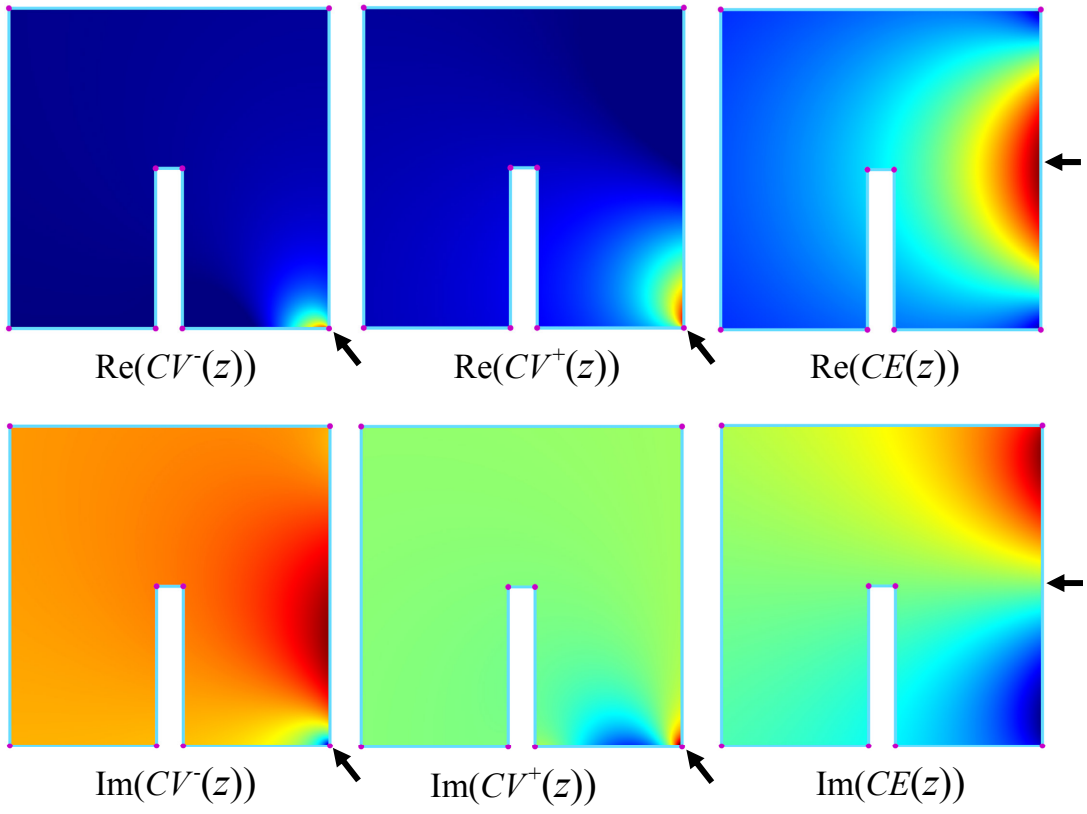


Figure 4.6: Visualization of the generalized Cauchy coordinates for the marked vertex and edge.

4.5.1 The Dirichlet Problem

Given the values $u(s)$ of a function on $\partial\Omega$, we want to solve the Dirichlet problem - find a harmonic extension $u(z)$ to Ω preserving the given boundary values. Once $u(z)$ is found we want to find its harmonic conjugate function $v(z)$. As we shall see next, the two problems can be solved simultaneously.

Since the real and imaginary parts of any holomorphic function $g(z)$ are harmonic and since the Cauchy transform spans the entire space of holomorphic functions, it follows that *any* harmonic function h may be expressed as:

$$h(z) = \mathrm{Re}\big(g(z)\big) = \mathrm{Re}\left(\frac{1}{2\pi i}\oint_{\partial\Omega}\frac{f(w)}{w-z}dw\right) \qquad 4.4$$

for some complex f. The following error functional measures the difference on $\partial\Omega$ between some arbitrary harmonic function $h(z)$ and the prescribed boundary conditions $u(s)$:

$$Err\big(h(z)\big) = \oint_{\partial\Omega}\big(u(s)-h(s)\big)^2\,ds \qquad 4.5$$

We would like to find the complex function f defined on $\partial\Omega$ that minimizes this error. Note that there always exists a function f which produces zero error since the Dirichlet problem always has a solution. To solve the optimization problem in practice we substitute the continuous Cauchy transform with the finite-dimensional discrete transform $g(z)=\sum_{j=1}^{s}C_j(z)f_j$ and approximate the integral in (4.5) as a sum over a k-sampling of $\partial\Omega$. Denoting the generalized Cauchy coordinates by $C_j(z)=\varphi_j(z)+i\psi_j(z)$ and the complex coefficients by $f_j=f_j^x+if_j^y$, the optimization problem becomes:

$$\underset{\{f_j\}_{j=1}^{s}}{\mathrm{argmin}}\sum_{i=1}^{k}\big(u_i-h(w_i)\big)^2 \qquad 4.6$$

where: $$h(w_i)=\mathrm{Re}\big(g(w_i)\big)=\sum_{j=1}^{s}\varphi_j(w_i)f_j^x-\psi_j(w_i)f_j^y$$

w_i are the locations of the k boundary samples and s is the total number of basis functions. Since the error (4.5) is quadratic, the optimization can be solved by solving a rather small dense (overdetermined) linear system of equations with $2s$ real variables. Once the optimal real coefficients f_j^x and f_j^y are found, we can plug them into $\mathrm{Re}\left(\sum_{j=1}^{s} C_j(z) f_j\right)$, obtaining a formula for the desired harmonic function $h(z)$ at any interior point z. It is important to note that the function $h(z)$ only approximates the given boundary conditions. However, we cannot hope for better than this since an analytic expression for the Dirichlet problem does not exist (in general). Moreover, using this construction we can guarantee that $h(z)$ is harmonic in the continuous (as opposed to discrete) sense.

The reader may have noticed that in order to solve (4.6), the generalized Cauchy coordinates need to be evaluated on the boundary of the domain, where they are singular. We define the values of the coordinate functions on $\partial\Omega$ to be their limit (in case it exists) when approaching the boundary from the interior of Ω:

$$C_j(w) = \lim_{z \to w} C_j(z), \quad z \in \Omega,\ w \in \partial\Omega$$

These limits exist everywhere except at the singular vertices. We refer to Appendix F for the appropriate formulae.

4.5.2 The Hilbert Transform

We now turn to finding the harmonic conjugate function. The main reason why we chose to solve the Dirichlet problem using the Cauchy transform can now be revealed. The desired conjugate is simply: $p(z) = \mathrm{Im}\left(\sum_{j=1}^{s} C_j(z) f_j\right)$. Since the discrete Cauchy basis functions are holomorphic, it follows that $p(z)$ is harmonic and that $h(z)$ and $p(z)$ are *exactly* harmonic conjugate functions. We thus conclude that the Dirichlet problem and finding a harmonic conjugate are two equivalent problems since once we solve for the complex coefficients f_j, we obtain a closed-form expression for $h(z)$ as well as for $p(z)$. Finally, the discrete Hilbert transform that we seek is: $\mathcal{H}(z) = \sum_{j=1}^{s} C_j(z) f_j$

The inverse of the Hilbert transform which takes the harmonic function v to the holomorphic function $\mathcal{H}^{-1}(z)=v-iu$ is easily obtained by multiplying $\mathcal{H}(z)$ by the complex number i.

4.6 The Antiderivative

In Section 4.5 we showed how to compute the Hilbert transform which enables us to construct a holomorphic function such that its real (or imaginary) part coincides with a given real-valued function along the boundary. Our ultimate goal is to reconstruct a conformal map from its prescribed angular factor along the boundary $\theta(t)$. Applying the inverse Hilbert transform to $\theta(t)$, results in the holomorphic function $g(z)=\mathrm{Log}(f'(z))$ from which we can evaluate $f'(z)=\exp(g(z))$ anywhere in Ω and in particular on $\partial\Omega$.

The next step is to reconstruct the actual mapping f. Even though computing f' using the discrete Hilbert transform as described in Section 4.5 guarantees that the approximated f' is holomorphic, its antiderivative does not possesses a closed-form expression. However, using the following observation we can avoid numerical "integration" of f'. Since f is holomorphic it can be approximated using the discrete Cauchy transform $f(z)=\sum_{j=1}^{s} C_j(z)d_j$. Differentiating with respect to z results in $f'(z)=\sum_{j=1}^{s} D_j(z)d_j$ where $D_j(z)=d\left(C_j(z)\right)/dz$ is the derivative of the generalized Cauchy basis functions, which have a closed-form expression. Following the same logic of Section 4.5, we employ the following discrete optimization problem:

$$\underset{\{f_j\}_{j=1}^{s}}{\operatorname{argmin}} \sum_{i=1}^{k} \left| f'(w_i) - \sum_{j=1}^{s} D_j(w_i) d_j \right|^2 \qquad 4.7$$

where w_i are the locations of k boundary samples and d_j are the complex coefficients that we solve for. In contrast to Section 4.5, here the linear system of equations is over the field of complex numbers. Once the linear system is solved and the coefficients d_j are found, they can be plugged into the discrete Cauchy transform formula in order to compute $f(z)$ at any arbitrary interior point z.

The expressions for the first derivative of the generalized Cauchy coordinates and their limits when an internal point approaches the boundary are given in Appendix G.

4.7 Deformation by Angle Prescription

Having laid down all the mathematical building blocks for our deformation algorithm, we can describe it from the user point of view. The user first defines a polygon with n vertices to serve as the boundary of the domain and a set of p singular points are selected among the cage vertices. To simplify the user interface, we allow the user to prescribe the angular change $\theta(t)$ only at the singular points rather than on any point on the boundary. Since the singular points are considered to be corner points we actually have two distinct angles for each one of them, resulting in $2p$ degrees of freedom. The values of $\theta(t)$ are then interpolated linearly along the boundary. We give the user the freedom to choose between two possible user interfaces.

4.7.1 Rotational Handles

This is the most direct user interface. A rotational handle manipulator is attached to each singular vertex. In addition, the user is required to place one anchor handle inside the domain in order to fix the global scale and translation degrees of freedom. To control the amount of rotation, the user first selects a particular handle. Rotating the handle adds the same amount of rotation to the tangent vectors just before and just after the selected corner, which results in an overall orientation change at the corner. Dragging the handle rather than rotating it causes the two tangent vectors to rotate in opposite directions, which has the effect of changing the corner angle rather than its orientation. During interaction, the system constantly updates the location of the handles to match the deformed location of the singular corners.

4.7.2 Cage-Based

Alternatively, the user can manipulate the vertices of the polygonal boundary curve (the cage). In this case, all the vertices of the cage are considered to be singular. The corner angles are taken to be equal to the angle of the edges of the target cage, which results in a deformed shape bound by straight lines. Figure 4.9 show an example of mapping a sinusoidal bar to a triangle. Since the edges of the target cage are straight, the image of the bar is a perfect triangle.

In addition, the user has an additional parameter per edge. This can be thought of as a tension parameter. When it has a large value, the angular change $\theta_i^+(t)$ just after the i^{th} singular vertex equals the angular change $\theta_{i+1}^-(t)$ just before the $(i+1)^{th}$ singular vertex. This results in a constant θ along the edge, so the edge is mapped to a straight line in the target image. When tension is low we reduce the angles by an amount proportional to the tension parameter. This has the effect of bending the edge to form a circular arc, creating a more relaxed conformal map with more moderate variations in scale. This is demonstrated in Figure 4.7. The three degrees of freedom can be either prescribed directly at an internal anchor point or can be deduced from the vertices of the target cage by some heuristic such as finding a scale and translation that minimizes the distance between the target cage and the boundary of the target image.

4.8 Implementation Details

During interaction, the user constantly changes the boundary condition $\theta_i \quad i \in [1, 2p]$ at the singular vertices which results in immediate visual feedback in the form of the deformed conformal map. Since this is an interactive application running in real-time, it is advisable to solve the linear systems during runtime as quickly as possible, even at the price of a longer preprocessing time.

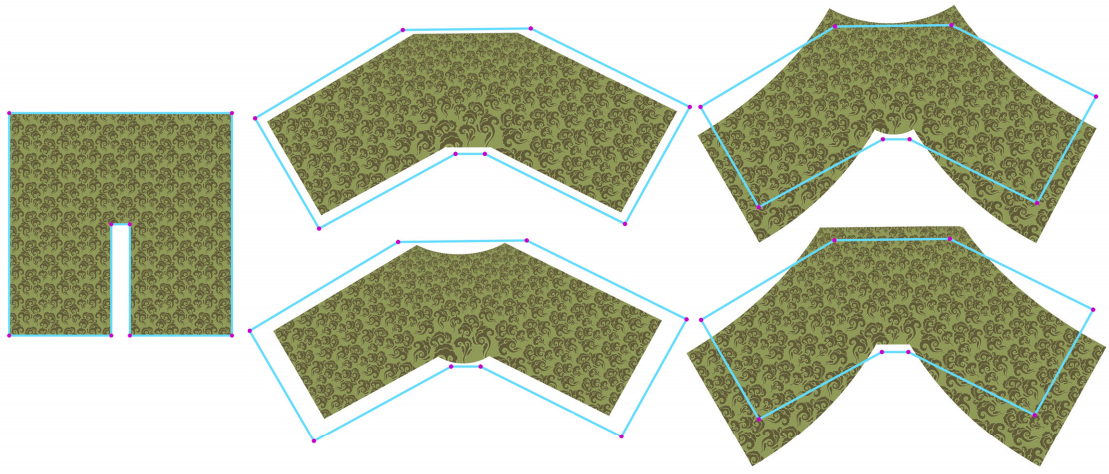


Figure 4.7: Deformation using angle prescription: (Left) original shape enclosed by a cage. (Top left) all target edges are straight. (Top right) Tension is relaxed at all edges. (Bottom left and right) Tension is relaxed at only some of the edges.

The number of basis functions needed in order to achieve good approximations depends on the complexity of the domain. Convex regions usually require less basis functions while highly concave and twisted regions require more. Hence, in practice we usually increase the number of basis functions by sampling the original edges of the cage, adding some *virtual* vertices. However, away from the singular vertices of the cage we expect that the mapping will behave well and will not contain radical changes, so at virtual vertices we can make do with the continuous case of generalized Cauchy coordinates, keeping the number of basis functions manageable. In contrast, at the singular vertices of the cage we always use the discontinuous case of generalized Cauchy coordinates in order to allow maximum flexibility. Denote by r the total numbers of vertices in the super sampled cage. We have one CE_j basis function per edge, one CV_j basis function per "regular" vertex and two basis functions CV_j^- and CV_j^+ per singular vertex which results in totally $s=2r+p$ basis functions. A typical value for p is 4-20 and a typical value for s is 200.

Once the user defines a cage with n vertices and a set of p singular vertices, the polygon is sampled and refined to have r vertices. We then uniformly sample the polygon with k samples, avoiding sampling the virtual vertices, and evaluate the s generalized Cauchy coordinates on each sample. Choosing k to be an order of magnitude larger than r led to good results in our experiments.

Next, we construct the real matrix $C_{k\times 2s}$ containing the real and imaginary parts of the coordinates, φ and $-\psi$, at each sample in interleaved columns. Computing its pseudo-inverse and combing every two consecutive rows into a single row of complex numbers results in the matrix $C^+_{s\times k}$.We can express the solution to the optimization problem (4.6) in matrix form:

$$f_{s\times 1} = C^+_{s\times k} u_{k\times 1}$$

$u_{k\times 1}$ are the given values of u on the k boundary samples. Since we assume that u changes linearly between singular vertices, we can further express the samples $u_{k\times 1}$ as a matrix-vector product $u_{k\times 1}=S_{k\times 2p}\theta_{2p\times 1}$ where $S_{k\times 2p}$ is a "uniform" sampling matrix having exactly two non-zero elements in each row corresponding to two singular vertices. The rate of change is proportional to the arc length. It follows that:

$$f_{s\times 1} = \underbrace{C^{+}_{s\times k} S_{k\times 2p}}_{T_{s\times 2p}} \theta_{2p\times 1} = T_{s\times 2p}\theta_{2p\times 1}$$

Once the matrix T is computed, we obtain a formula for evaluating the discrete Hilbert transform at any point z inside Ω:

$$\mathcal{H}(z) = \sum_{j=1}^{s} C_j(z) \sum_{i=1}^{2p} T_{ji}\theta_i = \sum_{i=1}^{2p} \theta_i \underbrace{\sum_{j=1}^{s} C_j(z) T_{ji}}_{\mathcal{H}_i(z)}$$

We denote by $\mathcal{H}_i(z)$, the i^{th} Hilbert coordinate which has a closed-form expression. Once the $2p$ Hilbert coordinates are computed in the preprocess step, the Hilbert transform can be computed during interaction by a simple matrix-vector multiplication:

$$\mathcal{H}(z) = \sum_{i=1}^{2p} \mathcal{H}_i(z)\theta_i \qquad 4.8$$

Since θ is in fact the imaginary part (rather than the real part) of the holomorphic function $\mathcal{H}(z)$, we need the inverse Hilbert transform. This is obtained simply by multiplying θ by the complex number i before computing (4.8).

The $2p$ Hilbert coordinates are evaluated during the preprocess step at the k boundary samples and are stored in a matrix. During interaction, the user provides a vector of $2p$ angles which is multiplied by the Hilbert coordinates matrix. This results in a new vector of k complex numbers. Taking the exponent of each element in the vector produce a k-vector of boundary derivatives.

Our final task is then to construct a conformal map that has those derivatives along the boundary. The linear system (4.7) is also pre-factored during preprocessing. Finding the optimal complex coefficients is achieved during interaction by a matrix-vector multiplication. Finally, each point $z \in \Omega$ is mapped to its target position using the discrete Cauchy transform. Note that computing the generalized Cauchy coordinates at all the internal points and storing them in a large matrix is also done in preprocessing.

There are three degrees of freedom that must be fixed. One is due to the Hilbert transform which corresponds to a constant addition to ϕ and has the geometric interpretation of global

scale. The other two are due to the fact that the complex matrix containing the coordinates of the derivatives has co-rank one. This can be easily fixed by augmenting the real matrix $C_{k\times 2s}$ and the complex matrix of derivatives with one additional row.

We have implemented our algorithm as a plug-in to Autodesk Maya™. Computation of the pseudo-inverse matrices required in order to solve the linear systems (4.6) and (4.7) is done by SVD on our Intel i7 processor, using the Intel MKL library. This requires up to 15 seconds for our most complex shape. Since our method is based on barycentric coordinates, during interaction, we only need to perform dense matrix vector multiplications (an "embarrassingly parallel" process). This is done on a Nvidia Quadro FX 5800 graphics processor. The deformation is computed interactively even for huge images, giving the user immediate feedback, which is crucial for an effective animation session.

4.9 Computing Harmonic Coordinates

Harmonic coordinates [JMD*07] have many useful properties, making them attractive for many graphics applications such as shape deformation and color and data interpolation. They are smooth, have the Lagrange (interpolation) property, and reproduce constant and linear functions. But their main advantage over the celebrated mean-value coordinates [HF06] is that they are non-negative, even for non-convex boundaries. This property is especially important for shape deformation.

On the downside, harmonic coordinates do not possess a closed-form expression for general boundary shapes. Hence a good numeric approximation is sought. Joshi *et al.* [JMD*07] compute the coordinates by discretizing the entire domain (2D or 3D) into piecewise–linear finite elements and solving a discrete Laplace equation for these elements. Martin *et al.* [MKB*08] applied the Method of Fundamental Solutions [MFS] for computing harmonic coordinates on 3D polyhedral domains. The harmonic function is constructed as a linear combination of radial basis functions (the fundamental solution of the Laplace equation) with a set of real coefficients.

We obtain an alternative derivation for the computation of harmonic coordinates for simply connected 2D domains. The i^{th} harmonic coordinate is nothing but the real part of the Hilbert coordinate $\mathrm{Re}(\mathcal{H}_i(z))$. This derivation of harmonic coordinates possesses the same constant

and linear reproduction properties as the discrete harmonic coordinates with the additional advantage of being *continuously* harmonic over Ω. Figure 4.8 and the accompanying video demonstrates that high quality approximations for harmonic coordinates can be achieved when the generalized Cauchy coordinates are used, even with a very small number of basis functions.

4.10 The Riemann mapping

The celebrated Riemann mapping theorem states that every simply connected region of the complex plane can be mapped conformally with a bijective map to the unit disk. This theorem arises in many different branches of mathematics and has a large range of applications.

While the exact Riemann mapping generally does not possess an explicit form using only elementary functions (not even in the simple case of mapping a square to a disk) many methods for approximating the Riemann map exists. Our goal in this section is to provide a relatively simple algorithm for computing the Riemann map based on Hilbert coordinates. Our method belongs to a class of methods that provide a continuous representation for the mapping. It is easy to implement, requires only the solution of a dense linear system, does not involve any sophisticated non-linear optimization and does not requires any numerical integration.

The algorithm is a direct implementation of the fundamental idea behind Riemann's original proof to the theorem. Given an arbitrary interior point $z_0 \in \Omega$, evaluate the following function $u(w) = -\log|w-z_0|$ on the boundary of Ω and construct an harmonic function $u(z)$ that coincide with the prescribed boundary values. We emphasize that this does not imply that $u(z) = -\log|z-z_0|$ within Ω, as the function $-\log|z-z_0|$ has a pole at z_0 where it is not harmonic. Nonetheless, by the Dirichlet principle, there always exists a unique harmonic function $u(z)$ which coincides with $-\log|w-z_0|$ on $\partial\Omega$. Next, construct $v(z)$, a harmonic conjugate function to $u(z)$ which exists due to the fact that Ω is simply connected. In that case $v(z)$ is only unique up to scalar addition. It follows that the function $g(z) = u(z)+iv(z)$ is holomorphic and that:

$$f(z) = \left(z - z_0\right)\exp\left(u(z) + iv(z)\right) \tag{4.9}$$

is also holomorphic and unique up to a rotation.

It remains to show that $f(z)$ is actually the Riemann map that we look for. Since $f(z)$ is holomorphic and has non-vanish derivative, it is also conformal. A boundary point w is mapped by $f(z)$ to a point on the unit circle. This can be seen from:

$$\left|f(w)\right| = \left|w - z_0\right| \left|\exp\left(u(w) + iv(w)\right)\right| = \left|w - z_0\right| \exp\left(-\log\left|w - z_0\right|\right) = 1$$

Since the Jacobian of the mapping is strictly positive and the boundary Ω is mapped to a convex shape we get that $f(z)$ is bijective which conclude the proof.

We thus conclude that the problem of finding the Riemann map may be reduced to the problem of finding a holomorphic function $g(z)$ given the real part of its boundary values. In that case, we can simply use the machinery of the Hilbert coordinates developed in Section 4.5. Once $g(z)$ is found, it is plugged into (4.9) to obtain $f(z)$. Figure 4.10 demonstrate some of the results obtained using this method.

4.11 Discussion

We have presented a novel method for 2D shape deformation with some *guaranteed* shape-preserving properties. Our method produces pure conformal maps, hence local foldovers are completely eliminated. We allow the user to augment the shape with a small set of singular points along the boundary.

Supported by a richer subspace spanned by a generalization of the complex Cauchy barycentric coordinates, we are able to create realistic deformations that have controllable sharp bends. The user has the ability to precisely prescribe the angular change along the boundary. The main tool is a natural representation for conformal maps. We achieve superb image quality and high performance thanks to efficient computation of the Hilbert transform, which is based on a new set of barycentric coordinates which we called Hilbert coordinates. Our method comes with a built-in ability to correctly interpolate shapes possessing the same shape-preserving properties. This is an important feature missing from all other deformation algorithms based on barycentric coordinates.

The optimization problem that we solve is linear. However, the final map is a non-linear function of the boundary angles. Although positional constraints can be added to (4.7), by doing so, we may lose the shape-preservation property (i.e. the map might not be injective anymore). An interesting direction for future research would be to solve a non-linear optimization problem in order to accommodate positional constraints.

Another direction for future research may be to define energies other than (4.5), optimizing both for orientation and scale constraints. Since satisfying both perfectly is impossible, the requirement can be relaxed to achieve a more balanced mapping with less scale variations and more local control.

We believe that Hilbert coordinates might be useful for other applications as well. For example, computing the Riemann map of a polygon to the unit disk is an important building block in many graphics and geometry applications. Initial results show that this can be done. Finally, we would like to find an analog to our theory in three dimensions, as an immediate extension does not seem to be possible. In fact, conformal maps in 3D rarely exist. Nonetheless, the theory of quasi-conformal maps may shed some light on this subject.

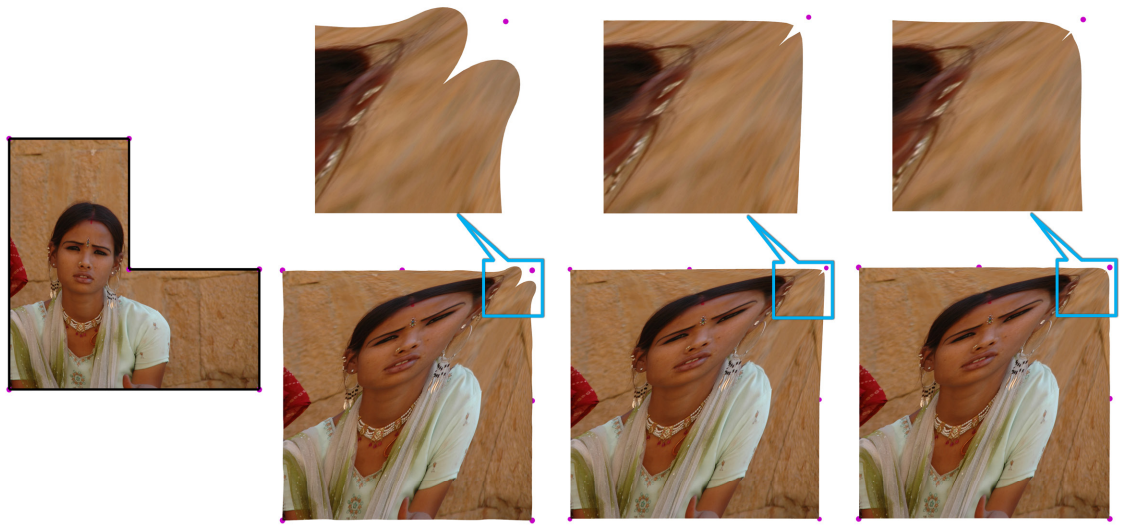


Figure 4.8: Image deformation using our derivation of harmonic coordinates. (extreme left) original L-shaped image enclosed by a cage with 8 vertices, (left to right) deformation to a square using various numbers and types of basis functions.

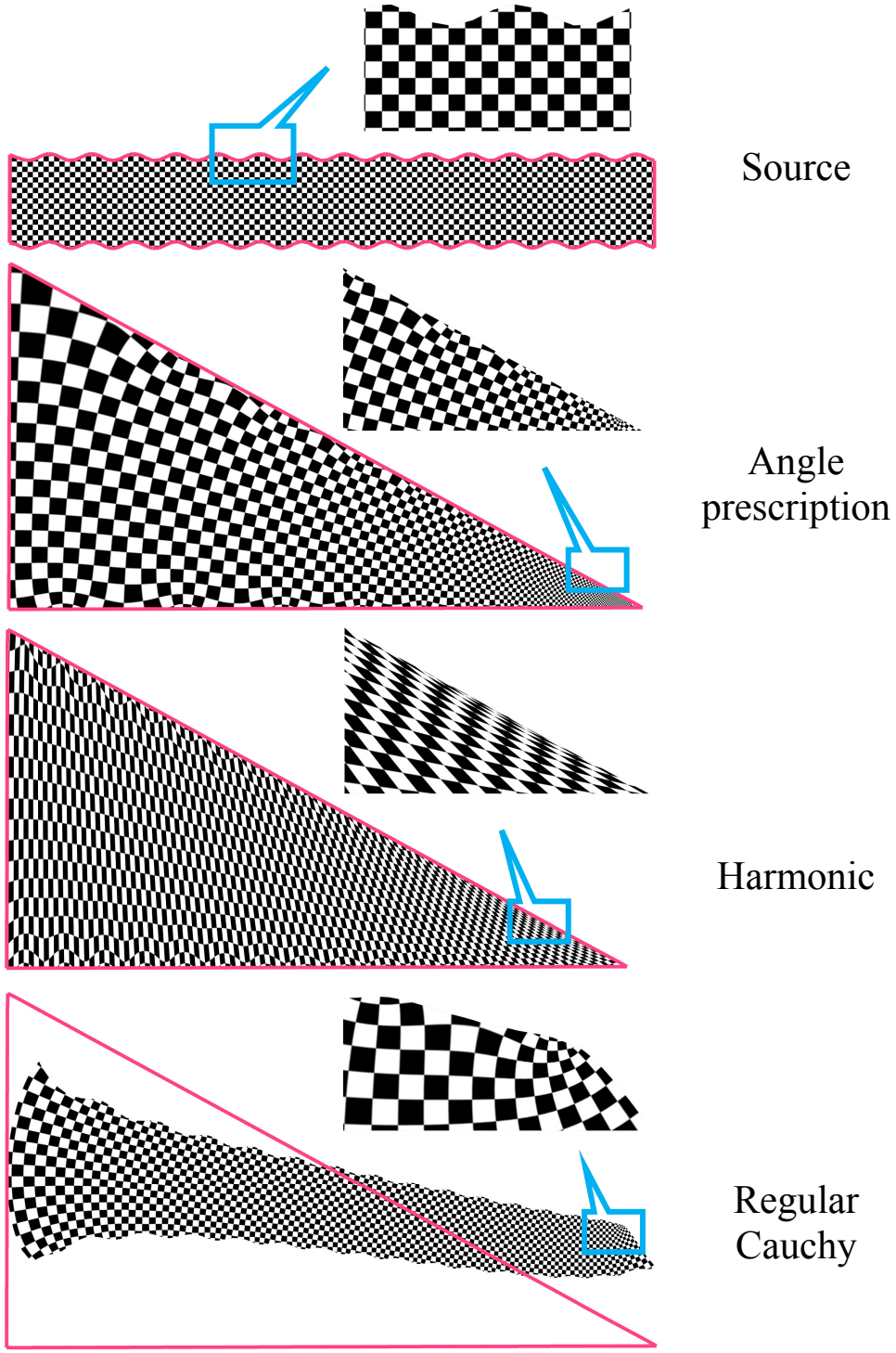


Figure 4.9: Deformation of a bar with sinusoidal boundary into a triangle. (top to bottom) Source with a cage having 200 vertices, Conformal deformation using exact angle prescription, Result using harmonic coordinates (note the shear), Result using regular Cauchy coordinates (the shape does not follow the cage edges position nor orientation).

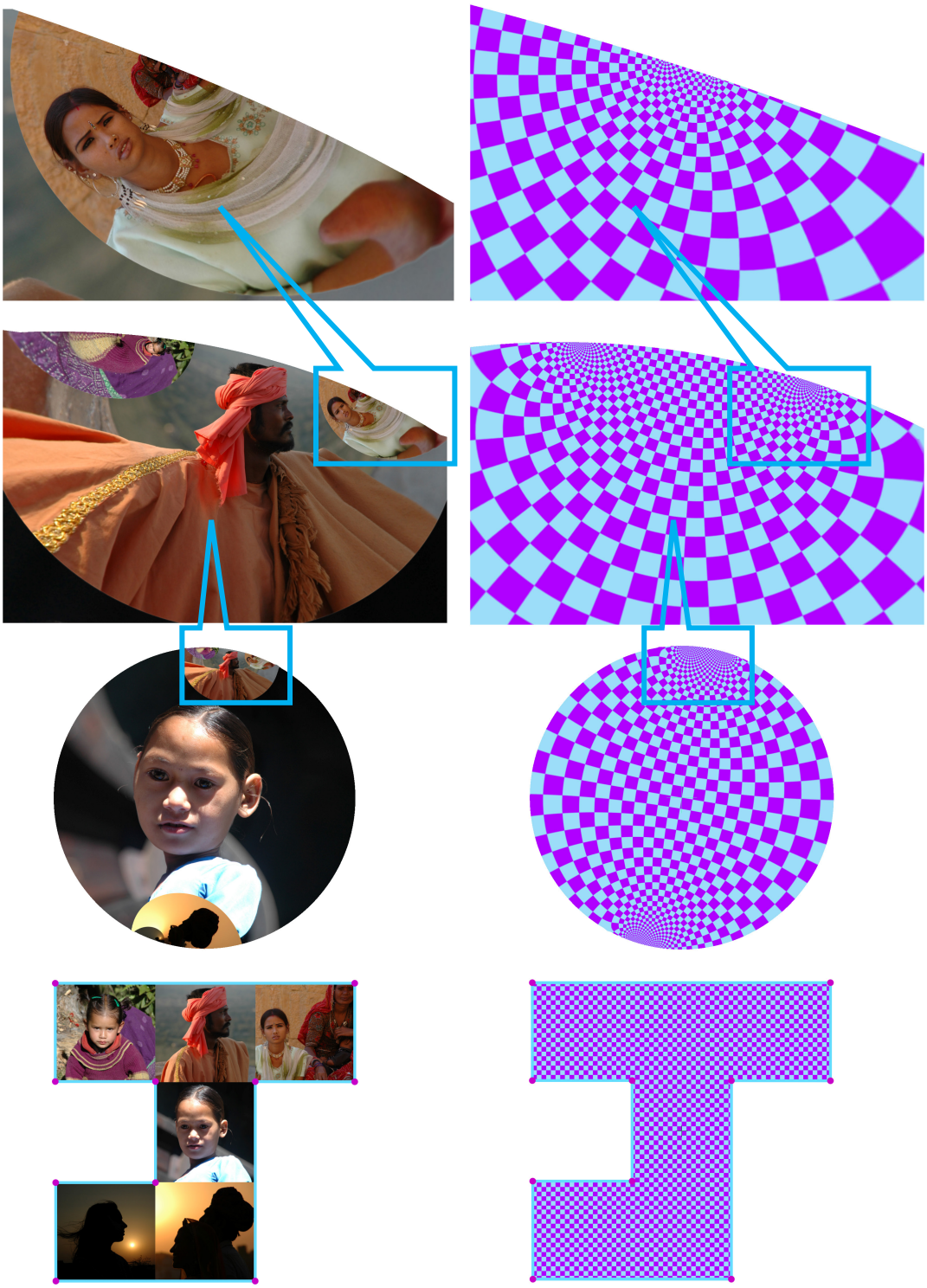


Figure 4.10: Riemann map of a polygon to a disk computed using Hilbert coordinates. (bottom row) polygon with image texture and a checker board texture. (3rd row) mapping to disk. (top two rows) zoom in.

5 Conclusions and Discussion

In this work we suggested several approaches for addressing the challenging task of detail preserving shape deformation. We first tackle the problem of deforming and animating a discrete surface represented as a triangle mesh. The deformation process is guided by a set of example shapes and is controlled by an intuitive skeletal structure. We were able to synthesis realistic deformations which preserve fine geometrical details of the shape as well as conform to the characteristic of the shape even under extreme deformations.

We continued our research by taking a somewhat different approach based on space deformation. Allowing barycentric coordinates to be complex-valued functions, we created the new concept of complex barycentric coordinates. We derive several specific recipes for such coordinates based on the Cauchy transform. The most fundamental one is the Cauchy coordinates which was shown to be equivalent to the Green coordinates. Such coordinates has the potential to produce detail preserving planar space deformations.

We further showed how to generate pure conformal maps that conforms to exact prescription of the angular change along the boundary of the domain. Generalizing the previously derived Cauchy coordinates to have quadratic precision as well as to accommodate singularities, we were able to derive new type of complex coordinates which we name the Hilbert coordinates. Using this type of coordinates leads to an efficient and accurate way for creating foldovers-free shape preserving planar mappings. Our framework also has a remarkable inherent ability to interpolate and animate shapes in a shape preserving way such that each in-between shape is guaranteed to be conformal as well.

While we hope that we were able to push the state-of-the-art a little further toward better shape deformation algorithms, there are still unresolved issues and open questions to answer. Complex barycentric coordinates has many appealing properties and were shown to be extremely useful for creating planar deformations. Since real-valued barycentric coordinates has been applied to a diverse range of applications in geometric processing and data interpolation, it is then interesting to investigate the use of complex coordinates for other uses beside deformation. In particular, it will be interesting to find the connection to surface parameterization algorithms. Another future research direction is the generalization to 3D. Complex numbers has several extensions to higher dimensions however, conformal maps

rarely exist in 3D which hints that straight forward generalization to 3D does not exists. The work of [BCWG09a; BCWG09b] provides some insights however, even if we relax the conformality requirements, it is not clear whether it is possible to generate shape preserving injective maps which conforms to specific boundary condition.

Appendix A
Various Proofs for Cauchy-Green Coordinates

Complex barycentric coordinates – similarity reproduction

Theorem 1: Complex barycentric coordinates $k_j(z)$ reproduce similarity transformations, i.e:

$$\sum_{j=1}^{n} k_j(z) T\left(z_j\right) = T\left(z\right)$$

where T is a 2D similarity transformation.

Proof:

Similarity transformations can be represented using a linear polynomial over the complex plane in the following way. If the similarity transformation T consists of rotation by positive angle θ, uniform scale s and a translation $t = t_x + it_y$, then according to the rules of complex numbers:

$$T(x+iy) = T(z) = se^{i\theta}z + t = \alpha z + \beta$$

Where α and β are complex numbers. Since complex barycentric coordinates reproduce linear and constant functions by definition, we have:

$$\sum_{j=1}^{n} k_j(z) T\left(z_j\right) = \sum_{j=1}^{n} k_j(z)(\alpha z_j + \beta) = \alpha \sum_{j=1}^{n} k_j(z) z_j + \beta \sum_{j=1}^{n} k_j(z) = \alpha z + \beta = T\left(z\right)$$

Complex barycentric coordinates – affine reproduction

Theorem 2: Complex barycentric coordinates $k_j(z)$ reproduce affine transformations of z_j which contain non-uniform scale if and only if the complex conjugates of the coordinates $k_j(z)$ also have linear precision, meaning:

$$\sum_{j=1}^{n} \overline{k}_j(z) z_j = z \tag{1}$$

Proof:

Let T be a 2D transformation which scales the x and y axes non-uniformly: $T = \begin{pmatrix} 2\alpha & 0 \\ 0 & 2\beta \end{pmatrix}$, $\alpha \neq \beta$, $\alpha,\beta \in R$

When applied to complex numbers, T can be described as:

$$T(z) = T(x+iy) = 2\alpha x + i2\beta y = \alpha(z+\overline{z}) + \beta(z-\overline{z})$$

To show that the transform reproduces affine transformations we need to show:

$$\sum_{j=1}^{n} k_j(z)\left(T(z_j)\right) = T(z)$$

$$\sum_{j=1}^{n} k_j(z)\left(\alpha(z_j+\overline{z}_j) + \beta(z_j-\overline{z}_j)\right) = T(z)$$

$$(\alpha+\beta)\sum_{j=1}^{n} k_j(z) z_j + (\alpha-\beta)\sum_{j=1}^{n} k_j(z)\overline{z}_j = \alpha(z+\overline{z}) + \beta(z-\overline{z})$$

$$(\alpha+\beta) z + (\alpha-\beta)\sum_{j=1}^{n} k_j(z)\overline{z}_j = (\alpha+\beta) z + (\alpha-\beta)\overline{z}$$

Since $\alpha \neq \beta$ we have:

$$\sum_{j=1}^{n} k_j(z)\overline{z}_j = \overline{z}$$

and conjugating both sides gives:

$$\sum_{j=1}^{n} \overline{k}_j(z) z_j = z$$

Hence, complex barycentric coordinates reproduce non-uniform scale if and only if (1) holds. Any complex barycentric coordinates reproduce similarity transformations by Theorem 1.

Those two facts together imply that complex barycentric coordinates reproduce affine transformations if and only if (1) is satisfied.

Complex three point coordinates

Theorem 5: Any set of complex functions $k_j(z)$ which satisfy

$$\sum_{j=1}^{n} k_j(z)(z_j - z) = 0$$

can be represented in the form:

$$\sum_{j=1}^{n}\left(m_j(z)\frac{B_{j+1}(z)}{A_{j+1}} - m_{j-1}(z)\frac{B_{j-1}(z)}{A_j}\right)(z_j - z) = 0$$

where $m_j(z)$ are arbitrary complex functions over Ω.

Proof:

Let m_1 be an arbitrary complex function: m_1: $\Omega \rightarrow$ C, and define m_j , $j = 2..n$, recursively:

$$m_j(z) = \frac{A_{j+1}A_j k_j(z) + m_{j-1}B_{j-1}(z)A_{j+1}}{B_{j+1}(z)A_j}$$

Define:

$$\hat{k}_j(z) = m_j(z)\frac{B_{j+1}(z)}{A_{j+1}} - m_{j-1}(z)\frac{B_{j-1}(z)}{A_j}$$

We claim that $\hat{k}_j(z) = k_j(z)$. This holds by construction for $j = 2..n$. Let us show that it also holds for $j = 1$.

From Theorem 4 (in chapter 3) we have:

$$\sum_{j=1}^{n} \hat{k}_j(z)(z_j - z) = 0$$

hence:

$$\hat{w}_1(z)(z_1 - z) = \sum_{j=2}^{n} \hat{w}_j(z)(z_j - z) = \sum_{j=2}^{n} w_j(z)(z_j - z) = w_1(z)(z_1 - z)$$

Since $z_1 - z \neq 0$, we have that $\hat{w}_1(z) = w_1(z)$, which concludes the proof.

Limits of discrete Cauchy transform on the boundary

Theorem A1: The limits of the discrete Cauchy transform $C_j(w)$ have constant and linear precision, meaning: for all $w \in S$, the following holds:

(a) $\sum_{j=1}^{n} C_j(w) = 1$

(b) $\sum_{j=1}^{n} C_j(w) z_j = w$

Proof:

(a) By definition we have:

$$\sum_{j=1}^{n} C_j(w) \triangleq \sum_{j=1}^{n} \left(\lim_{w^{in} \to w} C_j(w^{in}) \right)$$

Using the rules of limits, we get:

$$\sum_{j=1}^{n} \left(\lim_{w^{in} \to w} C_j(w^{in}) \right) = \lim_{w^{in} \to w} \sum_{j=1}^{n} C_j(w^{in}) = \lim_{w^{in} \to w} 1 = 1$$

which concludes the proof of (a).

(b) Again, by definition:

$$\sum_{j=1}^{n} C_j(w) z_j \triangleq \sum_{j=1}^{n} \left(\lim_{w^{in} \to w} C_j(w^{in}) \right) z_j$$

Using the rules of limits, we get:

$$\sum_{j=1}^{n} \left(\lim_{w^{in} \to w} C_j(w^{in}) \right) z_j = \lim_{w^{in} \to w} \sum_{j=1}^{n} C_j(w^{in}) z_j = \lim_{w^{in} \to w} w^{in} = w$$

which concludes the proof of (b).

Constant and linear precision of discrete Szegö coordinates

Theorem A2: The discrete Szegö coordinates $G_j(w)$ have constant and linear precision, meaning: for all $w \in \Omega$ the following holds:

(a) $\sum_{j=1}^{n} G_j(w) = 1$

(b) $\sum_{j=1}^{n} G_j(w) z_j = w$

Proof:

Using the definition of the Szegö coordinates:

$$\sum_{j=1}^{n} G_j(w) = \sum_{j=1}^{n}\sum_{k=1}^{n} C_k(w) M_{k,j} = \sum_{k=1}^{n} C_k(w)\left(\sum_{j=1}^{n} M_{k,j}\right) = \sum_{k=1}^{n} C_k(w)\left(MI_{nx1}\right)_k \qquad (2)$$

where I is a column vector of ones.

Since H is a sampling matrix over the polygon, each of its rows contains t and 1-t for some t. Hence, the rows of H sum to unity, and we have:

$$H_{kxn} I_{nx1} = I_{kx1} \qquad (3)$$

Since C_b have constant precision according to Theorem A1:

$$CI_{nx1} = I_{kx1}$$

Multiplying by the pseudo-inverse on both sides:

$$I_{nx1} = \left(C^* C\right)^{-1} C^* I_{kx1} \tag{4}$$

Plugging in the expression for M, and (3) and (4):

$$MI_{nx1} = (C^* C)^{-1} C^* HI_{nx1} = (C^* C)^{-1} C^* I_{kx1} = I_{nx1}$$

And back to (2):

$$\sum_{j=1}^{n} G_j(w) = \sum_{k=1}^{n} C_k(w)\left(MI_{nx1}\right) = \sum_{k=1}^{n} C_k(w) = 1$$

This completes the proof of (a).

Moving to (b), we have:

$$\sum_{j=1}^{n} G_j(w) z_j = \sum_{j=1}^{n}\sum_{k=1}^{n} C_k(w) M_{k,j} z_j = \sum_{k=1}^{n} C_k(w) \sum_{j=1}^{n} M_{k,j} z_j = \sum_{k=1}^{n} C_k(w)\left(Mz\right)_k \tag{5}$$

where z is the complex vector $z = (z_1, z_2, \ldots, z_n)$.

Since C is reproducing according to Theorem A1, i.e.:

$$\sum_{j=1}^{n} C_j(w) z_j = w$$

for $w \in S$, we have that:

$$Cz = Hz$$

Multiplying both sides by the pseudo-inverse of C we get:

$$z = \left(C^* C\right)^{-1} C^* Hz$$

Plugging in the definition of M we have: $z = Mz$. Note that this means that z is an eigenvector of M.

Plugging this back into (5) gives:

$$\sum_{j=1}^{n} G_j(w) z_j = \sum_{k=1}^{n} C_k(w) \left(Mz\right)_k = \sum_{k=1}^{n} C_k(w) z_k = w$$

which completes the proof of (b).

Second derivatives of the Cauchy transform on the boundary

The second derivatives of the discrete Cauchy transform are:

$$g''(z) = \sum_{j=1}^{n} w_j(z) z_j$$

$$w_j(z) = \frac{1}{2\pi i}\left(\frac{1}{B_{j-1}(z)B_j(z)} - \frac{1}{B_j(z)B_{j+1}(z)}\right)$$

Since the logarithm function has been eliminated, the derivatives are no longer multi-valued, and except at the vertices, the second derivative is well-defined, hence, for all $z \in S$, $z \neq \{z_1, z_2, \ldots, z_n\}$, the following holds:

$$w_j(z) = \lim_{z^{in} \to z} w_j(z^{in}) = w_j(z)$$

Theorem A3: The second derivatives of the Cauchy transform satisfy: for all $z \in S$, such that $m \notin \{z_1, z_2, \ldots, z_n\}$, the following holds:

(a) $\sum_{j=1}^{n} w_j(z) = 0$

(b) $\sum_{j=1}^{n} w_j(z) z_j = 0$

Proof:

(a) From the definition of w_j we have:

$$\sum_{j=1}^{n} w_j(z) = \frac{1}{2\pi i} \sum_{j=1}^{n} \left(\frac{1}{B_{j-1}(z)B_j(z)} - \frac{1}{B_j(z)B_{j+1}(z)}\right)$$

Splitting into two sums, and changing the summation index gives:

$$\sum_{j=1}^{n} w_j(z) = \frac{1}{2\pi i}\left(\sum_{j=1}^{n}\frac{1}{B_{j-1}(z)B_j(z)} - \sum_{j=1}^{n}\frac{1}{B_j(z)B_{j+1}(z)}\right) = \frac{1}{2\pi i}\left(\sum_{j=1}^{n}\frac{1}{B_{j-1}(z)B_j(z)} - \sum_{j=1}^{n}\frac{1}{B_{j-1}(z)B_j(z)}\right) = 0$$

(b) Again, from the definition of w_j:

$$\sum_{j=1}^{n} w_j(z)z_j = \frac{1}{2\pi i}\sum_{j=1}^{n}\left(\frac{1}{B_{j-1}(z)B_j(z)} - \frac{1}{B_j(z)B_{j+1}(z)}\right)z_j = \frac{1}{2\pi i}\sum_{j=1}^{n}\left(\frac{z_j}{B_{j-1}(z)B_j(z)} - \frac{z_{j-1}}{B_{j-1}(z)B_j(z)}\right)$$

By definition: $B_j = z_j - z$, hence:

$$z_j - z_{j-1} = z_j - z - \left(z_{j-1} - z\right) = B_j - B_{j-1}$$

Plugging this back in the previous expression:

$$\sum_{j=1}^{n} w_j(z)z_j = \frac{1}{2\pi i}\sum_{j=1}^{n}\left(\frac{B_j(z) - B_{j-1}(z)}{B_{j-1}(z)B_j(z)}\right) = \frac{1}{2\pi i}\sum_{j=1}^{n}\left(\frac{1}{B_{j-1}(z)} - \frac{1}{B_j(z)}\right) = \frac{1}{2\pi i}\left(\sum_{j=1}^{n}\frac{1}{B_{j-1}(z)} - \sum_{j=1}^{n}\frac{1}{B_j(z)}\right) = 0$$

which completes the proof.

Constant and linear precision of Point-2-Point Cauchy coordinates

Theorem A4: The point-to-point Cauchy coordinates, with positional constraints $f(w_k) = f_k$ have constant and linear precision. Meaning: for all $m \in \Omega$ the following holds:

(a) $\sum_{j=1}^{p} D_j(m) = 1$

(b) $\sum_{j=1}^{p} D_j(m)w_j = m$

Proof:

Going back to the definition of the point-to-point Cauchy coordinates:

$$\sum_{j=1}^{p} D_j(m) = \sum_{j=1}^{p}\sum_{k=1}^{n} C_k(m)N_{k,j} = \sum_{k=1}^{n} C_k(m)\sum_{j=1}^{p} N_{k,j} = \sum_{k=1}^{n} C_k(m)\left(NI_{px1}\right)_k$$

$$\sum_{j=1}^{p} D_j(m) w_j = \sum_{j=1}^{p} \left(\sum_{k=1}^{n} C_k(m) N_{k,j} \right) w_j = \sum_{k=1}^{n} C_k(m) \sum_{j=1}^{p} N_{k,j} w_j = \sum_{k=1}^{n} C_k(m) \left(Nw \right)_k$$

As was the case for the Szegö coordinates, to prove constant and linear precision, it is enough to show that:

$$NI_{px1} = I_{nx1}$$

$$Nw = z$$

Let C be the matrix, whose j-th column is $C_j(w_k)$, where $w_1, w_2, \ldots, w_p \in \Omega$ are the point constraints, and let W be the matrix whose j-th column is $W_j(m)$, where $m=Hz$ are the samples of the polygon. Since the discrete Cauchy coordinates are constant reproducing, we have:

$$CI_{nx1} = I_{px1}$$

In addition, as we showed in Theorem A3:

$$WI_{nx1} = 0_{kx1}$$

Where 0_{kx1} is a column vector of k zeroes, where k is the number of samples on the boundary.

Hence, we have:

$$AI_{nx1} = \begin{pmatrix} C \\ \lambda W \end{pmatrix} I_{nx1} = \begin{pmatrix} I_{px1} \\ 0_{kx1} \end{pmatrix}$$

Multiplying both sides by the pseudo-inverse of A, we have:

$$I_{nx1} = A^{+} \begin{pmatrix} I_{px1} \\ 0_{kx1} \end{pmatrix}$$

Since N is defined to be the first p columns of A^{+}, we get: $I_{nx1} = NI_{px1}$, as required.

In a similar fashion, since C is linear reproducing, combined with Theorem A3, we have:

$$Az = \begin{pmatrix} C \\ \lambda W \end{pmatrix} z = \begin{pmatrix} w_{px1} \\ 0_{kx1} \end{pmatrix}$$

Again, multiplying both sides by the pseudo-inverse of A, and using the definition of N we get:

$$z = A^{+}\begin{pmatrix} w_{px1} \\ 0_{kx1} \end{pmatrix} = Nw$$

which concludes the proof.

The MLS complex barycentric coordinates

When inspecting the MLS expression for the "as-similar-as-possible" deformation, it is relatively straight-forward to see that all the expressions can be replaced by their complex representations. Let $p_i \in \Omega$ be the positions of the constrained points, and $q_i \in \mathbb{C}$ their target position. Let the following expressions be defined as in the MLS [SMW06] paper:

$$w_i(z) = \frac{1}{|p_i - z|^{2\alpha}} \quad , \quad w^*(z) = \sum_i w_i(z) \quad , \quad p^*(z) = \frac{1}{w^*(z)} \sum_i w_i(z) p_i \quad , \quad \hat{p}_i(z) = p_i - p^*(z)$$

$$, \quad q^*(z) = \frac{1}{w^*(z)} \sum_i w_i(z) q_i \quad , \quad \hat{q}_i(z) = q_i - q^*(z)$$

In addition, let:

$$\mu(z) = \sum_i w_i(z)\, \hat{p}_i(z) \overline{\hat{p}_i(z)} \quad , \quad A_i(z) = w_i(z)\, \hat{p}_i \overline{(z - p^*(z))}$$

where $\bar{z}$ is the conjugate of z.

The MLS deformation is defined as:

$$f_{mls}(z) = \sum_i \hat{q}_i(z) \left(\frac{1}{\mu(z)} \overline{A_i(z)} \right) + q^*(z)$$

Plugging in our expressions:

$$f_{mls}(z) = \sum_i \left(q_i - \frac{1}{w^*(z)} \sum_j w_j(z) q_j \right) \left(\frac{1}{\mu(z)} \overline{A_i(z)} \right) + \frac{1}{w^*(z)} \sum w_i(z) q_i$$

Rearranging, to obtain the coefficients of q_i:

$$f_{mls}(z) = \sum_i \left(q_i - \frac{1}{w^*(z)} \sum_j w_j(z) q_j \right) \frac{\overline{A_i(z)}}{\mu(z)} + \frac{1}{w^*(z)} \sum w_i(z) q_i =$$

$$= \sum_i q_i \frac{\overline{A_i(z)}}{\mu(z)} - \sum_i \left(\frac{1}{w^*(z)} \sum_j w_j(z) q_j \right) \frac{\overline{A_i(z)}}{\mu(z)} + \frac{1}{w^*(z)} \sum w_i(z) q_i =$$

$$= \sum_i q_i \left(\frac{\overline{A_i(z)}}{\mu(z)} + \frac{w_i(z)}{w^*(z)} \right) - \frac{1}{w^*(z)} \sum_j w_j(z) q_j \sum_i \frac{\overline{A_i(z)}}{\mu(z)}$$

Changing the summation indices, and rearranging again:

$$f_{mls}(z) = \sum_i q_i \left(\frac{\overline{A_i(z)}}{\mu(z)} + \frac{w_i(z)}{w^*(z)} \right) - \frac{1}{w^*(z)} \sum_i w_i(z) q_i \sum_j \frac{\overline{A_j(z)}}{\mu(z)} =$$

$$= \sum_i q_i \left(\frac{\overline{A_i(z)}}{\mu(z)} + \frac{w_i(z)}{w^*(z)} - \frac{w_i(z)}{w^*(z)} \sum_j \frac{\overline{A_j(z)}}{\mu(z)} \right)$$

Finally:

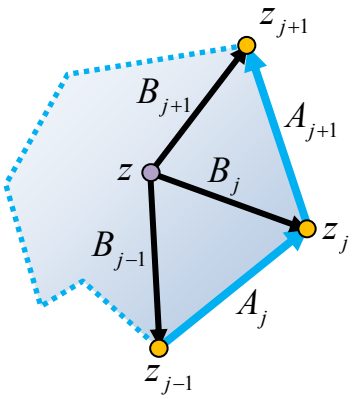
$$f_{mls}(z) = \sum_i M_i(z) q_i$$

$$M_i(z) = \frac{w_i(z)}{w^*(z)} \left(1 - \sum_j \frac{\overline{A_j(z)}}{\mu(z)} \right) + \frac{\overline{A_i(z)}}{\mu(z)}$$

Appendix B
Limits of the discrete Cauchy-Green coordinates

Notations:

$$e_j = (z_{j-1}, z_j), \quad e_{j+1} = (z_j, z_{j+1})$$
$$A_j = z_j - z_{j-1}, \quad A_{j+1} = z_{j+1} - z_j$$
$$B_j(z) = z_j - z$$

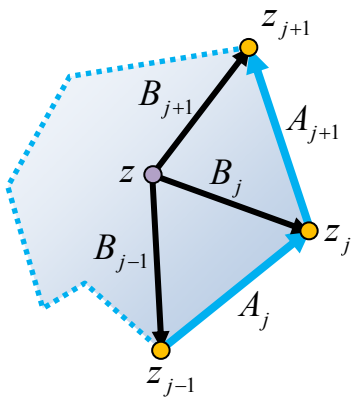


$$C_j(z) = \frac{1}{2\pi i}\begin{cases} \dfrac{B_{j+1}(z)}{A_{j+1}}\log\left(\dfrac{B_{j+1}(z)}{B_j(z)}\right) - \dfrac{B_{j-1}(z)}{A_j}\log\left(\dfrac{B_j(z)}{B_{j-1}(z)}\right) & z \notin \{e_j, e_{j+1}\} \\ \left(1+\dfrac{(1-t)A_j}{A_{j+1}}\right)\log\left(1+\dfrac{A_{j+1}}{(1-t)A_j}\right) + t\left(\log\dfrac{1-t}{t} + \pi i\right) & z \in e_j \Rightarrow \; z = z_{j-1}(1-t) + z_j t, \; t \in (0,1) \\ (1-t)\left(\log\dfrac{1-t}{t} + \pi i\right) - \left(1+\dfrac{t\cdot A_{j+1}}{A_j}\right)\log\left(1+\dfrac{A_j}{t\cdot A_{j+1}}\right) & z \in e_{j+1} \Rightarrow \; z = z_j(1-t) + z_{j+1}t, \; t \in (0,1) \\ \left(1+\dfrac{A_j}{A_{j+1}}\right)\log\left(1+\dfrac{A_{j+1}}{A_j}\right) & z = z_{j-1} \\ \log\dfrac{|A_{j+1}|}{|A_j|} + i\sphericalangle z_{j-1}z_j z_{j+1} & z = z_j \\ -\left(1+\dfrac{A_{j+1}}{A_j}\right)\log\left(1+\dfrac{A_j}{A_{j+1}}\right) & z = z_{j+1} \end{cases}$$

Appendix C
First Derivative of Cauchy-Green Coordinates and its limits

Notations:

$$e_j = (z_{j-1}, z_j), \quad e_{j+1} = (z_j, z_{j+1})$$
$$A_j = z_j - z_{j-1}, \quad A_{j+1} = z_{j+1} - z_j$$
$$B_j(z) = z_j - z$$

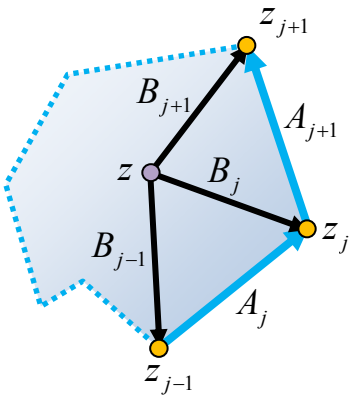


$$\frac{dC_j}{dz}(z) = \frac{1}{2\pi i}\begin{cases} \frac{1}{A_{j+1}}\log\left(\frac{B_j(z)}{B_{j+1}(z)}\right) + \frac{1}{A_j}\log\left(\frac{B_j(z)}{B_{j-1}(z)}\right) & z \notin \{e_j, e_{j+1}\} \\ \frac{1}{A_j}\left(\log\frac{1-t}{t} + \pi i\right) - \frac{1}{A_{j+1}}\log\left(1 + \frac{A_{j+1}}{(1-t)A_j}\right) & z \in e_j \Rightarrow \; z = z_{j-1}(1-t) + z_j t, \; t \in (0,1) \\ \frac{1}{A_{j+1}}\left(\log\frac{t}{1-t} - \pi i\right) - \frac{1}{A_j}\log\left(1 + \frac{A_j}{t \cdot A_{j+1}}\right) & z \in e_{j+1} \Rightarrow \; z = z_j(1-t) + z_{j+1} t, \; t \in (0,1) \\ \textit{undefined} & z = z_{j-1} \\ \textit{undefined} & z = z_j \\ \textit{undefined} & z = z_{j+1} \end{cases}$$

Appendix D
Second Derivative of Cauchy-Green Coordinates and its limits

Notations:

$$e_j = (z_{j-1}, z_j), \quad e_{j+1} = (z_j, z_{j+1})$$
$$A_j = z_j - z_{j-1}, \quad A_{j+1} = z_{j+1} - z_j$$
$$B_j(z) = z_j - z$$

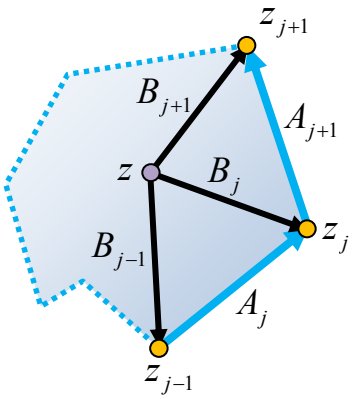


$$\frac{d^2 C_j}{dz^2}(z) = \frac{1}{2\pi i}\begin{cases} \dfrac{1}{B_{j-1}(z)B_j(z)} - \dfrac{1}{B_j(z)B_{j+1}(z)} & z \notin \{e_j, e_{j+1}\} \\ \dfrac{A_j + A_{j+1}}{\left(t^2 - t\right)A_j{}^2\left(\left(1-t\right)A_j + A_{j+1}\right)} & z \in e_j \Rightarrow \; z = z_{j-1}(1-t) + z_j t, \; t \in (0,1) \\ \dfrac{A_j + A_{j+1}}{\left(t - t^2\right)A_{j+1}^2\left(tA_{j+1} + A_j\right)} & z \in e_{j+1} \Rightarrow \; z = z_j(1-t) + z_{j+1} t, \; t \in (0,1) \\ \textit{undefined} & z = z_{j-1} \\ \textit{undefined} & z = z_j \\ \textit{undefined} & z = z_{j+1} \end{cases}$$

Appendix E
Third Derivative of Cauchy-Green Coordinates and its limits

Notations:

$$e_j = (z_{j-1}, z_j), \quad e_{j+1} = (z_j, z_{j+1})$$
$$A_j = z_j - z_{j-1}, \quad A_{j+1} = z_{j+1} - z_j$$
$$B_j(z) = z_j - z$$

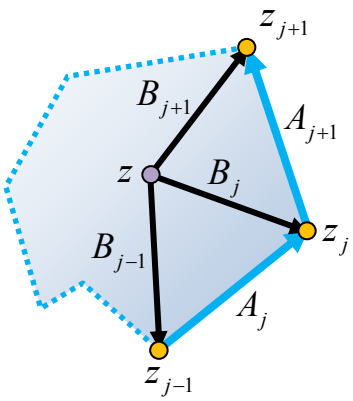


$$\frac{d^3C_j}{dz^3}(z) = \frac{1}{2\pi i}\begin{cases} \dfrac{1}{B_{j-1}(z)^2 B_j(z)} + \dfrac{1}{B_{j-1}(z) B_j(z)^2} - \dfrac{1}{B_j(z) B_{j+1}(z)^2} - \dfrac{1}{B_j(z)^2 B_{j+1}(z)} & z \notin \{e_j, e_{j+1}\} \\ \dfrac{A_j^5 t^6 + \left(-4A_j^5 - 2A_j^4 A_{j+1}\right)t^5 + \left(6A_j^5 + 6A_j^4 A_{j+1} + A_j^3 A_{j+1}^2\right)t^4 + \left(-6A_j^4 A_{j+1} - 4A_j^5 - 2A_j^3 A_{j+1}^2\right)t^3 + \left(A_j^5 + A_j^3 A_{j+1}^2 + 2A_j^4 A_{j+1}\right)t^2}{\left(3A_j^2 + 3A_j A_{j+1}\right)t^2 + \left(-2A_{j+1}^2 - 6A_j A_{j+1} - 4A_j^2\right)t + A_j^2 + A_{j+1}^2 + 2A_j A_{j+1}} & \begin{array}{l} z \in e_j \Rightarrow \\ z = z_{j-1}(1-t) + z_j t \\ t \in (0,1) \end{array} \\ \dfrac{A_{j+1}{}^5 t^6 + \left(2A_j A_{j+1}{}^4 - 2A_{j+1}{}^5\right)t^5 + \left(A_j{}^2 A_{j+1}{}^3 - 4A_j A_{j+1}{}^4 + A_{j+1}{}^5\right)t^4 + \left(-2A_j{}^2 A_{j+1}{}^3 + 2A_j A_{j+1}{}^4\right)t^3 + A_j{}^2 A_{j+1}{}^3 t^2}{\left(3A_j A_{j+1} + 3A_{j+1}{}^2\right)t^2 + \left(-2A_{j+1}{}^2 + 2A_j{}^2\right)t - A_j{}^2 - A_j A_{j+1}} & \begin{array}{l} z \in e_{j+1} \Rightarrow \\ z = z_j(1-t) + z_{j+1} t \\ t \in (0,1) \end{array} \\ \textit{undefined} & z = z_{j-1} \\ \textit{undefined} & z = z_j \\ \textit{undefined} & z = z_{j+1} \end{cases}$$

Appendix F
Limits of the generalized Cauchy coordinates

Notations:

$$e_j = \left(z_{j-1}, z_j\right), \quad e_{j+1} = \left(z_j, z_{j+1}\right)$$
$$A_j = z_j - z_{j-1}, \quad A_{j+1} = z_{j+1} - z_j$$
$$B_j(z) = z_j - z$$



Discontinuous quadratic Cauchy coordinates:

$$CV_j^-(z) = \frac{1}{2\pi i}\begin{cases} \dfrac{B_{j-1}(z)}{A_j^2}\left(\left(B_j(z)+B_{j-1}(z)\right)\log\left(\dfrac{B_j(z)}{B_{j-1}(z)}\right)-2A_j\right) & z \notin \{e_j, e_{j+1}\} \\ -t\left((1-2t)\left(\log\left(\dfrac{1-t}{t}\right)+\pi i\right)-2\right) & z \in e_j \Rightarrow \\ & z = z_{j-1}(1-t)+z_j t \\ & t \in (0,1) \\ \left(\dfrac{tA_{j+1}}{A_j}+1\right)\left(\left(\dfrac{2tA_{j+1}}{A_j}+1\right)\log\left(\dfrac{tA_{j+1}}{tA_{j+1}+A_j}\right)+2\right) & z \in e_{j+1} \Rightarrow \\ & z = z_j(1-t)+z_{j+1}t \\ & t \in (0,1] \\ 0 & z = z_{j-1} \\ \textit{undefined} & z = z_j \end{cases}$$

$$CV_j^+(z)=\frac{1}{2\pi i}\begin{cases}\dfrac{B_{j+1}(z)}{A_{j+1}^2}\left(\left(B_{j+1}(z)+B_j(z)\right)\log\left(\dfrac{B_{j+1}(z)}{B_j(z)}\right)-2A_{j+1}\right) & z\notin\{e_j,e_{j+1}\}\\ \left(\dfrac{(1-t)A_j}{A_{j+1}}+1\right)\left(\left(\dfrac{2(1-t)A_j}{A_{j+1}}+1\right)\log\left(\dfrac{A_{j+1}}{(1-t)A_j}+1\right)-2\right) & z\in e_j\Rightarrow\ z=z_{j-1}(1-t)+z_jt,\ t\in[0,1)\\ (1-t)\left((1-2t)\left(\log\left(\dfrac{1-t}{t}\right)+\pi i\right)-2\right) & z\in e_{j+1}\Rightarrow\ z=z_j(1-t)+z_{j+1}t,\ t\in(0,1)\\ \textit{undefined} & z=z_j\\ 0 & z=z_{j+1}\end{cases}$$

$$CE_j(z)=\frac{1}{\pi i}\begin{cases}\dfrac{1}{A_{j+1}^2}\left(A_{j+1}\left(B_{j+1}(z)+B_j(z)\right)-2B_{j+1}(z)B_j(z)\log\left(\dfrac{B_{j+1}(z)}{B_j(z)}\right)\right) & z\notin e_{j+1}\\ 1-2t+2t(1-t)\left(\log\left(\dfrac{1-t}{t}\right)+\pi i\right) & z\in e_{j+1}\Rightarrow\ z=z_j(1-t)+z_{j+1}t,\ t\in(0,1)\\ 1 & z=z_j\\ -1 & z=z_{j+1}\end{cases}$$

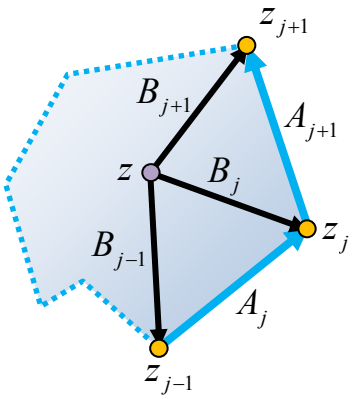
Continuous quadratic Cauchy coordinates at a vertex:

$$CV_j(z_j)=CV_j^-(z_j)+CV_j^+(z_j)=\frac{1}{2\pi i}\left(\log\left(\frac{|A_{j+1}|}{|A_j|}\right)+i\sphericalangle z_{j-1}z_jz_{j+1}\right)$$

Appendix G
First Derivative of generalized Cauchy coordinates

Notations:

$$e_j = \left(z_{j-1}, z_j\right), \quad e_{j+1} = \left(z_j, z_{j+1}\right)$$
$$A_j = z_j - z_{j-1}, \quad A_{j+1} = z_{j+1} - z_j$$
$$B_j(z) = z_j - z$$



Discontinuous quadratic Cauchy coordinate derivative:

$$DV_j^-(z) = \frac{1}{2\pi i A_j}\begin{cases} \dfrac{1}{A_j}\left(\dfrac{B_{j-1}(z)A_j}{B_j(z)} + 3A_j - \left(4B_{j-1}(z) + A_j\right)\log\left(\dfrac{B_j(z)}{B_{j-1}(z)}\right)\right) & z \notin \{e_j, e_{j+1}\} \\ 3 + (1-4t)\left(\log\left(\dfrac{t}{1-t}\right) - \pi i\right) - \dfrac{t}{1-t} & z \in e_j \Rightarrow \; z = z_{j-1}(1-t) + z_j t, \; t \in (0,1) \\ 4 + \dfrac{A_j}{tA_{j+1}} - \left(3 + \dfrac{4tA_{j+1}}{A_j}\right)\log\left(\dfrac{A_j}{tA_{j+1}} + 1\right) & z \in e_{j+1} \Rightarrow \; z = z_j(1-t) + z_{j+1}t, \; t \in (0,1] \\ \textit{undefined} & z = z_{j-1} \\ \textit{undefined} & z = z_j \end{cases}$$

$$DV_j^+(z)=\frac{1}{2\pi i A_{j+1}}\begin{cases}\frac{1}{A_{j+1}}\left(\frac{B_{j+1}(z)A_{j+1}}{B_j(z)}+3A_{j+1}-\left(4B_{j+1}(z)-A_{j+1}\right)\log\left(\frac{B_{j+1}(z)}{B_j(z)}\right)\right) & z\notin\{e_j,e_{j+1}\}\\ 4+\frac{A_{j+1}}{(1-t)A_j}-\left(3+\frac{4(1-t)A_j}{A_{j+1}}\right)\log\left(\frac{A_{j+1}}{(1-t)A_j}+1\right) & z\in e_j\Rightarrow\ z=z_{j-1}(1-t)+z_jt,\ t\in[0,1)\\ 3+\left(4t-3\right)\left(\log\left(\frac{1-t}{t}\right)+\pi i\right)-\frac{1-t}{t} & z\in e_{j+1}\Rightarrow\ z=z_j(1-t)+z_{j+1}t,\ t\in(0,1)\\ undefined & z=z_j\\ undefined & z=z_{j+1}\end{cases}$$

$$DE_j(z)=\frac{2}{\pi i A_{j+1}}\begin{cases}\left(B_{j+1}(z)+B_j(z)\right)\log\left(\frac{B_{j+1}(z)}{B_j(z)}\right)-2A_{j+1} & z\notin e_{j+1}\\ \left(1-2t\right)\left(\log\left(\frac{1-t}{t}\right)+\pi i\right)-2 & z\in e_{j+1}\Rightarrow\ z=z_j(1-t)+z_{j+1}t,\ t\in(0,1)\\ undefined & z=z_j\\ undefined & z=z_{j+1}\end{cases}$$

Continuous quadratic Cauchy coordinates derivative at a vertex:

$$DV_j(z_j)=DV_j^-(z_j)+DV_j^+(z_j)=undefined$$

Bibliography

[Ahl79] AHLFORS L.: Complex Analysis, 3rd Edition. McGraw-Hill Science, (1979).

[ASK*05] ANGUELOV D., SRINIVASAN P., KOLLER D., THRUN S., RODGERS J., DAVIS J.: SCAPE: shape completion and animation of people. ACM Trans. Graph. 24, 3 (2005).

[BBK05] BOTSCH M., BOMMES D., KOBBELT L.: Efficient linear system solvers for mesh processing. IMA Mathematics of Surfaces XI, Lecture Notes in Computer Science 3604 (2005).

[BCWG09a] BEN-CHEN M., WEBER O., AND GOTSMAN G.: Spatial Deformation Transfer. In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (2009).

[BCWG09b] BEN-CHEN M., WEBER O., AND GOTSMAN C.: Variational Harmonic Maps for Space Deformation. In ACM Trans. on Graphics Vol.28, No. 3 (Siggraph 2009).

[Bel92] BELL S.-R.: The Cauchy Transform, Potential Theory and Conformal Mapping. CRC-Press, (1992).

[Bel06] BELYAEV A.: On transfinite barycentric coordinates. Proc. Symp. Geometry Processing (2006).

[BFGS03] BOLZ J., FARMER I., GRINSPUN E., SCHRÖDER P.: Sparse matrix solvers on the GPU: conjugate gradients and multigrid. ACM Trans. Graph. 22, 3 (2003), 917–924.

[Bjö96] BJÖRK A.: Numerical Methods for Least Squares Problems. SIAM, (1996).

[Boo89] BOOKSTEIN F.L.: Principal warps: Thin-plate splines and the decomposition of deformations. IEEE Transactions on Pattern Analysis and Machine Intelligence, 11, 6, (1989).

[BPGK06] BOTSCH M., PAULY M., GROSS M., KOBBELT L.: PriMo: Coupled prisms for intuitive surface modeling. In Proceedings of the Symposium on Geometry Processing (2006).

[BS08] BOTSCH M., SORKINE O.: On linear variational surface deformation methods. IEEE Transactions on Visualization and Computer Graphics, 14, 1, (2008).

[BSPG06] BOTSCH M., SUMNER R., PAULY M., GROSS M.: Deformation transfer for detail-preserving surface editing. In Proceedings of VMV (2006).

[DMA02] DESBRUN M., MEYER M., ALLIEZ P.: Intrinsic parameterizations of surface meshes. Computer Graphics Forum, 21 (2002).

[DSP06] DER K. G., SUMNER R. W., POPOVIĆ J.: Inverse kinematics for reduced deformable models. ACM Trans. Graph. 25, 3 (2006).

[Dur04] DUREN, P.: Harmonic mappings in the plane. Cambridge University Press (2004).

[FHK06] FLOATER M. S., HORMANN K., KÒS G.: A general construction of barycentric coordinates over convex polygons. Adv. Comp. Math. 24, 1-4 (2006).

[FKR05] FLOATER M. S., KÒS G., REIMERS M.: Mean-value coordinates in 3D. Comp. Aided Geom. Design 22 (2005).

[Flo03] FLOATER M. S.: Mean-value coordinates. Comp. Aided Geom. Design 20, 1 (2003).

[Gra98] GRASSIA F. S.: Practical parameterization of rotations using the exponential map. Journal of Graphics Tools 3, 3 (1998).

[GWL*03] GOODNIGHT N., WOOLLEY C., LEWIN G., LUEBKE D., HUMPHREYS G.: A multigrid solver for boundary value problems using programmable graphics hardware. In Proceedings of Graphics Hardware (2003).

[HF06] HORMANN K., FLOATER M. S.: Mean-value coordinates for arbitrary planar polygons. ACM Trans. Graph. (2006).

[HL87] HROMADKA II, T.V., LAI, C.: The Complex Variable Boundary Element Method in Engineering Analysis, Springer-Verlag Publishers (1987).

[HS08] HORMANN, K., SUKUMAR, N.: Maximum entropy coordinates for arbitrary polytopes. Computer Graphics Forum, 27, 5 (2008).

[HSL*06] HUANG J., SHI X., LIU X., ZHOU K., WEI L.-Y., TENG S., BAO H., GUO B., SHUM H.-Y.: Subspace gradient domain mesh deformation. ACM Trans. on Graph. 25, 3 (2006).

[IMH05] IGARASHI T., MOSCOVICH T., HUGHES J.-F.: As-Rigid-As-Possible shape manipulation. ACM Trans. Graph. (Proc. SIGGRAPH),. 24, 3, (2005).

[JMD*07] JOSHI P., MEYER M., DEROSE T., GREEN B., SANOCKI T.: Harmonic coordinates for character articulation. ACM Trans. Graph. (Proc. SIGGRAPH), 26, 3 (2007).

[JP99] JAMES D.L., PAI D.K.: ArtDefo: Accurate real time deformable objects. Proc. SIGGRAPH (1999).

[JSW05] JU T., SCHAEFER S., WARREN J.: Mean value coordinates for closed triangular meshes. ACM Trans. Graph. (Proc. SIGGRAPH), 24, 3 (2005).

[KFG09] KARNI, Z., FREEDMAN, D., GOTSMAN, C.: Energy-based content-aware image deformation. Computer Graphics Forum (Proc. SGP), 28, 5 (2009).

[KJP02] KRY P. G., JAMES D. L., PAI D. K.: Eigenskin: real time large deformation character skinning in hardware. In Proceedings of the Symposium on Computer Animation (2002).

[KM04] KURIHARA T., MIYATA N.: Modeling deformable human hands from medical images. In Proceedings of the Symposium on Computer Animation (2004).

[KS06] KRAEVOY V., SHEFFER A.: Mean-value geometry encoding. International Journal of Shape Modeling 12, 1 (2006).

[Kyt95] KYTHE, K., P.: An introduction to boundary element methods. CRC Press (1995).

[LCF00] LEWIS J. P., CORDNER M., FONG N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In Proceedings of ACM SIGGRAPH (2000).

[LCOGL07] LIPMAN Y., COHEN-OR D., GAL R., LEVIN D.: Volume and shape preservation via moving frame manipulation. ACM Trans. on Graph. 26, 1 (2007).

[LKCOL07] LIPMAN Y., KOPF J., COHEN-OR D., LEVIN D.: GPU-assisted positive mean value coordinates for mesh deformations. Proc. Symp. Geometry Processing (2007).

[LLCO08] LIPMAN Y., LEVIN D., COHEN-OR D.: Green coordinates. ACM Trans. Graph. (Proc. SIGGRAPH), 27, 3 (2008).

[LPRM02] LÉVY B., PETITJEAN S., RAY N., MAILLOT J.: Least squares conformal maps for automatic texture atlas generation. Proc. SIGGRAPH (2002).

[LS08] LANGER T., SEIDEL H.-P.: Higher order barycentric coordinates. Computer Graphics Forum (Proc. Eurographics) 27 (2), (2008).

[Mer08] MERCAT C.: Discrete complex structure on surfel surfaces. Lecture Notes in Computer Science, 4992, (2008).

[MG03] MOHR A., GLEICHER M.: Building efficient, accurate character skins from examples. ACM Trans. Graph. 22, 3 (2003).

[MKB*08] MARTIN, S., KAUFMANN, P., BOTSCH, M., WICKE, M., GROSS, M.: Polyhedral finite elements using harmonic basis functions. Computer Graphics Forum 27, 5 (2008).

[PP93] PINKALL U., POLTHIER K.: Computing discrete minimal surfaces and their conjugates. Experiment. Math. 2, 1 (1993).

[RLN06] RHEE T., LEWIS J., NEUMANN U.: Real-time weighted pose-space deformation on the GPU. Computer Graphics Forum 25, 3 (2006).

[SAPH04] SCHREINER J., ASIRVATHAM A., PRAUN E., HOPPE H.: Inter-surface mapping. ACM Trans. Graph. (Proc. SIGGRAPH). 23, 3 (2004).

[SCFRC01] SLOAN P.-P. J., CHARLES F. ROSE I., COHEN M. F.: Shape by example. In Proceedings of I3D (2001).

[SCOIT05] SORKINE O., COHEN-OR D., IRONY D., TOLEDO S.: Geometry-aware bases for shape approximation. IEEE TVCG 11, 2 (2005).

[SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. ACM Trans. Graph. 23, 3 (2004).

[SMW06] SCHAEFER S., MCPHAIL T., WARREN J.: Image deformation using moving least squares. ACM Trans. Graph. (Proc. SIGGRAPH). 23, 3 (2004).

[SYBF06] SHI L., YU Y., BELL N., FENG W.-W.: A fast Multigrid algorithm for mesh deformation. ACM Trans. Graph. 25, 3 (2006).

[SZGP05] SUMNER R. W., ZWICKER M., GOTSMAN C., POPOVIĆ J.: Mesh-based inverse kinematics. ACM Trans. Graph. 24, 3 (2005).

[Tol03] TOLEDO S.: TAUCS: A Library of Sparse Linear Solvers, version 2.2. Tel-Aviv University, Available online at http://www.tau.ac.il/~stoledo/taucs/, Sept. 2003.

[WG10] WEBER O., GOTSMAN C.: Controllable Conformal Maps for Shape Deformation and Interpolation. ACM Trans. on Graphics 29, 3 (Siggraph 2010).

[WBCG09] WEBER O., BEN-CHEN M., AND GOTSMAN C.: Complex barycentric coordinates with applications to planar shape deformation. Computer Graphics Forum 28, 2, (2009). Patent pending. Günter Enderle Best Paper Award (1st place).

[WPP07] WANG R. Y., PULLI K., POPOVIĆ J.: Real-Time Enveloping with Rotational Regression. ACM Trans. Graph. 26, 3 (2007).

[WSHD07] WARREN J. D., SCHAEFER S., HIRANI A. N., DESBRUN M.: Barycentric coordinates for convex sets. Adv. Comput. Math. 27, 3 (2007).

[WSLG07] WEBER, O., SORKINE, O., LIPMAN, Y., AND GOTSMAN, C.: Context-aware skeletal shape deformation. Computer Graphics Forum 26, 3, (2007).

[YHM06] YAN H.-B., HU S.-M., MARTIN R. R.: Skeleton based shape deformation using simplex transformations. In Proceedings of CGI (2006).

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with Poisson-based gradient field manipulation. ACM Trans. Graph. 23, 3 (2004).

[ZRKS05] ZAYER R., RÖSSL C., KARNI Z., SEIDEL H.-P.: Harmonic guidance for surface deformation. In Computer Graphics Forum (2005).

תקציר המחקר

מניפולציה אינטראקטיבית של צורות בשני מימדים ושלושה מימדים היא משימה חשובה בתחום הגרפיקה הממוחשבת. האתגר הוא להיות מסוגלים לבצע שינוי גלובלי בצורה תוך כדי שימור המבנה הלוקאלי שלה. כלי דפורמציה מקבל כקלט אובייקט בצורת המקור ואוסף של אילוצים שהמשתמש יכול לערוך בצורה אינטראקטיבית. על מנת לאפשר עבודה אינטואיטיבית ונוחה עם הכלי, יש להגביל את מספר האילוצים שמסופק על ידי המשתמש למינימום. הפלט המתבקש אם כן הוא צורה חדשה המקיימת את אילוציו של המשתמש תוך כדי שמירה ככל שניתן על הפרטים הגאומטריים העדינים ואופייה של הצורה המקורית.

בעבודה זו, אנו בודקים טכניקות שונות על מנת להשיג דפורמציות של צורות המשמרות פרטים גאומטריים עדינים. ראשית, אנו מספקים אלגוריתם המשלב בין שיטות לייצוג פנימי של משטחים (בעזרת קואורדינטות דיפרנציאליות) עם שיטות מונחות מידע (פרק 2). מידת הראליסטיות הרבה של הדפורמציה מושגת על ידי שימוש בקלט נוסף המכיל אוסף של דוגמאות שונות של צורת המקור בפוזות שונות. קלט זה מנחה את האלגוריתם ביצירת ההקשר הנכון של שינויי הצורה למאפיניו הבסיסים של המודל שעליו אנו מפעילים את המניפולציה. למשל ניפוח של שרירים והופעת קפלים בגופו של אדם או בגופה של חיה. אנו משיגים מטרה זו בצורה שהינה יעילה חישובית וחסכונית בצריכת הזכרון שלה.

בהמשך, אנו מטפלים בבעיית הדפורמציה בגישה אשר מבוססת בבסיסה על דפורמציה מרחבית. דפורמציה מרחבית מעוותת את המרחב המשכן ולא את האובייקט עצמו באופן ישיר. כל אובייקט המשובץ במרחב המשכן יתעוות באופן עקיף כהשפעת לוואי. היתרון העיקרי של שיטות דפורמציה מרחבית היא הכלליות שלהן. שיטות אלה מסוגלות לטפל במגוון רחב של ייצוגים גאומטריים כגון משטחי חלוקה, משטחים חופשיים וכדומה ואינן מוגבלות לטפל ברשתות משולשיות בדידות בלבד. אחת הגישות הפופולריות ליצירת דפורמציה מרחבית עושה שימוש בקואורדינטות בריצנטריות, אולם לשימוש בקואורדינטות בריצנטריות לשם דפורמציה יש מחיר. הן אינן משמרות פרטים גאומטריים. אם כן האתגר הניצב בפנינו הוא פיתוח שיטה *משולבת* לדפורמציה מרחבית מבוססת קואורדינטות בריצנטריות אשר מצליחה בכל זאת לשמר פרטים גאומטריים.

אנו משיגים מטרה זו על ידי הכללה של מושג הקואורדינטות הבריצנטריות בדו מימד מעולם המספרים הממשיים לעולם המספרים המרוכבים (פרק 3). באופן מסורתי קואורדינטות בריצנטריות בדו מימד מוגדרות כקומבינציה לינארית של אוסף מקדמים ממשיים עם וקטורים דו-מימדים. ככאלה, הן פועלות על כל אחד משני הרכיבים של הוקטור באופן זהה. מצב זה ניתן לפרש בצורה מעט שונה שבה במקום וקטורים דו-מימדים נשתמש בקומבינציה לינארית של מספרים מרוכבים עם מקדמים ממשיים. לכן, אנו מציעים להכליל את המצב המתואר למקרה שבו גם המקדמים (הקואורדינטות הבריצנטריות) הם מספרים מרוכבים. לנקודת מבט חדשה זו מספר יתרונות. ראשית היא מובילה אל ההגדרה של קואורדינטות בריצנטריות מרוכבות אשר מאפשרת פעולה לינארית שונה על כל אחד

מרכיבי הוקטור. אבל בנוסף, היא מאפשרת שימוש בתאוריית האנליזה המרוכבת העשירה אשר מפשטת רבות את הניתוח והטיפול בתאורייה שאנו מציגים. קואורדינטות בריצנטריות מרוכבות שימושיות בעיקר לדפורמציה של צורות דו-מימדיות ותמונות. בתרחיש טיפוסי, המשתמש מגדיר עקום סגור (בדרך כלל פוליגון) המקיף את אובייקט המקור ומשנה את צורתו על ידי גרירת הקודקודים שלו כך שמתקבל עקום היעד. צעד זה מורה לאפליקציה לשנות את צורתו של אובייקט המקור בצורה הטבעית ביותר כך שיתאים את עצמו לפנים של עקום היעד. מאחר וזוהי פעולה יסודית בדפורמציה דו-מימדית, אלגוריתמים רבים עושים בה שימוש ומספר גדול של אלגוריתמים אלה מבוסס על קואורדינטות בריצנטריות. אבחנה חשובה היא שכל אלגוריתם דפורמציה דו-מימדי המבוסס על קואורדינטות בריצנטריות הינו אינווריאנטי לטרנספורמציות אפיניות. דבר זה מעודד שחזור של גזירה הנחשבת כהרסנית לפרטים גאומטריים עדינים ולכן למרות היתרונות הרבים שבשימוש בקואורדינטות בריצנטריות הן אינן אידאליות למטרה של דפורמציה. פרק 3 בעבודה זו מראה כי שימוש בקואורדינטות בריצנטריות מרוכבות הולומורפיות מוביל לדפורמציות שומרות פרטים גאומטרים. אנו מספקים מספר מתכונים ליצירת קואורדינטות בריצנטריות מרוכבות. הבסיסי שבהם הנקרא קואורדינטות בריצנטריות של Cauchy, הוא בעל נוסחא סגורה וקומפקטית המבוססת על דיסקרטיזציה של התמרת Cauchy. למרות יתרונותיו הרבים, מתכון זה אינו אופטימלי מנקודת המבט של המשתמש באפליקציה וזאת מכיוון שהתמונה של אובייקט המקור עשוייה להתרחק יתר על המידה מעקום היעד. אנו משפרים מתכון זה על ידי שימוש בהתמרת Szegö אשר על מנת לחשבה יש צורך בפתרון של מערכת משוואות לינארית. מתכון שלישי מאפשר שליטה בדפורמציה בעזרת אוסף אילוצי מקום נקודתיים המשפר את חוויית המשתמש ואף את יעילות האלגוריתם.

מחקר בשנים האחרונות התמקד בפיתוח אלגוריתמי דפורמציה אשר משמרת את הפרטים של צורת המקור בצורה מיטבית. המשמעות היא שהדפורמציה צריכה להיות חלקה, להמנע מווריאציות מיותרות ולהיות מורכבת מטרנספורמציות לוקאליות של סיבוב ולעיתים גם סילום אחיד. יש להמנע מגזירה וסילום לא אחידים ולאסור קיפול של התמונה על עצמה בכל מחיר. אלה הן בדיוק התכונות של מיפוי קונפורמי – מיפוי הרמוני חד-חד-ערכי ששני רכיביו מקיימים את משוואות Cauchy-Riemann (שלפיו שני וקטורי הגרדיינט של המיפוי ניצבים זה לזה ובעלי אותו אורך). מיפוים קונפורמים משמרים זוויות בין עקומים החותכים זה את זה, בנוסף הם משמרים אוריינטציה והם בעלי יעקוביאן חיובי ממש בכל התחום, לפיכך הם אידיאליים עבור דפורמציה של צורות גאומטריות.

בפרק 3 אנו מציגים את הרעיון החדשני של קואורדינטות בריצנטריות מרוכבות ומראים איך לייצר קואורדינטות מבוססות פונקציות הולומורפיות. ככאלה הן עשויות להוביל למיפוי קונפורמי אולם אם וכאשר הנגזרת המרוכבת הראשונה של הפונקציה מתאפסת, המיפוי איננו קונפורמי וקיפול עצמי של התמונה מתרחש. בפרק 4 אנו מציגים שיטה חדשנית המבטיחה כי המיפוי יהיה קונפורמי בכל התחום. בנוסף, אנו מאפשרים למשתמש שליטה רבה בתהליך הדפורמציה הכולל רישום מדוייק של השינוי הזוויתי של המשיק לעקום המקור בכל נקודה. הישג זה מתאפשר בעזרתן של קואורדינטות בריצנטריות מרוכבות חדשות שאנו מכנים קואורדינטות הילברט. בנוסף, אנו

מפתחים מחדש את קואורדינטות Cauchy ומכלילים אותן על מנת לאפשר דיוק ריבועי ולאפשר שחזור של פונקציות שאינן רציפות על שפת התחום.

אנו מקווים שבעבודה זו הצלחנו לקדם את המחקר צעד אחד קדימה לקראת אלגוריתמים איכותיים ויעילים יותר לפתרון בעיית הדפורמציה, אולם, בפנינו עומדות עדיין מספר שאלות ובעיות שאינן פתורות. ראשית, מאחר וקואורדינטות בריצנטריות (ממשיות) משמשות אבן בניין במספר רב של יישומים בגרפיקה ממוחשבת ועיבוד גאומטרי, היינו רוצים למצוא שימושים לקואורדינטות בריצנטריות מרוכבות גם בתחומים אחרים מלבד דפורמציה משמרת פרטים. בפרט, היינו רוצים לעמוד על טיבו של הקשר בין קואורדינטות בריצנטריות מרוכבות לבין בעיית הפרמטריזציה של רשתות בדידות. בנוסף, היינו רוצים להכליל את הקואורדינטות הבריצנטריות המרוכבות לשלושה מימדים. הכללות של מספרים מרוכבים למימדים גבוהים קיימות אך טרנספורמציות קונפורמיות בשלושה מימדים הן משפחה מצומצת ביותר, דבר המרמז כי הכללה ישירה של הקואורדינטות לא קיימת. לאחרונה, ביצענו מספר נסיונות להכליל את עבודתנו לתלת מימד תחת עידון הדרישה הקשיחה לקונפורמיות. יחד עם זאת לא ברור עדיין אם ניתן ליצור טרנספורמציות חלקות, משמרות פרטים אשר הינן חד-חד-ערכיות בשלושה מימדים ומקיימות במדויק תנאי שפה מסויימים.

המחקר נעשה בהנחיית פרופ' חיים גוטסמן בפקולטה למדעי המחשב.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

שיטות משולבות למניפולציה אינטראקטיבית של צורות

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
דוקטור לפילוסופיה

אופיר וובר

הוגש לסנט הטכניון - מכון טכנולוגי לישראל

אב התש"ע חיפה יולי 2010

שיטות משולבות למניפולציה אינטראקטיבית של צורות

אופיר וובר