Discrete Geometric Algorithms for Mesh Processing

Mirela Ben-Chen
Discrete Geometric Algorithms for Mesh Processing

Research Thesis

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

Mirela Ben-Chen

Submitted to the Senate of the Technion - Israel Institute of Technology
Sivan 5769 Haifa June 2009
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

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>..................................................................................................................</td>
<td>1</td>
</tr>
<tr>
<td>NOTATIONS</td>
<td>..............................................................................................................</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>INTRODUCTION ..........................................................................................................</td>
<td>3</td>
</tr>
<tr>
<td>1.1</td>
<td>MESHES ...................................................................................................................</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>CONFORMAL AND HARMONIC MAPS ...............................................................................</td>
<td>5</td>
</tr>
<tr>
<td>1.2.1</td>
<td>The continuous case .............................................................................................</td>
<td>6</td>
</tr>
<tr>
<td>1.2.2</td>
<td>The discrete case ..................................................................................................</td>
<td>7</td>
</tr>
<tr>
<td>1.3</td>
<td>MESH PARAMETERIZATION .......................................................................................</td>
<td>8</td>
</tr>
<tr>
<td>1.4</td>
<td>DEFORMATION AND ANIMATION ..................................................................................</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>CONFORMAL FLATTENING BY CURVATURE PRESCRIPTION AND METRIC SCALING ..............</td>
<td>13</td>
</tr>
<tr>
<td>2.1</td>
<td>INTRODUCTION ..........................................................................................................</td>
<td>13</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Previous work .......................................................................................................</td>
<td>15</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Contribution .........................................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>2.2</td>
<td>METRIC SCALING .......................................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Definitions ..........................................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Problem statement ...............................................................................................</td>
<td>18</td>
</tr>
<tr>
<td>2.2.3</td>
<td>The conformal scaling factor ...............................................................................</td>
<td>19</td>
</tr>
<tr>
<td>2.3</td>
<td>CURVATURE PRESCRIPTION ......................................................................................</td>
<td>22</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Pushing curvature around ...................................................................................</td>
<td>22</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Finding cone singularities ..................................................................................</td>
<td>24</td>
</tr>
<tr>
<td>2.4</td>
<td>IMPLEMENTATION DETAILS ......................................................................................</td>
<td>27</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Computing $\phi$ .................................................................................................</td>
<td>27</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Computing the target curvatures ........................................................................</td>
<td>28</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Computing the 2D embedding .............................................................................</td>
<td>28</td>
</tr>
<tr>
<td>2.5</td>
<td>EXPERIMENTAL RESULTS .......................................................................................</td>
<td>29</td>
</tr>
<tr>
<td>2.6</td>
<td>CONCLUSIONS AND DISCUSSION ...............................................................................</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>CHARACTERIZING SHAPE USING CONFORMAL FACTORS ........................................</td>
<td>35</td>
</tr>
<tr>
<td>3.1</td>
<td>INTRODUCTION ..........................................................................................................</td>
<td>35</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Previous work .....................................................................................................</td>
<td>36</td>
</tr>
<tr>
<td>3.2</td>
<td>THE SHAPE DESCRIPTOR .........................................................................................</td>
<td>38</td>
</tr>
<tr>
<td>3.2.1</td>
<td>The Discrete Conformal Factor ..........................................................................</td>
<td>38</td>
</tr>
<tr>
<td>3.2.2</td>
<td>The Signature ......................................................................................................</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>EVALUATION OF THE SHAPE DESCRIPTOR ..........................................................</td>
<td>40</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Pose Invariance ...................................................................................................</td>
<td>40</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Robustness to Noise ............................................................................................</td>
<td>43</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Sensitivity to Topology ......................................................................................</td>
<td>44</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Curvature Error ..................................................................................................</td>
<td>44</td>
</tr>
<tr>
<td>3.3.5</td>
<td>The Watertight Shapes Benchmark .....................................................................</td>
<td>45</td>
</tr>
<tr>
<td>3.4</td>
<td>CONCLUSIONS AND DISCUSSION ............................................................................</td>
<td>51</td>
</tr>
<tr>
<td>4</td>
<td>VARIATIONAL HARMONIC MAPS FOR SPACE DEFORMATION ....................................</td>
<td>54</td>
</tr>
<tr>
<td>4.1</td>
<td>INTRODUCTION ..........................................................................................................</td>
<td>54</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Contribution ......................................................................................................</td>
<td>56</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Previous Work ....................................................................................................</td>
<td>56</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Method Overview ...............................................................................................</td>
<td>59</td>
</tr>
<tr>
<td>4.2</td>
<td>VARIATIONAL HARMONIC MAPS ............................................................................</td>
<td>60</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Harmonic Maps from Boundary Functions .........................................................</td>
<td>60</td>
</tr>
<tr>
<td>4.2.2</td>
<td>The Energy Functional .......................................................................................</td>
<td>65</td>
</tr>
</tbody>
</table>
4.3 OPTIMIZATION ................................................................. 71
4.4 EXPERIMENTAL RESULTS ............................................. 76
  4.4.1 Comparison .............................................................. 76
  4.4.2 Locality of the deformation ........................................ 80
  4.4.3 As-Similar-As-Possible deformations ......................... 80
  4.4.4 The cage ................................................................. 81
  4.4.5 Non-articulated shapes ............................................. 82
  4.4.6 Efficiency ............................................................... 83
4.5 CONCLUSIONS AND DISCUSSION .................................. 84

5 SPATIAL DEFORMATION TRANSFER ........................................ 85
  5.1 INTRODUCTION ............................................................ 85
    5.1.1 Background and previous work ............................... 87
    5.1.2 Method overview ................................................. 89
  5.2 DEFORMATION ANALYSIS BY HARMONIC PROJECTION .... 89
  5.3 DEFORMATION SYNTHESIS BY HARMONIC RECONSTRUCTION ........................................ 93
  5.4 EXPERIMENTAL RESULTS ........................................... 95
    5.4.1 Implementation details - Caging ............................ 96
    5.4.2 Deformation Transfer Results ............................... 97
  5.5 CONCLUSIONS ........................................................... 100

6 CONCLUSIONS AND DISCUSSION .......................................... 101

APPENDIX A - THE DISCRETE CONFORMAL SCALING FACTOR ................ 102

APPENDIX B – THE VHM BASIS FUNCTIONS AND THEIR DERIVATIVES .... 105
  B.1 THE MAPPINGS ϕ AND ψ ................................................. 105
  B.2 THE GRADIENTS ............................................................. 106
  B.3 THE HESSIANS .............................................................. 106

BIBLIOGRAPHY ........................................................................ 107
List of Figures

Figure 1.1: The cow model with its marked edges (left), and vertices (right) ................................................................. 4
Figure 1.2: A vertex and its triangle-fan .......................................................................................................................... 4
Figure 1.3: Meshes with different properties. (a) Non-manifold meshes. (b) Multiple-component mesh. (c) Mesh with non triangular faces (d) closed genus-0 triangular manifold mesh. (e) Mesh with multiple boundaries. (f) Genus 1 mesh. .................................................................................................................................................. 5
Figure 1.4: An example of a continuous planar conformal map of the unit square. The marked small square on the left is mapped to the marked square on the right. (Based on an image from Wikipedia). ............................... 6
Figure 1.5: Mesh parameterization. (left) Input: 3D mesh. (right) Output: Planar parameterization with low angular and area distortion. ........................................................................................................................................... 9
Figure 1.6: Space deformation. (left) Input: giraffe shape, cage and the locations of the user’s constraints (red spheres). (right) Output: deformed giraffe, fulfilling the constraints. ............................................................................................................. 11
Figure 2.1: Texture mapped models using our conformal parameterization method, and the cone singularities which were used for the parameterization. ........................................................................................................ 13
Figure 2.2: Scale discontinuity across the cut, exhibited by parameterization methods which pre-cut the mesh. .......... 16
Figure 2.3: Flattening and texture mapping of parameterized meshes. .............................................................................. 21
Figure 2.4: Vertices with highest absolute curvature. Since discrete Gaussian curvature is local and noisy, these are not useful as singularities locations. ........................................................................................................ 25
Figure 2.5: Cone singularities computed using our method. Red (black) spheres indicate positive (negative) singularities. The number of singularities is stated, along with the total number of vertices (in parentheses). ................................................................. 27
Figure 2.6: Quasi-conformal distortion color coded over the mesh .................................................................................... 31
Figure 2.7: Comparison of the flattening of the foot model ............................................................................................ 32
Figure 2.8: Texture mapping and color coding of quasi-conformal distortion, for the meshes from Table 2.2. ................. 34
Figure 3.1: Color-coding of the conformal factor of a few poses of the Armadillo model .................................................. 35
Figure 3.2: Color-coding of the conformal factor of two hand models and two dancer models from the AIM@SHAPE shape repository (normalized to the range $[0,1]$). ................................................................. 39
Figure 3.3: Color-coding of conformal factors and histograms of the conformal factors, for three model classes in various poses (the values are normalized to the range $[0,1]$ for the color-coding). ............ 41
Figure 3.4: Color coding of conformal factors, and the histograms of the conformal factors of three springs. The two springs of the same length have similar histograms. The shorter spring has a histogram with the same shape, but different range of values. ........................................................................................................ 42
Figure 3.5: Shape retrieval results from the watertight shapes benchmark $[VtH07,GBP07]$ using the conformal factor (upper two rows) and the conformal factor normalized to $[0,1]$ (lower two rows). In both cases the first model on the upper left is the query model. The database contains exactly 20 springs. .................................................................................................................................................. 43
Figure 3.6: Color-coded conformal factors (values normalized to the range $[0,1]$), and histograms of the conformal factors, for a spring model, and a noisy version of it. .................................................................................................................................................. 43
Figure 3.7: The conformal factor for two similar models with a small topological difference. ........................................... 44
Figure 3.8: The curvature error obtained by the linear conformal factor, for a uniform target curvature. .......................... 45
Figure 3.9: Precision/recall graphs of various shape descriptors applied to the ants class in the watertight shapes benchmark $[VtH07,GBP07]$. .................................................................................................................................................. 46
Figure 3.10: Comparison of the retrieval of an ant in the watertight shapes benchmark $[VtH07,GBP07]$. The query model is in the upper left corner, followed by the retrieved shapes in decreasing order of similarity. ........................................................................................................ 48
Figure 3.11: Precision/recall graph, and the retrieval of a single model in the watertight shapes benchmark $[VtH07,GBP07]$ for the armadillo class. .................................................................................................................................................. 49
Figure 3.12: Precision/recall graph, and the retrieval of a single model in the watertight shapes benchmark $[VtH07,GBP07]$ for the heads class. .................................................................................................................................................. 50
Figure 3.13: Precision/recall graphs for some more classes from the watertight shapes benchmark $[VtH07,GBP07]$ ................................................................. 51
Figure 3.14: Retrieval results for some query shapes, one from each class of the watertight shapes benchmark. Each row is the result of a single query. The leftmost shape (highlighted in purple) is the query image. Shapes highlighted in green are shapes from the correct class (the same as the class of the query shape). Shapes highlighted in orange are shapes from wrong classes.

Figure 4.1: The Beast model enclosed in its cage (left) and its deformation using a variational harmonic map (right).

Figure 4.2: Deformation of a range-scanned model (polygon soup) using our harmonic mapping. (Right) Source model enclosed in its cage. (Left) Deformed model.

Figure 4.3: Generating a realistic muscle "bulge" effect by placing a single Jacobian constraint near the marked area, and requiring it to scale. In addition to the Jacobian constraints, we have also placed position constraints causing the hand to rotate.

Figure 4.4: The character of the Jacobian of the deformation within one slice through a vertical bar model, bent to a "U" shape. (left) Color-coding of the condition number. (right) Color-coding of the determinant.

Figure 4.5: Deformation using different numbers of anchor points. The leg of the armadillo model (left) was deformed to a bent position, using the specified number of anchor points. The top and bottom rows show different views of the same deformed shape. The source pose shows the five user constraints – red spheres are positional constraints, and black cylinders are orientation constraints.

Figure 4.6: Deformations of the Armadillo model (a) Cage and anchor locations (b) Original pose and constraints (c) Deformed pose (d,e) Another deformed pose from two different viewpoints.

Figure 4.7: The "local/global" optimization scheme is robust enough to converge to a good solution from any arbitrary initial configuration. (Left to right) the deformed shape after 1, 3, 17 and 200 iterations, starting from an arbitrary initial configuration. The graphs show the value of the energy functional vs. the number of iterations, starting from different random starting points.

Figure 4.8: Comparison of our deformation method – VHM - with ARC and ED on three deformations of the "bar" model.

Figure 4.9: The setup used for the comparisons in Figure 4.8. (top, from left to right) VHM cage and anchors, ARC cells (320) and ED deformation graph (187 vertices). (bottom) The VHM, ARC and ED constraints.

Figure 4.10: Comparison of our method (VHM) with ARC and ED on a deformation of the "Beast" model, and the setup used for the deformation. (top row) VHM cage and anchors, ARC cells (2148) and ED deformation graph (300 vertices), (middle row) the VHM, ARC and ED constraints, (bottom row) the deformed models.

Figure 4.11: Deformation of a tetrahedral mesh model of a hand. One finger is easily moved without influencing the nearby finger, even though it is close in Euclidean distance.

Figure 4.12: (top) An ASAP deformation, and the color coding of the condition number of the Jacobian of the deformation, sampled on the input cage. The graph shows the histogram of these values. (bottom) Two As-Similar-As-Possible deformations of the Beast model. Note the exaggerated hands and feet.

Figure 4.13: Two deformations using a manually built cage (left), and an automatic cage (right).

Figure 4.14: Deformation of a plate like object, using two different anchor configurations on the medial surface (left and middle). Deformation using ARC of the same model (right).

Figure 5.1: Transferring the gallop of the horse to the multiple-component robot dog.

Figure 5.2: Terminology of deformation transfer. (top left) Source reference pose. (top right) source deformed pose. (bottom left) target reference pose. (bottom right) target deformed pose – the output of the deformation transfer process.

Figure 5.3: Reconstruction error of harmonic projection, per vertex, as % of the bounding box diagonal, for two sets of poses, including 9 cats, and 48 horses. Also shown - the reference pose within its cage, and a few representative reconstructions (purple), overlaid on the original shape (pink).

Figure 5.4: Mimicking a linear blend skinning animation using a harmonic maps. (top right) Source shape in reference pose. The shape is "rigged" to be animated in Maya. (top left) target shape in reference pose. (bottom right) A deformed pose created using a skeleton and linear skinning weights. (bottom left) The pose reconstructed using a harmonic map.

Figure 5.5: Transferring the poses of a person to a multiple component polar bear, using a skeleton with 20 joints. The reference poses inside their cages are shown in the leftmost column.

Figure 5.6: Cage generation for the robot-dog model. Every row shows one iteration of the offset-reconstruction-simplification steps. The bottom right model is the cage we used for the deformation transfer in Figure 5.1.
Figure 5.7: Comparison of our spatial deformation transfer method, with DT [SP04], on a manifold triangular mesh. (top row) Source poses. (middle row) Result of DT. (Bottom row) Our result. Left most poses on each row are the reference poses enclosed in their cages.

Figure 5.8: Transferring the poses of a horse to a multiple component robot dog. The reference poses enclosed in their cages are shown in the left most column.

Figure 5.9: Transferring the poses of the cat to a multiple component model of the anatomical skeleton of a cat. The reference poses enclosed in their cages are shown in the left most column.

Figure 5.10: Transferring the poses of the person to a multiple component character of a gremlin. The reference poses enclosed in their cages are shown in the left most column.
List of Tables

Table 2.1: Curvature error relative to target curvature. .......................................................... 29
Table 2.2: Comparison between parameterization methods .................................................. 30
Table 4.1: Comparison of the rigidity error and volume change of the deformation methods. ... 79
Table 4.2: Performance measured in milliseconds on an Intel 2.67GHz i7 machine (using a single thread) with 4GB of RAM. "Solve" - time for one optimization iteration, "Def" - time for the matrix multiply in the deformation step. "Iterations" - average number of iterations a typical deformation requires to converge. .............................................................................................................................. 84
Abstract

Geometric modeling deals with representing real objects in a virtual world. A popular geometric representation is the polygonal model. For many applications the need arises to improve the model while preserving its intrinsic geometric property, that it still describes the same "real" object. In this research, we investigate a few such applications – conformal mappings of 3D models to the plane, shape retrieval, space deformation and animation transfer. Although the applications are quite different, they use similar geometric concepts and mathematical machinery, the most noticeable being the notion of conformality as a change to the shape which preserves its essence.

We first consider the problem of planar mesh parameterization. We show how a simple discretization of a classical equation for conformal maps on continuous surfaces can be applied to generate high quality planar mappings in an efficient manner. In addition, we show how the conformal factor, a function on the mesh, which is related to the local scaling the surface should undergo in order to be flattened to the plane, can be used a shape signature, for shape matching and retrieval.

Later, we address the problem of 3D shape editing and deformation, useful in animation applications. Deformation tasks are extremely time consuming, so the challenge there is "say less, do more", the user should specify as little as possible, and the deformation method should deduce the rest. Here, again, conformal and harmonic maps play an important role, as they allow the user to modify the global shape, while preserving the small details. We introduce a novel space deformation method based on harmonic maps, and show in addition how to use the same framework in order to transfer animations from one shape to another.
Notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(V,F,E)$</td>
<td>A manifold mesh, with vertices $V$, faces $F$ and edges $E$</td>
</tr>
<tr>
<td>$X_M$</td>
<td>The embedding of the mesh $M$ in $R^3$</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>A simple closed domain in $R^2$ or $R^3$</td>
</tr>
<tr>
<td>$\chi$</td>
<td>The Euler characteristic</td>
</tr>
<tr>
<td>$N(v)$</td>
<td>The 1-ring neighborhood of the vertex $v$</td>
</tr>
<tr>
<td>$L_M$</td>
<td>A metric of the mesh $M$</td>
</tr>
<tr>
<td>$K$</td>
<td>Discrete Gaussian curvature</td>
</tr>
<tr>
<td>$F_v$</td>
<td>The faces in the triangle fan of the vertex $v$</td>
</tr>
<tr>
<td>$\alpha_v^f$</td>
<td>The angle in the face $f$ near the vertex $v$</td>
</tr>
<tr>
<td>$\phi$</td>
<td>The conformal factor</td>
</tr>
<tr>
<td>$n$</td>
<td>The number of vertices in the mesh</td>
</tr>
<tr>
<td>$G(p,q)$</td>
<td>The fundamental solution to the Laplace equation in $R^d$</td>
</tr>
<tr>
<td>$\hat{n}(q)$</td>
<td>The unit normal direction to the surface at the point $q$</td>
</tr>
<tr>
<td>$a(q), b(q)$</td>
<td>The continuous functions on the boundary which induce the harmonic map</td>
</tr>
<tr>
<td>$J_f(q)$</td>
<td>The Jacobian of the mapping $f$ at the point $q$</td>
</tr>
<tr>
<td>$\hat{\psi}, \hat{\phi} : (S \times \Omega) \to R$</td>
<td>The continuous kernel functions</td>
</tr>
<tr>
<td>$f_{a,b}$</td>
<td>The harmonic map induced by the boundary maps $a$ and $b$</td>
</tr>
<tr>
<td>$\varphi(p), \psi(p) : \Omega \to R$</td>
<td>The discrete kernel functions</td>
</tr>
<tr>
<td>$a, b$</td>
<td>The discrete functions on the boundary which induce the harmonic map</td>
</tr>
<tr>
<td>$\varphi, \psi$</td>
<td>Matrices with column stacks of $\varphi$ and $\psi$</td>
</tr>
<tr>
<td>$G_\varphi, G_\psi$</td>
<td>Matrices with column stacks of $\nabla \varphi$, and $\nabla \psi$,</td>
</tr>
<tr>
<td>$H_\varphi, H_\psi$</td>
<td>Matrices with values from $Hess(\varphi)$ and $Hess(\psi)$</td>
</tr>
<tr>
<td>$E_{Rigid}$</td>
<td>The rigidity energy</td>
</tr>
<tr>
<td>$E_{Smooth}$</td>
<td>The smoothness energy</td>
</tr>
</tbody>
</table>
1 Introduction

Geometric modeling deals with representing real objects in a virtual world. One of the most popular geometric representations is the polygonal model. Such models are described as a set of vertices with locations in space, along with a connectivity graph. However accurate such a model may be, it will always remain just a model – a representation of the "real" (physical) object. For many applications the need arises to improve the model: the locations of the vertices, the connectivity graph, or even create or remove new vertices, while preserving the intrinsic geometric property of the model – that it still describes the same "real" object. The nature of the change to the model depends on the application in which the model is used. In this research, we investigate a few such applications – conformal mappings of 3D models to the plane, shape retrieval, shape deformation and animation transfer. Although the applications are quite different, similar geometric concepts and mathematical machinery are used in all of them, the most noticeable being the notion of conformality as a change to the shape which preserves its essence.

Before diving into the details of the different applications, we will give a short introduction to polygonal meshes and conformal maps, and an overview of the applications addressed in later chapters.

1.1 Meshes

Shapes processed in geometric modeling and geometry processing applications come in a variety of representations. The most popular one is the polygonal model, or a mesh. We first repeat a few well known definitions and facts about meshes, which are used later on.

A mesh $M$ is given by the sets of its vertices and faces, which we denote by $V$ and $F$ respectively:

$$V = \{v_1, v_2, ..., v_n\}, F = \left\{ f_i = (u_1, u_2, ..., u_k) \mid u_j \in V \right\}$$

Each face $f_i$ is ordered, and induces the edges $e_{ij} = (v_i, v_j)$. The number of vertices in a face can vary across the mesh, but usually faces are either triangular or quads, although it is not mandatory. If all the faces are triangles, the mesh is called a triangular mesh.
An embedding of a mesh $M$ into $\mathbb{R}^3$ is the assignment of a point in $\mathbb{R}^3$ to each vertex of the mesh: $X_M = \{x_v \in \mathbb{R}^3 \mid v \in V\}$. The edges of the mesh are sometimes known as its connectivity, and the embedding may be referred to as the geometry. Figure 1.1 shows a representative model – a cow, with its connectivity and geometry.

![Connectivity and Geometry](image)

**Figure 1.1:** The cow model with its marked edges (left), and vertices (right)

Given a vertex $v$, the set of triangles which contain $v$ are its triangle fan. The vertices of the triangle fan, other than $v$, are the 1-ring neighborhood of $v$. Figure 1.2 shows a vertex and its triangle fan.

![Vertex and Triangle-Fan](image)

**Figure 1.2:** A vertex and its triangle-fan

If the triangle fans of all the vertices have the topology of a disk then the mesh is a manifold mesh. In a manifold mesh, every edge is a neighbor of either one or two faces. Edges which are neighbors of just one face are called boundary edges. A mesh which has no boundary edges is a closed mesh. If, by walking along the edges, any vertex of the mesh can be reached from any other vertex, then the mesh is connected, and has a single component. The genus of a mesh is the number of cuts along closed curves on the mesh, which will leave it connected. For example, a sphere has genus 0 and a torus has genus 1. Figure 1.3 shows examples of a few meshes with different properties.
The genus of a single component mesh is related to the number of faces, vertices, edges and boundary loops of the mesh through the Euler characteristic $\chi$ and the Euler formula:
\[ |V| - |E| + |F| = \chi = 2 - 2g - b \]
where $g$ is the genus and $b$ is the number of boundary loops.

1.2 Conformal and harmonic maps

Conformal and harmonic maps are a basic building block for all of our applications, hence we will give a short introduction of these topics.
1.2.1 The continuous case

Continuous conformal maps are easiest to define on the plane. Given a domain $\Omega \subset \mathbb{R}^2$, and a mapping $f : \Omega \to \mathbb{R}^2$, where $f(x,y) = (u(x,y), v(x,y))$, then $f$ is a conformal map if and only if $u$ and $v$ have continuous first partial derivatives, and:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} ; \quad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$$

$$\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2} \neq 0$$

The first two equations are known as the Cauchy-Riemann equations, and the last inequality prevents degeneracy. Let $J(p)$ be the Jacobian of the map $f$ at the point $p = (x,y) \subset \Omega$:

$$J(p) = \begin{vmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{vmatrix}$$

The Cauchy-Riemann equations imply that if a map is conformal, then its Jacobian has the following structure:

$$J(p) = \begin{pmatrix} a & b \\ -b & a \end{pmatrix}$$

which is the structure of a similarity matrix in 2D.

Figure 1.4: An example of a continuous planar conformal map of the unit square. The marked small square on the left is mapped to the marked square on the right. (Based on an image from Wikipedia).
Hence, continuous planar conformal maps, are just maps which locally scale and rotate the original domain. Figure 1.4 shows an example of a continuous planar conformal map. Note that all the squares (such as the marked one for example) are mapped to rotated and scaled squares. This property makes conformal maps very attractive, as they allow to globally change a shape, while preserving the small details, which is very useful for parameterization and deformation applications.

Another appealing property of conformal maps, is that they preserve angles between intersecting curves. Using this property, conformal maps from differentiable surfaces to the plane can be defined, where the intersection angle between two curves on the surface is measured as the intersection angle on the tangent plane. As their planar counterpart, continuous conformal maps from a surface to a plane are locally a similarity transform, only rotating and scaling the local geometry on the surface.

Finally, conformal maps have the property that both of their components ($u$ and $v$) are harmonic functions: $\nabla^2 u = 0; \nabla^2 v = 0$. For the planar case, the Laplace operator is defined as 

$$\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2},$$

and on the surface the Laplacian is defined using the local surface metric.

This makes conformal maps a special case of the more general harmonic maps. Given a map $f : \Omega \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$, $f((x_1, x_2, ..., x_n)) = (y_1(x_1, ..., x_n), ..., y_n(x_1, ..., x_n))$, then $f$ is a harmonic map if and only if: $\forall i = 1..n, \nabla^2 y_i = \sum_{j=1}^{n} \frac{\partial^2 y_i}{\partial x_j^2} = 0$, meaning that each component of $f$ is a harmonic function. This defines harmonic maps on $\mathbb{R}^n$. Alternatively, one can define a harmonic map on a differentiable surface, using the local metric on the surface to define the Laplace operator.

1.2.2 The discrete case

The surfaces we deal with in geometry processing applications are usually provided as polygonal meshes, and hence are piecewise linear, and non-differentiable. However, discrete counterparts for conformal and harmonic maps can still be defined, either by discretization of the continuous notions, as is done in [HPW06], or by developing the theory from scratch specifically for discrete surfaces, as is presented in [Gli05], and in [Hir03] using Discrete Exterior Calculus.
An important concept is the discrete Laplace operator, which is used for the definition of harmonic maps on discrete surfaces. In the discrete setting, scalar functions on the mesh are usually defined at the vertices, and the Laplace operator is also defined at the vertices as:

\[ \nabla^2 v_i = \sum_{v_j \in N(v_i)} w_{ij} (v_i - v_j) \]  

(1.1)

where \( N(v_i) \) are the 1-ring neighbors of the vertex \( v_i \in V \). A commonly used discretization, which can be derived either using a finite element approach [PP93], or using the notion of primal-dual meshes [Hir03], is the so called "cotangent weights" Laplacian, which defines the weights \( w_{ij} \) as:

\[ w_{ij} = \frac{1}{2} (\cot(\alpha) + \cot(\beta)) \]

where the angles \( \alpha \) and \( \beta \) are the angles opposite to the edge \( e_{ij} = (v_i, v_j) \). This is the Laplacian discretization that we use in the rest of this work. Other discrete Laplacians exist, and the relationship between them is investigated in [WMKG07].

Using the discrete definition of the Laplacian operator, it is possible to define discrete harmonic functions as follows. Given a mesh \( M \) with a set of vertices, faces and edges, denoted by \( V, F \) and \( E \) respectively, a function \( f : V \rightarrow R \) is discrete harmonic, if and only if:

\[ \forall v \in V, \quad \nabla^2 v = 0 \]

where the Laplace operator is defined as in (1.1).

Discrete conformal maps are somewhat more complicated, and there are a few definitions known in the literature. These definitions are discussed in Chapter 2.

1.3 Mesh Parameterization

In the next chapter we address our first application – planar mesh parameterization. This application takes as input a polygonal mesh \( M \) and its embedding in \( R^3 \), and return as output an embedding of its vertices in \( R^2 \), which are sometimes known as texture coordinates. Figure 1.5 shows an example of a planar parameterization of the cow model.

Mesh parameterization has many applications, ranging from texture mapping – pasting an image on a model, to remeshing – re-sampling the surface of the mesh in order to reduce the number of polygons, or distribute them evenly across the mesh. The challenge in parameterization applications is to generate (in an efficient way) a planar mapping, which has
a low distortion. The distortion is measured in various ways, with the most popular distortion measures being angle and area distortions.

![Mesh parameterization](image)

**Figure 1.5:** *Mesh parameterization. (left) Input: 3D mesh. (right) Output: Planar parameterization with low angular and area distortion.*

If a mesh has planar faces, for example if it is a triangular mesh, then every triangle can separately be flattened to the plane, without any distortion. Such a mapping however will not be continuous across the faces of the triangles. The same vertex would have different texture coordinates on the different triangles it belongs to. Such discontinuities would make the parameterization un-useful for many applications. Hence, there is always a trade-off between the distortion and the continuity of the parameterization. For example, the cut that induced the parameterization in Figure 1.5, was specifically tailored to reduce the area distortion, while being as short as possible.

For some applications, the continuity restriction can be simplified, by requiring only continuity in *edge length*. Meaning, that although the texture coordinates on specific edges might be discontinuous, the edge lengths induced by these coordinates are the same on both sides of the cut. In the next chapter, we present a new method to generate a *discrete conformal parameterization*, which has edge length continuity. We show how a simple discretization of a classical equation for conformal maps on continuous surfaces can be applied to generate high quality angle-preserving mappings in an efficient manner. In addition, we show how to
absorb all the curvature of the mesh into a set of automatically computed points, so that the resulting area distortion of the parameterization is low.

An important part of our parameterization method, is computing the conformal factor, a function on the mesh, which is related to the local scale the surface should undergo in order to be flattened to the plane. As this function is invariant under articulation of the mesh, we show in Chapter 3 how it can be used effectively as a shape signature, for shape matching and retrieval.

1.4 Deformation and animation

In a common scenario, a 3D model is either hand-crafted or scanned from a real life object, and then deformed to created key-frames for an animation. This requires a simple and intuitive way to deform three dimensional shapes. Unlike in the parameterization application, where the inputs and outputs to the problem were well defined, here the definition of the problem is hazier. As a user is involved, the "wheel" the user uses to control the deformation process is an important part of the application. In general, every deformation method defines two things: how the user specifies the new pose, and how the deformation method achieves it. There are many different ways to specify the required deformation, but we chose to concentrate on deformations defined by placing a set of position and orientation constraints on the resulting shape. This method of deformation specification is both simple to manipulate and flexible enough to create various poses. Hence, we would like to consider the following deformation setup: we are given as input a 3D shape, and a set of position and orientation constraints, and we want to produce as output a deformed shape which fulfills these constraints.

In the parameterization application, we concentrated on shapes which are given as manifold triangular meshes. For deformation, however, this is more than often not enough, and we would like a deformation method which is more general, and can be applied to a wider variety of inputs. Hence, we define our deformation as a space deformation – instead of deforming the shape directly we deform the underlying embedding space. So, we choose to define our deformation as a mapping \( f : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \).

As in the case of parameterization, we would like to reduce the distortion of the deformation. If we consider a small cube in the source shape, we would like it to be deformed to a small cube in the deformed shape – this is equivalent to reducing angular distortion in
parameterization. In addition, we would like the local volume of the shape to be preserved after the deformation, which is similar to reducing the area distortion for parameterizations. In fact, parameterization and deformation have much in common: the geometry is given for our shape and we seek for a new geometry which fulfills some constraints: in the case of parameterization, the new geometry should be flat. For deformation, the new geometry should fulfill the user's constraints. In both cases, we would like to reduce the angular and area/volume distortion, and optimally to achieve an isometric geometry which fulfils the constraints.

Due to these similarities, it seems reasonable to adapt successful tools, such as conformal maps, from parameterization to deformation applications. Unfortunately, when considering space deformations, the underlying space is three dimensional, and not two dimensional as in parameterization. Since conformal maps in three dimensions are very limited, we use the more general harmonic maps for deformation, and enforce the distortion requirements explicitly, by minimizing an energy functional of the deformation mapping. Moreover, to allow a large variety of harmonic maps, and to simplify the solution, we confine the input shape inside a cage, a simple closed domain in \( \mathbb{R}^3 \), and define the harmonic map only on the interior of the domain. Figure 1.6 shows the setup we used for our space deformation method described in Chapter 4.

![Figure 1.6: Space deformation. (left) Input: giraffe shape, cage and the locations of the user's constraints (red spheres). (right) Output: deformed giraffe, fulfilling the constraints.](image)

As we discussed before, shapes are often deformed in order to generate frames for an animation sequence. Even given the best possible deformation method, realistic animation
sequences are hard to achieve. Consider animating a character to walk across the room – many issues need to be addressed: the character walks on a floor, hence its feet should not go "through" the floor; a female's and a male's walks are different; the walk should be different even depending on the "mood" of the character. Thus, quite a lot of time and effort are invested in generating plausible animations. This problem motivated "animation reuse", also known as "motion retargeting" and "deformation transfer". The challenge here is: once an animation of a given character is given, how can it be transferred to a new, similar character? In Chapter 5, we address this issue, and show how to use our space deformation method in order to transfer animation sequences from one shape to another.
2 Conformal Flattening by Curvature Prescription and Metric Scaling

The first application we address is conformal parameterization to the plane. In this chapter, we present a new conformal parameterization method, which is fast, automatic, and generates high quality parameterizations, with low quasi-conformal distortions. The parameterization method utilizes cone-singularities – special vertices in the mesh, which absorb the curvature of the mesh, and allow reducing the area distortion usually exhibited by conformal parameterizations. Figure 2.1 shows two models, texture mapped with a checkerboard pattern, using the texture coordinates generated by our conformal parameterization method. In addition, the figure shows the locations of the cone singularities used for the parameterization.

![Figure 2.1: Texture mapped models using our conformal parameterization method, and the cone singularities which were used for the parameterization.]

2.1 Introduction

Triangular meshes are a popular representation for 3D models. They are used in a wide range of applications, many of which require the parameterization of the model to a planar domain. Examples are texture mapping, detail mapping, morphing and remeshing, to mention just a few. Parameterization is also an important preliminary step for many geometry processing algorithms.

The main challenge for parameterization algorithms is to bound the distortion of the resulting parameterization. The distortion can be either angular– the angles between edges in
the parameter domain are very different from those in the input 3D mesh, or area distortion – large areas of the 3D mesh are mapped to small areas in the parameterization and vice-versa, or both. Angle preserving parameterizations are called \textit{conformal}, and area preserving parameterization \textit{authalic}. Most surfaces do not have a parameterization which is \textit{isometric}, meaning that both area and angles are preserved.

The main obstacle to a distortion-free parameterization is Gaussian curvature. In fact, only meshes which are \textit{developable} - have zero Gaussian curvature everywhere – can be mapped to the plane without any distortion, thus the main objective in planar parameterization is the “removal” of Gaussian curvature. Since the Gauss-Bonnet theorem dictates that the total sum of Gaussian curvature in the mesh is fixed (and determined only by the mesh topology), eliminating all Gaussian curvature is generally not possible.

If the mesh has the topology of a disk (with a boundary), one way to eliminate Gaussian curvature is to move it from the interior vertices to the mesh boundary. Thus a common technique employed by many mesh parameterization methods, when presented with an input which is topologically closed, is to first \textit{create} a boundary by cutting the mesh into one or more pieces, each of which is a topological disk. Each piece is then mapped to the plane, while trying to preserve angles, area, or some combination of the two. The curvature originating in the 3D input ultimately ends up on the new disk boundaries, namely the external angle at boundary vertices is not necessarily $\pi$. Cutting the mesh is not only a means of achieving a disk-like topology, but is also important for reducing the distortion of the resulting parameterization, and may be even advantageous for meshes which were originally equivalent to a disk. However, cutting results in discontinuities along the cuts introduced into the mesh. Since each patch is parameterized independently, there is no guarantee that edges along the cut will be mapped to edges of the same length on both sides of the cut in the parameter plane.

An alternative to cutting the mesh and creating a boundary is the introduction of \textit{cone singularities}, first proposed by Kharevych et al. [KSS06]. The main idea is as follows: Instead of introducing artificial boundaries to absorb the undesired curvature, a few vertices of the mesh are designated as \textit{cone singularities} and the entire Gaussian curvature of the mesh is concentrated at those singularities. Once this abstract sparse curvature distribution is computed, a \textit{metric} (i.e. edge lengths) having this target curvature is found, and this is subsequently realized by an \textit{embedding} in the plane (i.e. a parameterization), by cutting the
mesh such that the cone singularities are on the boundary. Note that the new metric will be different from the original 3D edge lengths. The main difference between this method, in which the cut is performed after the new metric is computed, and the older methods, in which the cut is performed before the new metric is computed, is that this method guarantees that edges on both sides of the cut will be mapped to edges of the same length. Also, for every non-singular vertex on the boundary, the sum of the curvatures on both sides of the cut will be 0. This means that the flattened version of the mesh may be “zippered” back together in the plane at these vertices. This reduces the discontinuities in the parameter plane. Unfortunately, despite their novel idea, Kharevych et al. [KSS06] do not provide an automatic way of identifying the locations and the curvatures of these cone singularities, such that the resulting parameterization will have a small distortion. Since the distortion is generally not known in advance, this is somewhat a "chicken and egg" problem. The method we propose is closest in spirit to that of [KSS06] and improves upon it in a number of ways.

2.1.1 Previous work

The body of research dedicated to the parameterization of a triangular mesh to the plane is quite vast, and a survey of it is beyond our scope. We will focus here only on recent algorithms which are most relevant to our work. The interested reader is referred to [SPR06] for a comprehensive survey of parameterization methods.

As mentioned above, many parameterization methods solve the problem in two quite separate steps: first cut the mesh to one or more disk-like pieces, and then "flatten" each piece separately. The cutting process is performed independently of the subsequent flattening process. Methods such as ABF++ [SLMB05], LinABF [ZLS07] and others [HG00, LPRM02] concentrate mostly on reducing the angular distortion when flattening given disk-like regions. These methods ignore the discontinuities generated by the cuts, as the same 3D edge might be mapped to edges of different length in the parameterization. Such a mapping will generate a change of scale in the parameterization which will be quite visible in applications such as texture mapping. Figure 2.2 illustrates this on the texture mapping of the David model, parameterized using the LinABF method [ZLS07]. The red lines are the cut generated by Seamster [SH02]. Note that any method which pre-cuts the mesh is prone to such problems.
Kharevych at al [KSS06] proposed a way to generate a planar parameterization with low area distortion without cutting the mesh first, by introducing \textit{cone singularities}. These are vertices which, at the end of the parameterization process, do \textit{not} have zero Gaussian curvature as the other vertices of the mesh. Their method is based on the concept of \textit{circle patterns}, a powerful technique for generating discrete conformal mappings. Another method for redistributing Gaussian curvature, also based on circle patterns, is discrete Ricci flow [JKG07]. Both these methods require a non-linear solver.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure2_2.png}
\caption{Scale discontinuity across the cut, exhibited by parameterization methods which \textit{pre-cut the mesh}.}
\end{figure}

The idea of using cone singularities can be probably extended to any angle-based parameterization method, such as ABF++ [SLMB05] and its recent linearized version LinABF [ZLS07]. However, the real challenge in such an approach is finding the location and target curvature of those cone singularities, such that the resulting parameterization has low distortion.

Other methods which use cone singularities, but do not cut the mesh, are the recent quad remeshing algorithms, such as Ray et al. [RLL*06], Dong et al. [DBG*06], Tong et al. [TACSD06], Kalberer et al [KNP07] and others. Some of these methods even find the singularities automatically. However, since their main goal is quad remeshing, the singularities must have curvature which is a multiple of $\pi/2$. Many of these methods try to approximate a given input \textit{frame field}, which is usually derived from the principal curvature directions. In these cases, the cone singularities are the singular points of this input field.
2.1.2 Contribution

Our contribution is twofold. First, given the locations and curvatures of a set of cone
singularities we propose a novel conformal parameterization technique, which is fast and
simple to implement, and generates results comparable to existing parameterization methods.
Our method is inspired by both discrete conformal theory, and the continuous recipe for
conformally transforming between metrics having different Gaussian curvature distributions.
Second, at the heart of our parameterization technique lies a method to compute the conformal
scaling factor by which the mesh should be locally scaled in the vicinity of a vertex in order
to achieve the target curvature at that vertex. This scaling factor can be used to automatically
determine the location and curvatures of the cone singularities, such that the resulting
parameterization will have low distortion.

Hence, we propose a simple and efficient method for the parameterization of a mesh with
arbitrary topology to the plane, with low angle and area distortions. Our method guarantees
that edge lengths will be equal, and non singular vertices’ curvature will sum to 0, across the
cuts in the parameter plane.

2.2 Metric scaling

2.2.1 Definitions

A triangular mesh \( M \) is given by the sets of its vertices, faces and edges, which we denote by
\( V,F \) and \( E \) respectively. An embedding of a mesh \( M \) is the assignment of a point in \( R^3 \) to each
vertex of the mesh: \( X_M = \{ x_v \in R^3 \mid v \in V \} \).

A (discrete) metric of a mesh \( M \) is the assignment of a positive number to each edge of the
mesh: \( L_M = \{ l_{ij} \in R^+ \mid (i,j) \in E \} \). The natural metric of a mesh embedding \( X_M \) is a metric
which uses the Euclidean edge lengths:

\[
N_{X_M} = \{ l_{ij} = \| x_i - x_j \| \mid (i,j) \in E \}
\]

The angles induced by a metric \( L_M \) are:

\[
A_{e_{uw}} = \left\{ \alpha' = \arccos \left( \frac{l_{uw}^2 + l_{vw}^2 - l_{uv}^2}{2 l_{uw} l_{vw}} \right) \right\}
\]

The (discrete) Gaussian curvature induced by a metric \( L_M \) is:
where \( F_v \) is the set of the faces in \( F \) which share the vertex \( v \). The discrete Gaussian curvature is also known as the angle defect of a vertex. A vector of Gaussian curvatures \( K \) is feasible for a mesh \( M \) if \( \sum K = 2\pi \chi(M) \), where \( \chi(M) \) is the Euler characteristic of \( M \).

### 2.2.2 Problem statement

In our setting, cone singularities are defined by prescribing a target Gaussian curvature for all the vertices of the mesh. For a planar parameterization, most of the vertices will be prescribed zero target curvature, and the cone singularities will be prescribed some nonzero target curvature. The formal definition of the problem is as follows.

**Conformal Mapping via Curvature Prescription**

Given a triangular mesh \( M \), its embedding \( X \), and feasible target Gaussian curvatures \( K \), find a metric \( L_M \) which induces the target curvatures \( K \), and is conformal to the natural metric of the mesh \( N_X \).

*Conformality* is a well defined concept when dealing with continuous surfaces, and essentially means that angles are preserved. In the discrete setting, however, the angles of the 3D embedding cannot be preserved exactly when it is flattened to 2D, since the angle sum around a vertex in the 3D embedding is arbitrary, and in 2D the angle sum must be \( 2\pi \). Thus some angle distortion is inevitable.

As described in [SPR06], there are a few different measures of discrete conformality, and different methods to achieve it. One of the recent discrete conformal mapping approaches uses circle patterns. In this approach, the 3D mesh geometry is represented as a pattern of intersecting circles, circumscribing either the faces [KSS06] or the vertices [JKG07]. Once such a pattern is defined, the conformal mapping problem reduces to seeking new radii for the
circles, such that *the intersection angles between the circles are preserved* and the Gaussian curvatures the radii induce are the required target curvatures.

Unfortunately, solving the circle patterns problem requires a complicated two-phase non-linear optimization method (albeit with a unique minimum) in [KSS06], and an iterative flow in [JKG07]. In addition, [KSS06] generates a parameterization which is a Delaunay realization, and hence requires the computation of an "*intrinsic Delaunay triangulation*" of the mesh to avoid large distortion for non-Delaunay input meshes. [JKG07] requires a non-trivial initialization of the circle pattern. So computing these conformal mappings in practice is quite complex and its runtime can be slow.

We now propose a simple linear method to solve the conformal parameterization problem.

### 2.2.3 The conformal scaling factor

Conformal mappings of Riemannian manifolds are very well understood. In the continuous setting, a conformal mapping can be achieved by applying a *scaling function* $e^{2\phi}$ to the Riemannian metric. Intuitively, this can be thought of as scaling infinitesimal patches of the surface. The Gaussian curvature change caused by such a mapping is related to the scaling function by the following Poisson equation [SY94, Chapter V]:

$$\nabla^2 \phi = K_{\text{orig}} - e^{2\phi} K_{\text{new}}$$

where $\nabla^2$ is the Laplace-Beltrami operator of the manifold, $K_{\text{orig}}$ is the Gaussian curvature of the original manifold, and $K_{\text{new}}$ is the Gaussian curvature of the manifold after the conformal mapping. The equation is non-linear, due to the factor $e^{2\phi}$. This can be attributed to the fact that continuous Gaussian curvature *scales* when the metric scales. For example, a larger sphere will have smaller curvature. Hence, to be able to compare the original and final curvature, one must scale back the final curvature to compensate. Discrete Gaussian curvature however, is not affected by uniform scaling – both a large cube and a small cube have a discrete Gaussian curvature of $\pi/2$ at the vertices. Since the initial and final curvatures are comparable without scaling, the scaling factor is redundant, and we are left with a simpler equation. In fact, in a recent paper [Bun07], Bunin showed that the equivalent equation relating the scaling function $\phi$ to the change of Gaussian curvature, in the special case that the new Gaussian curvature distribution is a sum of delta functions, is:
\[ \nabla^2 \phi = K^{\text{orig}} - K^{\text{new}} \]

When working with a discrete mesh, the natural thing to do is to approximate the continuous solution using a finite elements solution. In this case, the discrete scaling factor will be defined as a scalar function on the vertices of the mesh, and extended to the faces in a piecewise linear manner. The Laplacian is now the cotangent weights Laplacian [PP93], which is the FEM approximation to the Laplace-Beltrami operator.

Apart from the FEM interpretation, this Poisson equation has also a meaning in the pure discrete setting. Using the derivative of the cosine law, and some simple derivations, it is easy to show that for an infinitesimal change of the discrete metric (edge lengths) near a vertex \( v \), the following holds (see a proof in Appendix A):

\[ \nabla^2 \phi_v \approx k_v^{\text{new}} - k_v^{\text{orig}} \]

As in the FEM approximation to Bunin's equation, the discrete Laplacian is defined using the cotangent weights. But in contrast to that equation, this one is correct only for small changes in the metric. Note that the discrete Laplacian is typically defined to have a sign opposite that of its continuous counterpart.

Motivated by these observations, we suggest the following solution to the problem of conformal mapping via curvature prescription.

Given a mesh \( M \), its embedding \( X \), and target Gaussian curvatures \( K^T \), the required scaling factors \( e^h \) are computed by first solving the following discrete Poisson equation on the mesh vertices:

\[ \nabla^2 \phi = K^T - K^{\text{orig}} \]  \hspace{1cm} (2.1)

where \( K^{\text{orig}} \) is the Gaussian curvature induced by the natural metric of the embedding \( X \).

\( \phi \) is extended in a piecewise-linear manner to be defined over the entire mesh surface. For an edge \( (i,j) \) we therefore have: \( \phi(t) = t \phi_i + (1-t) \phi_j \), where \( t \in [0,1] \) parameterizes the edge. The scaling factor of the edge \( (i,j) \) is obtained by integrating \( \phi(t) \) over the edge:

\[ s_{ij} = \int_0^1 e^{h(t)} dt = \begin{cases} 
-\phi_i + \phi_j & \phi_i \neq \phi_j \\
\phi_j - \phi_i & \phi_i = \phi_j 
\end{cases} \]  \hspace{1cm}, \hspace{1cm} (i, j) \in E
The target metric is then computed by multiplying the original edge lengths of the embedding by the edge's scaling factors:

\[
L^T = \left\{ l^T_{ij} = l_{ij} \cdot s_{ij} \mid (i, j) \in E, l_{ij} \in N_x \right\}
\]

As we shall see in the following sections, this method is only an approximation of the true metric that we seek. The curvature induced by the target metric \( L^T \) differs from the target metric \( K^T \) by an amount that depends on the amount of distortion that is necessary for the flattening.

![Figure 2.3: Flattening and texture mapping of parameterized meshes.](image)

The final 2D embedding of the target metric is performed using linear least squares, as in the ABF++ method [SLMB05]. This way, the accumulated errors introduced by the inaccurate metric are better distributed across the mesh. Only in this step do we require that the mesh be a topological disk such that all the cone singularities – the vertices which have non-zero target Gaussian curvature – are on the boundary. In Section 3 we explain how to find the locations and curvatures of the cone singularities, and how to cut the mesh such that the singularities are on the boundary.
Figure 2.3 shows some results of using this parameterization method, given some suitable target curvatures. The cow and bunny models were pre-cut by the Seamster [SH02] algorithm to have disk-like topology. The hand and camel were parameterized by first computing the cone singularities and the cuts, as will be explained in the next sections, and then flattening them.

2.3 Curvature Prescription
In the previous section we explained how to compute a conformal metric given target Gaussian curvatures. Now we show how to determine suitable target curvatures, most of which will be zero.

The process consists of two steps: first, identify the cone singularities – those vertices which will have non-zero target Gaussian curvature, and second, determine the target curvature of these singularities. We first explain how to determine the curvature of the cone singularities, once these are identified, and then explain how to decide which vertices should be cone singularities.

2.3.1 Pushing curvature around
Given a mesh $M$, an embedding $X$, and a set of vertices $S$ designated as cone singularities, we want to assign to each vertex $s \in S$ a target Gaussian curvature $k_s$. The sole, but important, constraint is that the target curvatures should satisfy the Gauss-Bonnet condition, i.e.:

$$\sum_{s \in S} k_s = 2\pi \chi$$

where $\chi$ is the Euler characteristic of the mesh $M$. Thus we need to distribute the total Gaussian curvature induced by the original metric of the mesh among the cone singularity vertices.

Our distribution method may be thought of as an iterative process. In each step, each non-singular vertex tries to dispose of its curvature, thus equally distributes it among its neighbors. The cone singularities vertices, on the other hand, try to absorb as much curvature as possible, thus do not distribute their curvature, rather absorb the curvature passed to them. The process stops when all the curvature has been absorbed by these vertices. Since no curvature was added or removed from the system at any point in time, the total curvature is preserved and the Gauss-Bonnet condition satisfied throughout the process. This distribution process can be
modeled as an absorbing Markov chain. Each vertex \( v \in V \) is a state, and the transition probabilities from vertex \( i \) to vertex \( j \) are defined as follows:

\[
P_{ij} = \begin{cases} 
  w_{ij} & (i, j) \in E, i \notin S, \sum_j w_{ij} = 1 \\
  1 & i = j, i \in S \\
  0 & Otherwise 
\end{cases} \quad (2.2)
\]

This means that once a random walker on the graph enters a non-singular vertex, it must continue to a neighbor of that vertex. The cone singularities are the absorbing states, and a random walker arriving at a singularity must remain there. We wish to find the probabilities of winding up at the different absorbing states, depending on the initial state. Stochastic process theory [WC07], provides a closed solution for these probabilities, given in terms of the transition matrix \( P \) defined in (2.2).

Without loss of generality we reorder the vertices of the mesh, such that the cone singularities are last. Then the transition matrix \( P \) has the special structure:

\[
P = \begin{pmatrix} S_{nn} & T_{nS} \\ 0_{Sn} & I_{SS} \end{pmatrix}
\]

where \( n \) is the number of regular vertices (not cone singularities) and \( s \) is the number of cone singularities. A simple computation shows that after \( k \) time steps, the transition probabilities are given by:

\[
P^k = \begin{pmatrix} S^k & (I + S + \ldots + S^{k-1})T \\ 0 & I \end{pmatrix}
\]

Hence, as time goes to infinity, and the random walker converges to the absorbing states, we have:

\[
P = \begin{pmatrix} 0 & (I - S)^{-1}T \\ 0 & I \end{pmatrix}
\]

Subsequently, the probabilities of ending up in the different absorbing states are:

\[
G = (I - S)^{-1}T \quad (2.3)
\]
The size of $G$ is $n \times s$, the entry $G_{ij}$ representing the probability of ending up in cone singularity vertex $j$, if we started from vertex $i$. Since this is a probability matrix, all the rows sum to unity.

Using $G$, we can compute the target Gaussian curvatures of the cone singularities vertices as a function of the initial Gaussian curvatures in closed form:

$$K_{S}^{\text{New}} = K_{S}^{\text{Orig}} + G^{T} K_{Y \not\in S}^{\text{Orig}}$$  \hspace{1cm} (2.4)

In order to compute $G$ as in (2.3), observe that $P$ has the structure of the incidence matrix of the mesh, thus it is easy to check that (2.3) is equivalent to the following Poisson equation:

$$L \hat{G}_i = \delta_i$$  \hspace{1cm} (2.5)

where $L$ is a generalized Laplacian operator of the mesh, defined with weights $w_{ij}$ as in $P$, and $\delta_i$ is a column vector which is 1 at row $i$ and zero elsewhere. $G_i$, the columns of $G$, are sub-vectors of the solutions $\hat{G}_i$. Each of the $s$ $G_i$ is a function on the $n$ regular mesh vertices, also called discrete Green's functions [CY00]. The sum of the $s$ $G_i$ at each regular mesh vertex is unity (because the curvature of that vertex is distributed in those proportions to the singular vertices). This matrix $G$ has been recently used as a type of barycentric coordinates (so-called harmonic coordinates) for mesh deformation in animation applications [JMD*07].

### 2.3.2 Finding cone singularities

Given a mesh $M$ and an embedding $X$, we search for a set of vertices $S$ of cone singularities, such that when flattening the mesh with these singularities the distortion will be small. By the very definition of the problem, it can be seen that finding cone singularities is a "chicken-and-egg problem": one needs the singularities to compute the flattening, and one needs the flattening to compute the distortion, in order to decide where to place the singularities.

As is usually the case in such scenarios, we resort to an iterative process. Here we rely heavily on the fact that our flattening method is based on computing local scaling factors. In fact, the function $\phi$ computed in the previous section indicates how much distortion we can expect in a given region of the mesh.

Since scaling all the edges of the mesh by the same factor $e^{\phi}$ will preserve the Gaussian curvature, $\phi$ is unique only up to an additive constant. Hence, we may assume that $\phi$ has zero mean. Now, let us consider how the parameterization will be distorted. If $\phi$ is all zeros, the
metric does not change, resulting in zero distortion relative to the original embedding. This is possible of course, only if the mesh is a developable surface. Otherwise, the largest distortions will occur near the vertices where $\phi$ obtains its maximal and minimal values. Placing a singular vertex at the locations of extreme distortion allows curvature to accumulate at that vertex, and thus reduces the distortion in the vicinity of that vertex. A similar concept was used by Gu et al. [GGH02] for finding cut vertices by repeatedly parameterizing a mesh, and finding the triangle with the maximal distortion. In our case however, $\phi$ is an indicator for the final distortion, and there is no need to compute the full parameterization at every iteration.

Note that simply choosing the vertices having maximal absolute curvatures as singular vertices will not achieve the same effect, since the discrete curvature is a local feature, dependent only on the vertex and its neighbors, whereas $\phi$ is obtained by a global computation on the entire mesh. Figure 2.4 shows the set of vertices with highest absolute curvatures in the hand model, which are obviously not a good choice as singularities.

![Figure 2.4: Vertices with highest absolute curvature. Since discrete Gaussian curvature is local and noisy, these are not useful as singularities locations.](image)

We thus propose the following algorithm for finding the locations of the cone singularities:

1. Initialize the set of cone singularities as follows:
   - If the mesh has a boundary, designate all the boundary vertices as cone singularities.
   - If the mesh is closed and has positive (negative) Euler characteristic, select the vertex with the largest positive (negative) curvature as a cone singularity.
• If the mesh is closed and has zero Euler characteristic, the initial set of cone singularities is empty.

2. Find the target curvatures for the set $S$ using (2.4)

3. Compute $\phi$ using the Poisson equation (2.1). If $\max(\phi) - \min(\phi) > \epsilon$, where $\epsilon$ is a user-specified tolerance, or the maximal allowed number of singularities has not been reached, add two cone singularities to $S$ at the locations of $\max(\phi)$ and $\min(\phi)$ and go to step 2.

Note that since the total sum of the target curvature must be equal to the total sum of the original curvature, the initial set $S$ cannot be empty unless the mesh has genus 1, hence the need for the initialization step.
Using different values of \( \varepsilon \), one can trade off the number of cone singularities with the resulting distortion. In our experiments we observed that using \( \varepsilon = 1 \) gives reasonable results.

Figure 2.5 shows the cone singularities we found for a variety of models. Red spheres indicate singularities with positive curvature, and black spheres singularities with negative curvature. For each model we also state the number of singular vertices generated, and (in parentheses) the total number of vertices. Singularities tend to emerge in regions with large total curvature. For example, at the tips of the fingers of the hand, although each individual vertex is nearly flat, there is a total curvature of about \( 2\pi \) on the complete tip of the finger. Another example is the back of the elephant, where a single singularity emerges to account for the total curvature of the elephant's back.

### 2.4 Implementation Details

In this section we provide the details necessary to reproduce our results. Our algorithm consists of the following steps:

1. Find the cone singularities and their target curvatures.
2. Compute \( \phi \) as a solution to the Poisson equation.
3. Compute the new 2D edge lengths using \( \phi \).
4. Compute the 2D coordinates from the new lengths.

#### 2.4.1 Computing \( \phi \)

Computation of \( \phi \) involves solving the Poisson equation (2.1). We use the discrete symmetric Laplacian with cotangent weights derived from the input 3D mesh [PP93]:

\[
\omega_{ij} = 0.5(\cot(\alpha) + \cot(\beta)),
\]

where \( \alpha \) and \( \beta \) are the angles opposite edge \((i,j)\). Since the co-rank of the Laplacian of a connected mesh is 1, \( \phi \) is defined up to an additive constant. This is consistent with the fact that uniformly scaling the edges of the mesh by a scalar does not affect the Gaussian curvature. Note also, that since all the columns of the Laplacian sum to 0 and the original Gaussian curvature vector is feasible, a necessary and sufficient condition for the linear system to have a solution is that the target Gaussian curvature vector is also
feasible, i.e. the target curvatures sum to $2\pi \chi$. In this case, the right hand side of Eq. (2.1) also sums to 0, thus the co-rank of the augmented matrix of the linear system is positive, which is a sufficient condition for the existence of a solution.

Since the Laplacian is sparse and symmetric, Eq. (2.1) can be solved very efficiently, e.g. with Matlab's `mldivide` operator, which uses the CHOLMOD package [Dav05].

### 2.4.2 Computing the target curvatures

Once the cone singularities are found, we need to determine their target curvatures. These curvatures are an accumulation of the curvatures of the regular vertices, as dictated by the Green’s functions (2.3), of Section 3.1. The entries of the matrix $P$ are $p_{ij} = w_{ij} / \sum_j w_{ij}$.

Unfortunately, the computation of the inverse of $L=I-S$ required in (2.3) is not feasible for large meshes. Even storing the inverse requires a prohibitive amount of memory. To avoid these problems, we compute $G$ as the solutions to (2.5), where the Laplacian matrix $L$ is sparse, symmetric and semi-positive definite, thus can be efficiently factorized, so that the columns of $G$ (denoted $G_i$) may be computed one by one by back-substitution. Since the number of columns in $G$ is only as the number of cone singularities, this process is quite efficient. The solution, however, might take more time for meshes which have a relatively large boundary, since all the boundary vertices function as cone singularities.

### 2.4.3 Computing the 2D embedding

To generate the final 2D embedding, the mesh is cut such that all the singular vertices are on the boundary of the cut, and the homology generators of the mesh are part of the cut. This is achieved using the Seamster algorithm [SH02], where our singular vertices are used instead of the terminal vertices of Seamster. We used the method of Dey et al. [DLS07] to compute the homology generators. Once a cut is found, we may apply the least squares method of ABF++ [SLMB05] to compute the locations of the 2D vertices given the edge lengths. Since there is no guarantee that the triangle inequality will hold for the computed edge lengths, this might result in complex angles when converting the edge lengths into the angles required for the system of equations. If this is the case, we use the real part of the resulting angles.

In our case, since the edge lengths might have relatively large errors, it is crucial to use the least squares method, and not the more naïve, but also more efficient, original greedy ABF approach where the triangles are laid out incrementally, thus accumulate error.
2.5 Experimental Results

We have run our algorithm, which we call CF (Conformal Factor) on a variety of inputs. For each input we computed the resulting quasi-conformal distortion, the main measure of success, as defined in [KSS06]. The objective is that the quasi-conformal distortion be as close as possible to unity.

Since the edge lengths generated by CF do not induce the exact target curvature, we also computed the average curvature error as the $L_2$ norm: $||K^T - K||/n$, where $K^T$ is the input target curvature, $K$ is the actual curvature induced by the computed edge lengths, and $n$ is the number of vertices. As can be seen in Table 2.1 this error is quite small, on the order of 1e-5 $\pi$. This error is important for another reason - since the final 2D embedding is computed on the cut mesh, edge lengths on both sides of the cut might not be identical, depending on the target curvature error. However, since the curvature error is small, the resulting embedding error is negligible.

<table>
<thead>
<tr>
<th>Model</th>
<th>#Δ</th>
<th>Curvature Error (units of $\pi$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>fandisk</td>
<td>13K</td>
<td>4.6e-6</td>
</tr>
<tr>
<td>rockerarm</td>
<td>20K</td>
<td>1.4e-5</td>
</tr>
<tr>
<td>foot</td>
<td>20K</td>
<td>2.7e-5</td>
</tr>
<tr>
<td>gargoyle</td>
<td>20K</td>
<td>2.9e-5</td>
</tr>
<tr>
<td>elephant</td>
<td>40K</td>
<td>1.6e-5</td>
</tr>
<tr>
<td>horse</td>
<td>40K</td>
<td>1.8e-5</td>
</tr>
<tr>
<td>hand</td>
<td>106K</td>
<td>3.2e-6</td>
</tr>
<tr>
<td>torso</td>
<td>284K</td>
<td>1.4e-6</td>
</tr>
<tr>
<td>isi-horse</td>
<td>358K</td>
<td>8.4e-7</td>
</tr>
</tbody>
</table>

Table 2.1: Curvature error relative to target curvature.
We compared the results of CF with those of the Circle Pattern (CP) method of [KSS06], feeding that method our cone singularities, curvatures and cuts. We also compared to the Linear ABF (LABF) method of [ZLS07], feeding it just the cut (as boundary). All the comparisons were done using software kindly provided by the respective authors, run on the same machine as our software.

Table 2.2 shows the results of this comparison. For each method we list the quasi-conformal distortion, and the computation time in seconds, as measured on a 1.4GHz CPU with 1.5GB RAM. As the other methods do not compute the singularities and the cut, the computation time quoted for all methods includes only the flattening – generating the new edge lengths or angles – given the cone singularities and the cut. Since the final step – generating the 2D layout (coordinates) from the angles or the edge lengths – is common to all three methods, it is not included in the timings. Where data is missing, the respective methods failed to generate the parameterization.

<table>
<thead>
<tr>
<th>Model</th>
<th>#Δ</th>
<th>Quasi-conformal distortion</th>
<th>Runtime (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CP</td>
<td>LABF</td>
</tr>
<tr>
<td>fandisk</td>
<td>13K</td>
<td>1.013</td>
<td>1.007</td>
</tr>
<tr>
<td>rockerarm</td>
<td>20K</td>
<td>1.045</td>
<td>1.027</td>
</tr>
<tr>
<td>foot</td>
<td>20K</td>
<td>1.029</td>
<td>1.016</td>
</tr>
<tr>
<td>gargoyle</td>
<td>20K</td>
<td>1.213</td>
<td>1.032</td>
</tr>
<tr>
<td>elephant</td>
<td>40K</td>
<td>1.028</td>
<td>1.022</td>
</tr>
<tr>
<td>horse</td>
<td>40K</td>
<td>1.035</td>
<td>1.017</td>
</tr>
<tr>
<td>hand</td>
<td>106K</td>
<td>1.009</td>
<td>1.008</td>
</tr>
<tr>
<td>torso</td>
<td>284K</td>
<td>-</td>
<td>1.004</td>
</tr>
<tr>
<td>isi-horse</td>
<td>358K</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2.2: Comparison between parameterization methods

As is evident from Table 2.2, our CF algorithm achieves quasi-conformal distortions comparable to CP at a fraction of the time CP requires. The CP algorithm solves two non-
linear problems, one for the angles and one for the radii. The last is convex and has a unique minimum. Kharevych et al. [KSS06] state that the performance bottleneck of the CP method is the angle optimization step. The distortions generated by CF are only slightly worse than those generated by LABF, which seems to come at the price of a longer runtime. This is to be expected, since LABF solves a sparse linear system of $4n$ equations in $6n$ variables, compared to the sparse linear system of $n$ equations in $n$ variables solved by CF. In addition, CF has the advantage that edge lengths on both sides of the cut are the same, and the curvature of all the vertices except the cone singularities is zero. It is possible that LABF can be modified to accommodate cone singularities, by changing the constraints the method optimizes. If it is indeed possible, the cone singularities generated by CF can be used to drive LABF, and it would be interesting to compare our results to those.

![Figure 2.6: Quasi-conformal distortion color coded over the mesh](image)

Figure 2.6 compares the quasi-conformal distortion per face color-coded over the mesh, produced by the three methods. The color ranges from blue (quasi-conformal distortion = 1) to red (quasi-conformal distortion > 1.5). In all comparisons to CP, we compared to the version without the intrinsic Delaunay triangulation. Figure 2.7 compares the flattening of the foot mesh generated by the three methods. Finally, Figure 2.8 shows the models listed in Table 2.2 texture mapped and color-coded with the quasi-conformal distortion generated by CF.
2.6 Conclusions and Discussion

In this chapter we have presented a new method for conformal parameterization of arbitrary meshes, when given cone singularities and their target curvatures. In addition, we showed how to identify suitable cone singularities for a given 3D mesh. Our method relies only on the solution of sparse linear systems, thus is both simple to implement and very efficient. Despite its simplicity, resulting in runtimes which are much faster than other state-of-the-art methods, it still yields results comparable to those methods.

Our CF method has a few drawbacks. The most important is that it is somewhat sensitive to high-distortion parameterizations, to which the CP method, for example, is more robust. If, for example, the disk-topology hand model in Fig. 2.2 is required to be parameterized without additional cuts (thus without cone singularities), our method will probably not produce a good parameterization. However, as high area distortions are usually undesirable, it is not obvious that this scenario will ever arise in practice. Another drawback is that the method is somewhat sensitive to long and skinny triangles in the original mesh, since the negative cotangent weights generated in this case affect our evaluation of $\phi$ as a piecewise linear function.

Our method is based on manipulating the intrinsic geometry (the curvature distribution and its associated metric) instead of processing the extrinsic geometry (e.g. the vertex coordinates).
directly. The conformal factor $\phi$ is an intrinsic property of the mesh – it only depends on the angles of the mesh and the prescribed target curvatures. Hence, for example, two meshes which are isometric – have the same edge lengths – will have identical conformal factors, for the same prescribed curvature. In the next chapter, we elaborate further on the isometry invariance of the conformal factor, and use this property in order to compute a pose invariant shape descriptor.
Figure 2.8: Texture mapping and color coding of quasi-conformal distortion, for the meshes from Table 2.2.
3 Characterizing Shape Using Conformal Factors

In the previous chapter we presented a novel conformal parameterization method, which is based on computing a special function on the mesh – the conformal factor. As the computation of this function relies only on intrinsic properties of the mesh – such as edge lengths and the discrete Gaussian curvature, it is invariant to isometric transformations. This property is invaluable for a different application – shape matching and retrieval. In this chapter we present a new 3D shape descriptor which is based on the conformal factor defined in Chapter 2. The descriptor is invariant to non-rigid quasi-isometric transformations, such as pose changes of articulated models, and is both compact and efficient to compute. We demonstrate its performance on a database of watertight models, and show it is comparable with state-of-the-art descriptors. Figure 3.1 shows the color coding of our descriptor on a few poses of the armadillo mesh.

![Color-coding of the conformal factor of a few poses of the Armadillo model](image)

3.1 Introduction

As the number of available 3D models grows, the need for efficient shape indexing and retrieval has emerged. 3D models are usually represented in the simplest manner: a set of vertices and faces of a polygonal mesh. However, such a representation cannot be used for a direct comparison of two shapes, since it is not unique: the same shape can be represented by different sets of vertices and faces, with different resolutions. To reliably compare two shapes, search engines usually use a shape signature or descriptor, a numerical representation of the
shape, invariant to the specific tessellation used to represent it as a polygonal mesh. In addition, shape descriptors should be invariant to rigid transformations and should produce "close" descriptors for "similar" shapes. An important invariance property for a shape descriptor is invariance to pose changes. A pose-invariant shape descriptor should be able to identify different poses of the same 3D shape, such as a running, sitting or standing human.

Pose changes are a quasi-isometric transformation of the 3D mesh, in the sense that edge lengths do not change much as a result of the transformation. The Gaussian curvature of a surface is invariant to isometric transformations, so it seems natural that some version of the Gaussian curvature may be used as a pose-invariant shape descriptor. Unfortunately, the computation of the (discrete) Gaussian curvature on a discrete surface is noisy, and the resulting function is not useful in practice as a shape descriptor.

In addition to Gaussian curvature, conformal geometry provides additional intrinsic measures for the surface geometry. The celebrated uniformization theorem (see [Abi81] for a modern proof) shows that any 2-manifold can be conformally mapped to a surface with the same topology having constant Gaussian curvature. Hence, we can compute the conformal factor required to conformally transform the given mesh, to a mesh with constant Gaussian curvature, using the algorithm presented in Chapter 2. This function is both efficient to compute, and, as we will show, considerably less noisy than the Gaussian curvature. Consequently, a histogram of the conformal factor may serve as a robust pose-invariant signature of a 3D shape.

3.1.1 Previous work

The shape matching and retrieval application is heavily researched, and various types of shape descriptors appear in the literature. We will focus here only on recent descriptors which are most relevant to our work. The interested reader is referred to [SMKF04, TV08, DBP06] for comprehensive surveys of different shape descriptors and evaluations of their performance.

As we shall see later, our shape descriptor makes use of the discrete Laplace-Beltrami operator. Several pose-invariant shape signatures based on this operator have been proposed recently. Reuter et al. [RWP05] use the spectrum (i.e. the eigenvalues) of this operator and Rustamov [Rus07] uses its eigenvectors. Xiang et al. [XHGC07] use the histogram of the solution to the volumetric Poisson equation $\nabla^2 U = -1$ as a pose-invariant shape descriptor. This equation also involves the Laplace-Beltrami operator ($\nabla^2$). Superficially similar to what
we shall propose, it is different since it solves an equation on the volume of the shape, whereas we work on the boundary surface alone.

Another spectral-related shape signature was introduced by Jain and Zhang [JZ07], who use the eigenvectors and eigenvalues of the geodesic distances matrix. A pose invariant descriptor based on histograms of surface functions is presented by Gal et al. [GSCO07]. They use two scalar functions on the mesh. The algorithm performs quite well, even on meshes which contain more than one connected component. This shape descriptor is, however, relatively heavy to compute.

Some pose invariant shape descriptors are derived from geodesic distances on the mesh, which are invariant to isometric transformations. Elad and Kimmel [EK03] embed these distances in Euclidean space, where two such embeddings can be compared as rigid objects. In a more recent work, Tung and Schmitt [TS05] use geodesic distances for building a multiresolution Reeb graph. The discriminative power of such descriptors is usually very good, however this comes at the price of a high computational cost for computing the geodesic distances.

Conformal geometry has been applied to shape retrieval in the past. Wang et al. [WWJ*06] propose to do face recognition using the 2D conformal maps of 3D shapes, instead of comparing the 3D models directly. This method is applicable only to shapes which are a topological disc, since general shapes need to be cut (to form a boundary) before they can be embedded in 2D. As the cuts might not be compatible between different 3D shapes, the resulting 2D parameterization might be very different even if the original 3D shapes were similar. Since a 2D parameterization can be generated using the conformal scaling factor and the original 3D edge lengths, as we have shown in the previous chapter, using the conformal factor as a signature can be viewed as a generalization of Wang et al.’s method to general 3D meshes. Gu et al. [GV04] and Wang et al. [WCT05] combine the conformal factor with the mean curvature for shape matching in medical applications. However, since the mean curvature is an extrinsic measure dependent on the embedding, it is not invariant to isometric transformations. Recently, Jin et al. [JLYG07] suggested the use of the conformal class of a shape as a signature. Although the signature is invariant to isometries, it is not discriminative enough – for example, all closed genus zero surfaces belong to the same conformal class.
3.2 The Shape Descriptor

3.2.1 The Discrete Conformal Factor

The celebrated uniformization theorem [Abi81] states that any 2-manifold surface can be conformally mapped to a surface with the same topology having constant Gaussian curvature. Such a mapping can be achieved by defining a positive scalar function on the surface, and locally changing the surface metric (which can be thought of as scaling infinitesimal patches of the surface) using this function. The scaling function $\phi$ (also known as the conformal factor) that achieves this depends only on the Gaussian curvature of the surface, hence is invariant to isometric transformations.

Our conformal parameterization algorithm, presented in the previous chapter, can compute the edge lengths of a conformal mapping using any target curvature, not necessarily zero. We used this property previously in order to prescribe curvature at the cone singularities to be non-zero. However, we can now apply the same theory, in order to compute the conformal factor required to conformally map the input mesh, to a mesh with constant Gaussian curvature. In this case, the prescribed target curvature should be uniform on all the mesh, so we set:

$$ k^T_v = \left( \sum_{i \in F} \kappa_i \right) \frac{1}{3 \sum_{f \in F} \text{area}(f)} \sum_{f \in F} \frac{\text{area}(f)}{\text{area}(f)} $$

This formula assigns to each vertex a portion of the total curvature. Although the target curvature is supposedly uniform, it is not uniformly distributed among the discrete vertices. Was this the case, the descriptor would not be tessellation invariant. Hence, the portion of the curvature assigned to a vertex is determined by the "influence area" of the vertex – a third of the area of the faces near the vertex, divided by the total surface area of the mesh. This also guarantees that the sum of the target curvatures on all vertices is identical to the sum of the original vertex curvatures.

For a connected mesh, the Laplacian has co-rank 1, so the solution $\phi$ is defined only up to an additive constant. This is resolved by requiring $\phi$ to have zero mean. Since the Laplacian is sparse and symmetric, Equation (2.1) may be solved for $\phi$ very efficiently, e.g. with Matlab's \textit{mldivide} operator, which uses the CHOLMOD package [Dav05]
Figure 3.2 shows the color coding of the conformal factor (normalized to the range [0,1]) computed for two dancer models, and two hand models from the AIM@SHAPE shape repository. As the figure demonstrates, the conformal factor identifies well the features of the shape - such as the fingers of the hand and the feet of the dancer. This is because the conformal factor represents how much local "work" is involved in globally transforming the model into a sphere (in the case of a genus zero model). Long extrusions such as the fingers and the feet require more "work", which is expressed as a larger conformal factor.

Figure 3.2: Color-coding of the conformal factor of two hand models and two dancer models from the AIM@SHAPE shape repository (normalized to the range [0,1]).

3.2.2 The Signature

Given the conformal factor we can now describe how to generate the signature of a given mesh, and how to compare two signatures.

As is commonly done when defining a shape descriptor based on a real function on the mesh [GSCO07, XHGC07, Rus07], we use the histogram of the conformal factor as our shape signature. Since the conformal factors are invariant to rigid transformations, so are their histograms. To make the signature invariant also to the tessellation of the mesh, we sample \( \phi \) at \( 5|F| \) random points on the surface of the mesh. This is achieved by selecting a random
triangle with probability proportional to its area, and then randomly generating a point uniformly distributed in the triangle interior. The value of \( \phi \) at this point is defined to be the linear interpolation of \( \phi \) at the vertices of the triangle. To this sample set we add the original mesh vertices, and compute the histogram of \( \phi \) at all these points, which is now almost independent of the mesh' tessellation. Finally, we compute the histogram with 200 uniform bins spanning the range [-99,100], resulting in a signature with 200 values. See Figure 3.3 and Figure 3.4 for some examples.

To compare two signatures, \( S_1 \) and \( S_2 \) we use the L_1 norm:

\[
d(S_1, S_2) = \sum_i |S_1^i - S_2^i|
\]

### 3.3 Evaluation of the Shape Descriptor

In this section we evaluate the performance of our shape descriptor, from both a theoretical and experimental point of view. First we explain why it is indeed pose-invariant, and then we show some experimental results demonstrating its robustness to noise. Next, we evaluate its performance in retrieval from a shape database of a few hundred watertight models. Finally, we compare it to other state-of-the-art shape descriptors, both pose-invariant and pose-dependent.

#### 3.3.1 Pose Invariance

Gaussian curvature is a good starting point for a pose-invariant shape descriptor. Being an intrinsic measure, dependent only on the metric (i.e. the edge lengths) of the mesh, and not its embedding, it is invariant under isometric transformations - transformations which preserve edge lengths. Pose changes of articulated shapes are one example of such transformations, but also non-articulated shapes exhibit such transformations. For example, unfolding a developable surface to a plane is an isometric transformation, and it indeed preserves the (vanishing) Gaussian curvature during the transformation.

Unfortunately, standard discrete Gaussian curvature measures are too noisy to allow comparison using a simple histogram descriptor, since they are computed locally for each vertex. The conformal factor, on the other hand, despite being derived directly from the Gaussian curvature, depends also on the metric of the mesh (through the weights of the Laplacian matrix). Since the conformal factor is the result of a global computation on the entire mesh, it is much smoother than the discrete Gaussian curvature, and hence can be easily
used in a simple histogram descriptor. Since the Gaussian curvature is invariant under isometric transformations, so is the conformal factor.

Figure 3.2 shows the conformal factors and their histograms, computed as described in the previous section, for three models in different poses. Evidently the same model in different poses results in very similar histograms, and different models have quite different histograms – both in shape, and in the range of the values the conformal factor achieves.

**Figure 3.3:** Color-coding of conformal factors and histograms of the conformal factors, for three model classes in various poses (the values are normalized to the range [0,1] for the color-coding).
Figure 3.4 shows the discriminative power of the conformal factor. The conformal factor tends to be positive on long extrusions - the longer the extrusion, the larger the conformal factor. This means that similar objects, which are not isometric transformations of each other – for example two springs of different length – will have similar-shaped histograms, but with different ranges of values. Hence, two springs of different length will be considered different. Normalizing the conformal factor to the range [0,1] will result in shapes with the same number of extrusions, but different extrusion lengths to be considered similar. This can be useful in certain cases, for example, for the springs from the watertight shapes benchmark [VtH07,GBP07], as can be seen in Figure 3.5. Unfortunately, for most models, normalizing the conformal factor may have undesirable effects. For example, chairs and humans might be considered similar, as they have more or less the same number of extrusions. Empirically, we have noticed that the un-normalized conformal factor performs better on most models.

**Figure 3.4:** Color coding of conformal factors, and the histograms of the conformal factors of three springs. The two springs of the same length have similar histograms. The shorter spring has a histogram with the same shape, but different range of values.
Figure 3.5: Shape retrieval results from the watertight shapes benchmark [VtH07,GBP07] using the conformal factor (upper two rows) and the conformal factor normalized to [0,1] (lower two rows). In both cases the first model on the upper left is the query model. The database contains exactly 20 springs.

Figure 3.6: Color-coded conformal factors (values normalized to the range [0.1]), and histograms of the conformal factors, for a spring model, and a noisy version of it.

3.3.2 Robustness to Noise

An important property of any shape descriptor is the ability to match similar shapes even in the presence of noise. Since our descriptor is based on Gaussian curvature, which is notorious for being sensitive to noisy geometry, verifying that our descriptor is robust is a major concern.
We contaminated the geometry of one of the springs from the watertight benchmark with Gaussian noise with $\sigma = 0.01$, which is about 20% of the mean edge length of the model. We then computed the conformal factor and its histogram on the noisy spring, and compared it to the original. Figure 3.6 shows the results of this comparison – both the histogram and the color coding show that the difference between the signature of the original model and the noisy model is minor.

### 3.3.3 Sensitivity to Topology

A drawback of our descriptor is that it is sensitive to small topological differences. Since the descriptor is based on the conformal mapping to a surface with uniform curvature of the same genus, mapping between shapes of different genera is possible, but not necessarily successful. For example, the watertight shapes benchmark [VtH07,GBP07] contains two versions of the teapot, one of genus 0, and the other genus 1.

![Figure 3.7: The conformal factor for two similar models with a small topological difference.](image)

As Figure 3.7 shows, the distributions of the conformal factor of these models are quite different (as are their histograms). In this case, a simple visual-based descriptor (such as the light field descriptor [COTS03] for example) will do a better job. Of course, those two surfaces are not isometric transformations of each other.

### 3.3.4 Curvature Error

The discrete conformal factor computed by our linear method is only an approximation of the true continuous conformal factor, and is accurate only for small curvature changes. Therefore, it is interesting to check the distribution of the error in the conformal factor over the mesh, and how it affects our shape descriptor. To test this, we computed the curvature error, as $|K^T - K^\theta|$, where $K^T$ is the target curvature we wanted to obtain, as defined in section 2.2, and $K^\theta$ is the obtained curvature, computed after scaling the original edge lengths with the computed conformal factor. Figure 3.8 shows the distribution of this color-coded error over one of the
meshes from the watertight shapes benchmark [VtH07,GBP07]. The average curvature error on all the mesh, defined as \(|K^T - K^R|/|V|\) is 1.8e-5\(\pi\). As the figure shows, the error is close to zero on most of the mesh, and is localized in areas where the difference between the original and target curvatures were relatively high. Hence, in practice, the error in the conformal factor resulting from the discretization will not have much effect on our shape descriptor.

Figure 3.8: The curvature error obtained by the linear conformal factor, for a uniform target curvature.

3.3.5 The Watertight Shapes Benchmark

We tested our conformal factor (CF) descriptor on a large database of watertight meshes [VtH07,GBP07], and compared its performance with that of two state-of-the-art descriptors which are not pose-invariant: the spherical harmonics (SH) descriptor [KFR03] and light field (LF) descriptor [COTS03]. We also compared to the global point signature (GPS) - a more recent pose-invariant descriptor [Rus07], and to the augmented multiresolution Reeb graph (aMRG) – a pose invariant descriptor [TS05] which won the SHREC 2007 competition on the database of watertight meshes [VtH07,GBP07].

The implementations of SH and LF were taken from the respective Web sites of the authors. The comparison with GPS was based on the simulated search engine at the author's Web site. The comparison with aMRG was based on the results published in [GBP07].

The database contains 20 classes of objects, each containing 20 models. Some classes, e.g. the "ants" class, contain models which are indeed a result of quasi-isometric transformations. In these cases, our CF descriptor performs better than the non pose-invariant descriptors SH and
LF, and almost as good as the pose-invariant GPS. On almost all classes however, the aMRG descriptor outperforms our descriptor. This superior retrieval performance comes with a price, as the aMRG descriptor is graph-based, and relatively computationally heavy both to compute and to compare. For example, the authors state [GBP07] that the computation time for the aMRG for a model containing 15,000 vertices is about 20 seconds on a 1.6GHz laptop. In contrast, computing our CF descriptor for the same model takes about 0.9 seconds on a 1.4GHz laptop. Hence, our descriptor can be efficiently computed even for relatively large meshes. Comparing two signatures is also much more efficient using our descriptor, as it is vector based, and requires only the computation of a simple L1 sum.

![Figure 3.9: Precision/recall graphs of various shape descriptors applied to the ants class in the watertight shapes benchmark [VtH07,GBP07].](image)

To quantitatively compare the performance of our descriptor with other shape signatures, we used a standard measure: the precision/recall graph [SMG83]. This describes the relationship between precision and recall in a ranked list of matches. For each query model in the class and any number $k$ of top matches, “recall” (the horizontal axis) represents the ratio of models in the class returned within the top $k$ matches, while “precision” (the vertical axis) indicates the ratio of the top $k$ matches that are members of the class. A perfect retrieval result produces a horizontal curve (at precision = 1.0), indicating that all the models within the query object’s class are returned as the top ranked matches. In general, curves that stay close to the precision 1.0 mark represent superior retrieval results. Different points on the graph represent different values of $k$. 

46
Figure 3.9 shows the precision/recall graph for the ants class of CF versus aMRG, SH and LF. Since the comparison with GPS was based on the results from the simulated search engine, this particular descriptor is not shown in the precision/recall graph. As discussed before, for the ants class, our descriptor performs better than the non pose invariant descriptors, but worse than the aMRG descriptor.

Figure 3.10 shows the query result for a specific ant using CF, GPS, LF and SH. The aMRG descriptor was not compared using query results. As each class contains exactly 20 models, showing the first 20 results of the query is quite informative.

The GPS descriptor manages to retrieve all the ants in the class, achieving a perfect precision/recall ratio of 1.0. Our CF retrieval is very similar (all ants except one were retrieved), yet the CF descriptor is much simpler to compute, requiring only the solution of a sparse symmetric linear system, instead of an eigenvector problem.

For most of the other classes, the results are not that clear cut. For example, in the armadillo class, our CF descriptor finds most of the deformations of the armadillo, which do not include missing parts. This is natural, since severing parts of a surface is not an isometric deformation. The aMRG descriptor, however, achieves the best performance on almost all the classes. Figure 3.11 and Figure 3.12 show the precision/recall graph and the retrieval results for a representative model using the four descriptors, for the armadillos and heads classes.
Global point signature (GPS)

Spherical harmonics (SH)

Light field (LF)

Conformal factor (CF)

Figure 3.10: Comparison of the retrieval of an ant in the watertight shapes benchmark [VtH07,GBP07]. The query model is in the upper left corner, followed by the retrieved shapes in decreasing order of similarity.
Figure 3.11: Precision/recall graph, and the retrieval of a single model in the watertight shapes benchmark [VtH07,GBP07] for the armadillo class.
Figure 3.12: Precision/recall graph, and the retrieval of a single model in the watertight shapes benchmark [VtH07,GBP07] for the heads class.
Figure 3.13: Precision/recall graphs for some more classes from the watertight shapes benchmark [VtH07,GBP07].

Figure 3.13 shows some more precision/recall graphs for other classes. Figure 3.13 shows the results of sample queries, one query per each model class. Correct retrieval results (results from the same class as the query model) are colored green, and incorrect results are colored orange.

3.4 Conclusions and Discussion

In this chapter we have presented a new shape descriptor, based on conformal geometry, which is invariant to isometric transformations. The descriptor is simple and very easy to compute and compare. It is based on solving a sparse linear set of equations, and does not require complex computations, such as computation of geodesic distances, solution of eigenvalue problems, or computing a Reeb graph. It performs comparably to state-of-the-art shape descriptors, both pose-dependent and pose-invariant.

Our descriptor has a few drawbacks. The most important is probably that the descriptor is applicable only to a manifold mesh. Otherwise, discrete geometric notions – such as the Laplace-Beltrami operator and the discrete curvature – are not well defined. As the majority
of models available on the web are not manifolds, it is important to make this descriptor robust to non-manifold meshes. Another robustness issue is the problem of instability under local genus changes. As we conformally map between surfaces of the same genus, similar meshes of different genus might have a different signature.

A natural extension would be to use the conformal factor for full and partial shape matching. As the color coded meshes show, similar parts of shapes – for example the fingers of the armadillo model – match on different deformations of the mesh. We believe that the conformal factor, together with other intrinsic measures, may be a useful tool for such applications.

So far we explored the application of conformal maps to planar parameterization and shape retrieval. In the next chapter we depart from planar conformal maps, in favor of the more general harmonic maps, which allow us to define a novel 3D space deformation method.
Figure 3.14: Retrieval results for some query shapes, one from each class of the watertight shapes benchmark. Each row is the result of a single query. The leftmost shape (highlighted in purple) is the query image. Shapes highlighted in green are shapes from the correct class (the same as the class of the query shape). Shapes highlighted in orange are shapes from wrong classes.
4 Variational Harmonic Maps for Space Deformation

Conformal maps have proved to be a very useful tool for geometry processing applications. A natural extension would be to use conformal maps for deformation. This idea was exploited successfully in [WBCG09] for planar shape deformation. Unfortunately, in three dimensions conformal maps are somewhat limited, and cannot be used directly for deformation. Instead, we use the more general family of harmonic maps. In this chapter we show how harmonic maps can be leveraged to define an efficient, high quality and user friendly deformation method. Figure 4.1 shows an example of a source shape, and its deformation using our algorithm.

![Figure 4.1: The Beast model enclosed in its cage (left) and its deformation using a variational harmonic map (right)](image)

4.1 Introduction

Space deformation methods deform the ambient space in which a shape is embedded, instead of explicitly deforming the shape itself. Such methods have become popular in recent years [HSL*06, LKCOL07, JMD*07, LLCO08, SSP07, BPWG07], for several reasons. First, they are more general than explicit deformation – space deformation can be applied to any shape representation, whether it is a polygonal mesh, a point cloud or volumetric data.
Second, by deforming the ambient space, the computational complexity of the deformation is decoupled from the complexity of the shape, hence even extremely complex shapes can be deformed at interactive rates. Some space deformation methods [LKCOL07, JMD*07, LLCO08] are "cage-based". In these methods, a given "source cage" is manipulated by the user to create a "target cage". Then, based on the source and target cages, a mapping of the source cage is defined. If the mapping function has a closed-form expression, the deformation method becomes accurate and efficient. On the other hand, manipulating a cage is a tedious and time-consuming task. A more user-friendly and natural deformation method is direct manipulation – the user positions a small number of "control points" inside the domain, and manipulates them instead of the cage. Such methods [HSL*06, SSP07, BPWG07] define the space deformation on a domain which is coarser than the input shape, and solve an optimization problem to find the parameters of the deformation, given the user's constraints. As this optimization problem is generally non-linear, the robustness and efficiency of these algorithms depend critically on the formulation of the deformation, and the optimization method used.

We propose to use harmonic maps of the source region as the underlying deformation model. Since harmonic functions are smooth and regular, they are used for a wide range of applications, from parameterization [FH05] and remeshing [DKG05] to space deformations [JMD*07, LLCO08]. We generate harmonic maps on the domain as a linear combination of harmonic basis functions. In the special case that the domain is a polyhedron, these basis functions, and their first and second derivatives, will have closed-form expressions, as will the harmonic maps. Using these expressions, we allow the user to place position and orientation constraints at arbitrary locations inside the domain and define an energy functional which depends also on these constraints. By defining additional "rigidity lines" in a semi-automatic way, the resulting deformation is a natural "As-Rigid-As-Possible" deformation of the shape, respecting the specified constraints. It is worth noting that our harmonic basis functions are a variant of the "Green coordinates" of Lipman et al. [LLCO08] (and Weber et al. [WBCG09]), however, we give simpler expressions for them, and also provide their first and second derivatives.
4.1.1 Contribution

Our main contribution is a robust and very efficient space deformation method, which provides some advantages over existing methods. First, the user manipulates a set of position and orientation constraints, instead of directly manipulating the "source cage", hence our method is more intuitive and easy to control. Second, we have a closed-form expression - a linear combination of basis functions - for the deformation of a continuous domain, thus do not require a voxelization of the input domain, as some other methods do. And finally, since we have closed-form expressions also for the gradients of the deformation, our optimization procedure may be based on an alternating least-squares "local/global" algorithm, which, until now, was applicable only in discrete mesh-based settings. This optimization method is extremely efficient, as its computational complexity is dominated by matrix-vector multiplications using pre-computed matrices, thus may also be easily implemented on the GPU. In addition, it is quite simple to implement, and guaranteed to converge.

4.1.2 Previous Work

Shape deformation is one of the most active research subjects in computer graphics, and a thorough review of all the recent work is outside our scope. We shall thus concentrate on the space deformation methods most relevant to our work. In general, these methods can be classified into two major groups – "cage-based" deformation, and direct manipulation deformation.

In "cage-based" deformation, the user specifies the boundary of a relevant region of space – the source "cage" – which contains the input shape. The cage is typically a piecewise-linear closed surface. The user then manipulates the vertices of this cage to generate a target cage and the deformation is defined by the relationship between these two cages. Cage-based methods are closely related to barycentric coordinates, as typically the deformation is defined as a linear combination of the vertices of the target cage with a set of barycentric coordinate functions defined on the input cage. Since these barycentric coordinate functions depend only on the source cage, they can be pre-computed making for a very efficient method, as the deformation then requires only a matrix-vector multiplication. One of the first such methods [HSL*06] used mean-value coordinates [FKR05, JSW05] as the coordinate functions. Unfortunately, mean-value coordinates are not guaranteed to be positive inside the domain unless it is convex. This causes severe artifacts in the resulting deformation. Later methods
[JMD*07] suggested using harmonic coordinates instead, as these are guaranteed to be positive inside the domain. However, harmonic coordinates are the solution of a Dirichlet problem on the boundary of the domain, and they do not have a closed form expression. Thus, computing these coordinates is not easy. Recently, Lipman et al. [LLCO08] showed how to define two sets of coordinate functions – Green coordinates, which have closed-form expressions, and result in detail-preserving mappings. Later, Weber et al. [WBCG09] showed that these coordinates in two-dimensions are a special case of complex-valued barycentric coordinates, and may be derived from the celebrated Cauchy integral theorem. They called them Cauchy-Green coordinates. All the cage-based methods have a common disadvantage – detailed deformations are possible only with relatively complex cages, and such cages – even with a few hundred faces – are extremely hard to manipulate in order to generate a satisfying result. To overcome this problem, Weber et al. [WBCG09] proposed something similar in spirit to our method: use the complex Cauchy-Green coordinates as conformal basis functions, and find the new cage location by solving an optimization problem derived from position constraints supplied by the user. Although their method is quite effective, the complex conformal formulation applies only to planar deformation. Our method can be considered as a generalization of [WBCG09] to three dimensions, and $\mathbb{R}^d$ in general. But there is a major difference between the two methods. We use harmonic mappings as basis functions instead of conformal functions, since complex holomorphic functions (which in two dimensions generate conformal maps) do not have a simple generalization to three dimensions. As a result, in order to achieve detail-preservation, we need to solve a non-linear minimization problem, whereas in [WBCG09] the optimization required the solution of a linear system.

It is worth noting that all previous methods [JMD*07, LLCO08, WBCG09] use harmonic maps of some sort as their underlying deformation function. Harmonic coordinates use independent harmonic functions for each coordinate, thus are able to enforce an exact interpolation of the target cage. However, this is both hard to compute, and causes serious shearing effects. Cauchy-Green coordinates in two dimensions [LLCO08, WBCG09] enforce conformal maps – harmonic maps, whose two components have orthogonal gradients with equal norm. Green coordinates in three dimensions use the vertices of the target cage and its normals as the coefficients of a linear combination of the harmonic basis functions. From this point of view, our framework is a generalization of all those coordinates – we seek a harmonic
map, but instead of predefining the relationship between its components, the relationship is derived implicitly by minimizing an energy functional.

Two other recent space-deformation methods which solve a non-linear optimization problem given positional constraints are those of [SSP07] and [BPWG07]. In [SSP07], the deformation is defined using a deformation graph, which is automatically computed from the input shape. An affine transformation is associated with each node in the deformation graph, which describes the transformation this node undergoes. These transformations are the variables of an energy function – which forces them to be rigid and have a smooth behavior. Minimizing this energy, combined with the position constraints imposed by the user, generates the deformation parameters of the deformation graph. The deformation of a point in the ambient space is then computed from the transformations of nodes in the deformation graph, which are close in Euclidean distance to this point. There are two disadvantages of this method compared to ours – first, the deformation graph is not a cage, in the sense that the deformation function is computed based on Euclidean distances. This causes artifacts when deforming a shape which has pieces which are close to each other in Euclidean distance, but far apart in geodesic distance, for example, fingers of a hand. In addition, the smoothness of the deformation is enforced discretely, by requiring neighboring faces of the deformation graph to have similar transformations. In our setting, we have a closed-form expression for the second derivatives of the deformation, and we require these to vanish on the boundary of the domain, hence the regularization term of the energy is more robust. A similar method is that of [BPWG07], where the deformation is defined on a voxelization of the input region. Here the deformation is also computed by solving a non-linear optimization problem using a multi-grid framework. This method suffers from some aliasing effects due to the discretization, and in addition its implementation is somewhat involved. Comparisons of the results of our method with the methods of Sumner et al. and Botsch et al. will be presented in the Section 4.4. Other direct surface manipulation techniques exist, such as such as those of Sorkine et al. [SCOL*04], Lipman et al. [LSLCO05] and Sorkine et al. [SA07], to mention only a few. However, as these methods work directly on the surface of a manifold mesh, they are somewhat limited, and cannot be applied to other shape representations, such as polygon soups or point clouds.

Our formulation of the deformation mapping is based on Green's third identity, which relates the values of a harmonic function on the boundary of a region to its values inside the
region. This is closely related both to the Green coordinates defined by Lipman et al. [LLCO08], and to a common method for solving boundary-value problems known as the “Boundary Element Method" or BEM [Kyt95]. BEM has been used, for example, by Martin et al. [MKB*08] to discretize harmonic basis functions for polyhedral finite elements. Despite the common mathematical machinery, our approach is somewhat different from both these methods. In the BEM framework, one seeks a harmonic function on the domain having some given boundary values, whereas we seek a harmonic map which minimizes a given functional. In the Green coordinates setting, the boundary mapping functions are set to be the "target cage" and its normal vectors, whereas in our setting the boundary mapping functions are variables in an optimization problem.

4.1.3 Method Overview

Before diving into the underlying mathematics, we present a brief overview of our deformation method. The input is a polyhedral cage enclosing some region of interest, and a set of position and orientation constraints on a number of points within the cage. The output is a harmonic deformation mapping \( f \), which maps every point in the input cage to some point in \( \mathbb{R}^3 \).

As will be explained in the next section, the deformation mapping is uniquely defined by two functions, \( a \) and \( b \), defined on the vertices and faces of the cage, respectively. In order to find \( a \) and \( b \), we pose an optimization problem where the discrete values of \( a \) and \( b \) are the variables. The goal of the optimization problem is to minimize an energy functional which requires detail preservation and smoothness, while enforcing the user's constraints.

In the discrete setting, our method is somewhat similar to solving for the locations of the vertices of the target cage, using the Green coordinates [LLCO08] deformation method. Thus, the function \( a \) is analogous to the vertex locations of the target cage, and \( b \) is analogous to the normals to the faces. However, there is an important difference – in our setup, the functions \( a \) and \( b \) are independent, whereas in the Green Coordinates setup, \( b \) (the normals to the faces of the target cage) are uniquely defined by \( a \) (the vertices of the target cage). Hence, we have more degrees of freedom, and a larger space of possible deformations.
4.2 Variational Harmonic Maps

Given the input domain – the region of space in which our shape lies, we consider all possible harmonic mappings of this domain. Within this large space of possible deformations, we will choose the harmonic map which both satisfies the user's constraints, and preserves detail as much as possible. We begin by describing our deformation mapping, first for a general domain, and then for a domain with a piecewise-linear (polyhedral) boundary. Once the deformation mapping is established, we will discuss the energy functional.

4.2.1 Harmonic Maps from Boundary Functions

Let $\Omega$ be an open region of $\mathbb{R}^d$ with a smooth boundary $S$, and let $f$ be a continuous function from $\Omega$ to $\mathbb{R}^d$. For example, for $d = 3$, $f = (u(x,y,z), v(x,y,z), w(x,y,z))$. We say that $f$ is a harmonic map if each of its $d$ components are harmonic functions from $\Omega$ to $\mathbb{R}$. Specifically, in three dimensions, $f$ is a harmonic map if:

$$\forall p = (x,y,z) \in \Omega, \quad \nabla^2 u(p) = 0, \nabla^2 v(p) = 0, \nabla^2 w(p) = 0$$

where $\nabla^2$ is the Laplacian operator:

$$\nabla^2 u(x,y,z) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}$$

Since the Laplacian is a linear operator, harmonic mappings form a linear subspace of functions from $\mathbb{R}^d$ to $\mathbb{R}^d$. We would like to select a mapping from this linear space, which both satisfies the user's constraints and is detail-preserving. However, using the current formulation, it is not obvious how to find such a mapping. Fortunately, all harmonic maps on $\Omega$ can be generated by integrating two smooth maps defined on $S = \partial \Omega$, (the boundary of $\Omega$) with two special functions. This is formalized in the following theorem.

**Theorem:** The mapping $f : \Omega \rightarrow \mathbb{R}^d$ is a harmonic mapping if and only if there exist two $C^2$ mappings $a$ and $b$: $a, b : S \rightarrow \mathbb{R}^d$ such that

$$f(p) = \int_{q \in S} a(q)(\nabla G(q,p) \cdot \hat{n}(q))dA - \int_{q \in S} b(q)G(q,p)dA \quad (4.1)$$

where $G(p,q)$ is the fundamental solution of the Laplace equation in $\mathbb{R}^d$ and $\hat{n}(q)$ is the unit normal direction to the surface $S = \partial \Omega$ at the point $q$. 

**Proof:** Let us concentrate on the case \( d = 3 \). Then \( G(p,q) = 1/(4\pi|p-q|) \). It is straightforward to see that \( f \) as defined in (4.1) is a harmonic mapping by taking the derivative relative to \( p \) under the integral sign. Let us consider the second integral – the mapping \( b \) is defined on \( S \), hence does not depend on \( p \). So:

\[
\nabla^2 \left( b(q)G(q,p) \right) = b(q)\nabla^2 G(q,p)
\]

\( G \) is a solution to the Laplace equation, hence harmonic: \( \nabla^2 \left( b(q)G(q,p) \right) = 0 \). Similarly, for the first integral – \( a \) is defined on the boundary and does not depend on \( p \). So, we have:

\[
\nabla^2 \left( a(q)(\nabla G(q,p) \cdot \hat{n}(q)) \right) = a(q)\nabla^2 \left( \nabla G(q,p) \cdot \hat{n}(q) \right)
= a(q)\nabla^2 \left( G_x n_x + G_y n_y + G_z n_z \right)
\]

where \( G_x = \partial G/\partial x \), and so on, and \( \hat{n}(q) = (n_x, n_y, n_z) \). Since \( G \) is harmonic, all its partial derivatives \( G_x \), \( G_y \) and \( G_z \) are harmonic functions. In addition, the normal to the surface does not depend on \( p \), so its dot product with \( \nabla G \) is just a linear combination of harmonic functions, which is again a harmonic function.

The opposite direction is due to Green's third identity, which guarantees that any harmonic scalar function \( u \) satisfies:

\[
u(p) = \oint_{q \in S} u(q)(\nabla G(q,p) \cdot \hat{n}(q))dA - \oint_{q \in S} (\nabla u(q) \cdot \hat{n}(q))G(q,p)dA \tag{4.2}
\]

for any point \( p \in \Omega \). This is true for all the \( d \) components of \( f \). Hence, taking

\[
a(q) = f(q) \quad \quad b(q) = J_f(q) \cdot \hat{n}(q)
\]

where \( J_f(q) \) is the Jacobian of \( f \) at \( q \), completes the proof. ∗

We will now use Eq. (4.1) to define our deformation mapping.

**The continuous deformation mapping.**

The fundamental solution \( G(q,p) \) to the Laplace equation has a closed-form expression for any dimension \( d \). We define the following two scalar kernel functions: \( \hat{\psi}, \hat{\phi} : (S \times \Omega) \to R \)

\[
\hat{\psi}(q,p) = G(q,p) \quad \quad \hat{\phi}(q,p) = \nabla G(q,p) \cdot \hat{n}(q)
\]
Given two smooth mappings \(a, b : S \rightarrow \mathbb{R}^d\), we define the deformation mapping \(f: \Omega \rightarrow \mathbb{R}^d\) to be

\[
f_{a,b}(p) = \int_{q \in S} a(q)\hat{\phi}(q, p) dA - \int_{q \in S} b(q)\hat{\psi}(q, p) dA
\]

In this way we are able to represent \(f_{a,b}\) at any point of the domain as boundary integrals of the kernel functions with \(a\) and \(b\). By the Theorem, the deformation mapping spans the linear space of all harmonic mappings on \(\Omega\), through the mappings \(a\) and \(b\) on \(S\). Since \(a\) and \(b\) do not depend on \(p\), we can also obtain expressions for the partial derivatives of the mapping. For example:

\[
\frac{\partial f_{a,b}(p)}{\partial x} = \int_{q \in S} a(q)\frac{\partial \hat{\phi}(q, p)}{\partial x} dA - \int_{q \in S} b(q)\frac{\partial \hat{\psi}(q, p)}{\partial x} dA
\]

Similar expressions may be derived for any partial or higher order derivatives of \(f\). Note that both the deformation mapping, and its derivatives, are linear in \(a\) and \(b\). Using the deformation mapping and its derivatives, we will later define an energy functional \(E(f_{a,b})\) and the final deformation of a point \(p \in \Omega\) will be \(f'_{a',b'}(p)\) where

\[
(a', b') = \text{arg min}(E(f_{a,b}))
\]

Before defining the energy functional, we first show how the deformation mapping can be simplified in the special case that the source region is polyhedral.

**The discrete deformation mapping.**

In the most general setup, the domain \(\Omega\), and the boundary mappings \(a\) and \(b\), can be arbitrary. However, deformation applications usually bound \(\Omega\) with a piecewise linear surface – meaning the deformed shape is contained in a \(d\)-dimensional polyhedron. In addition, we would like to restrict \(a\) and \(b\) to be of specific types, so that we can find closed expressions for the integrals of \(\hat{\phi}\) and \(\hat{\psi}\) on the faces of the cage, and for their derivatives.

Note, that if \(a\) and \(b\) are restricted to a specific family of functions, one direction of the Theorem is not true anymore, and we cannot generate all harmonic mappings on \(\Omega\) using (4.1) anymore. When choosing the families that \(a\) and \(b\) belong to, we have to make sure the identity mapping \(f(p) = p\), and, more generally, any affine mapping \(f(p) = Ap + T\) (where \(A\) is an \(d \times d\) matrix, and \(T\) is a vector), are still obtainable by (4.1). In this case we should use:

\[
a(q) = Aq + T \quad b(q) = A \cdot \hat{n}(q)
\]
Hence, for a piecewise-linear cage, $a$ can be restricted to be piecewise-linear on $S$. Since a piecewise-linear surface has piecewise-constant normals, $b$ can be restricted to be piecewise-constant. These are the simplest families for $a$ and $b$. Of course, one could use higher degree polynomials, but then the expressions for the integrals of $\hat{\phi}$ and $\hat{\psi}$ will be more complicated.

We will discuss now specifically the three-dimensional case for a region $\Omega$ bounded by a triangle mesh $S = (V, F)$, $V$ are the vertices of $S$ and $F$ are its faces. As mentioned, $a$ and $b$ are not arbitrary mappings anymore – $a$ is the piecewise-linear map on $S$ defined by values at the vertices $\{a_v \in \mathbb{R}^3 | v \in V\}$, and $b$ is the piecewise constant map defined by values at the faces $\{b_t \in \mathbb{R}^3 | t \in F\}$.

In this case, our deformation map becomes:

$$f_{a,b}(p) = \sum_{t \in F} \int_{Q_{qst}} a(q) \hat{\phi}(q, p) dA - \sum_{t \in F} \int_{Q_{qst}} b_t \hat{\psi}(q, p) dA$$

Here, $a(q)$ is the piecewise linear interpolation of the values $a_i, a_j, a_k$ on the vertices of the triangle $t = (i, j, k) \in F$.

**Figure 4.2:** Deformation of a range-scanned model (polygon soup) using our harmonic mapping. (Right) Source model enclosed in its cage. (Left) Deformed model.

Precisely this equation was considered by Lipman et al [LLCO08], and the analytic solutions of the integrals were given. However, we prefer to use different expressions, which were developed in the context of boundary element methods by [Ura00]. These expressions have a somewhat geometric interpretation, and their derivatives are easier to compute.
analytic solutions of the integrals allow us to express $f$ using two sets of scalar functions
$\{ \phi_v(p) : \Omega \rightarrow R \mid v \in V \}$, and $\{ \psi_t(p) : \Omega \rightarrow R \mid t \in T \}$. Using these functions, the deformation map can be expressed as:

$$f_{a,b}(p) = \sum_{v \in F} a_v \phi_v(p) + \sum_{t \in T} b_t \psi_t(p)$$

The expressions for $\phi_v$ and $\psi_t$, their gradient vectors and Hessian matrices are given in Appendix B. Figure 4.1 and Figure 4.2 show examples of such deformations, given specific mappings $a$ and $b$ on $S$.

Given a point $p$, we can write its deformation mapping in matrix notation as follows:

$$\left( f_{a,b}(p) \right)_{1 \times 3} = \begin{pmatrix} \phi_{1:n}^T & \psi_{1:m}^T \end{pmatrix} \begin{pmatrix} a_{nx3} \\ b_{nx3} \end{pmatrix}$$

(4.3)

where $n$ is the number of vertices, $m$ is the number of faces, $\phi$ is the row vector whose entries are $\phi_i(p)$, $\psi$ is the row vector whose entries are $\psi_t(p)$, $a$ is the matrix whose $i$-th row is $a_i$, and similarly for $b$.

The transpose of the Jacobian of the deformation at the point $p$ is:

$$\left( J_f(p) \right)_{3 \times 3} = \begin{pmatrix} G_{\phi} & G_{\psi} \end{pmatrix}_{3 \times 3} \begin{pmatrix} a_{nx3} \\ b_{nx3} \end{pmatrix}$$

(4.4)

where $G_{\phi}$ is a matrix whose $i$-th column is the gradient of $\phi_i(p)$, and similarly for $G_{\psi}$.

The Hessian of the deformation at the point $p$ is:

$$\left( H_f(p) \right)_{5 \times 3} = \begin{pmatrix} H_{\phi} & H_{\psi} \end{pmatrix}_{5 \times 3} \begin{pmatrix} a_{nx3} \\ b_{nx3} \end{pmatrix}$$

(4.5)

If $p = (x,y,z)$, and $f_{a,b}(p) = (u(x,y,z),v(x,y,z),w(x,y,z))$, then the Hessian of $u$ contains 9 values, of which only 6 are independent, due to the symmetry of the Hessian. In addition, since $u$ is harmonic, $u_{zz} = -u_{xx} - u_{yy}$ so there are actually only 5 linearly independent values in the Hessian. These five values are present in the first column of $H_f(p)$. The second and third columns hold the relevant Hessian values of $v$ and $w$. $H_{\phi}$ is a matrix whose $i$-th column holds the respective five values from the Hessian of $\phi_i(p)$, and similarly for $H_{\psi}$.

In many cases, the shape to be deformed is accompanied by normal vectors. For example, a point cloud which has a normal associated with every point, or a triangulated mesh which contains the normals of the original surface. In these cases, we would like to deform the
normals as well as the shape. We can do this by deforming the plane which is orthogonal to the normal vector at the point \( p \in \Omega \). Given two vectors \( n_1(p) \) and \( n_2(p) \), which span the plane orthogonal to \( \hat{n}(p) \), we have:

\[
f_{a,b}(\hat{n}(p)) = \tilde{n}_1 \times \tilde{n}_2 = (J_f(p)n_1(p)) \times (J_f(p)n_2(p))
\]

where \( \tilde{n}_1 \) and \( \tilde{n}_2 \) span the deformed plane. Plugging this back into the expression for the Jacobian matrix in (4.4):

\[
\tilde{n}_i^T(p) = n_1^T(p)(G_{\phi} \quad G_{\psi})\begin{pmatrix} a \\ b \end{pmatrix}
\]

and similarly for \( n_2 \). Hence, we can pre-compute the relevant matrices:

\[
\begin{pmatrix} \tilde{n}_1^T(p) \\ \tilde{n}_2^T(p) \end{pmatrix}_{2 \times 3} = \begin{pmatrix} (N_{\phi})_{2 \times m} \\ (N_{\psi})_{2 \times m} \end{pmatrix} \begin{pmatrix} a_{m \times 3} \\ b_{m \times 3} \end{pmatrix}
\]

(4.6)

where \( N_{\phi} \) is a matrix whose \( i \)-th column holds the dot product of \( \nabla \phi_i(p) \) with \( n_1(p) \) and \( n_2(p) \), and the same for \( N_{\psi} \).

Equipped with the deformation mapping and its first and second derivatives, we can proceed to define our energy functional, and show how to use it to find the mappings \( a \) and \( b \).

4.2.2 The Energy Functional

Our energy functional is similar to functionals which were used previously in “As-Rigid-As-Possible” deformation applications [SA07, SSP07, BPWG07]. It attempts to satisfy the constraints specified by the user and, in addition, balance detail preservation with smoothness.

**Figure 4.3**: Generating a realistic muscle "bulge" effect by placing a single Jacobian constraint near the marked area, and requiring it to scale. In addition to the Jacobian constraints, we have also placed position constraints causing the hand to rotate.
User constraints.

As our deformation mapping $f$ is defined everywhere in $\Omega$, the user can choose a set of $r$ points $q_i \in \Omega$, and a set of $s$ points $t_i \in \Omega$, and specify their target positions $f(q_i) = f_i$, and their Jacobians $J_f(t_i) = g_i$. The Jacobian constraints can be used to prescribe the orientation of the points $t_i$, or any other affine transform on these points. For example, in Figure 4.3 we have prescribed a set of position constraints, and a Jacobian constraint in the marked location, requiring its affine transform to be a small scale. This allowed us to easily generate the muscle "bulge" effect seen in the figure. The position and Jacobian constraints are hard constraints in our optimization process.

Detail vs. volume preservation.

It is well known that the details of a shape at a point in space are preserved during a deformation if the local transformation that point undergoes is close to rigid. This fact has been used in many As-Rigid-As-Possible deformation methods [BPWG07, SSP07, SA07]. However, recently Lipman et al. [LCOGD07], have shown that detail preservation might come at the expense of volume preservation. In fact, in order to preserve the volume, Lipman et al. scaled the transformations, according to local curvature information. Hence, requiring the Jacobians of all the points in the domain to be rotations, will not necessarily give the desired effect, and might result in volume loss.

However, in an As-Rigid-As-Possible deformation, it is reasonable to assume that the points on the medial axis of the domain, which is a very sparse subset of the domain itself, undergo only rotations. For example, consider Figure 4.4. The figure illustrates the character of the deformation of a bar to an upside-down "U" shape, similar to the deformations in Figure 4.8. This deformation is almost volume preserving, as its relative change in volume is 0.04. The figure color-codes the determinant and condition number of the Jacobian of the deformation on a vertical slice through the shape. The determinant indicates the local change in volume, and the condition number ($\sigma_{\text{max}}/\sigma_{\text{min}}$) indicates the amount of non-uniform scale. As evident in the figure, the volume on the top of the bar increases, the volume on the bottom the bar decreases, but the medial axis of the bar is only bent – the volume near it remains constant. In addition, the condition number is closest to 1 near the medial axis, which indicates that the transformations in this area are close to rotations.
Figure 4.4: The character of the Jacobian of the deformation within one slice through a vertical bar model, bent to a "U" shape. (left) Color-coding of the condition number. (right) Color-coding of the determinant.

In our setting, the local transformation of a point $p$ is simply the Jacobian matrix of $f_{a,b}$ at $p$. Since we can prescribe the Jacobians of the deformation in any location we choose, we prescribe the Jacobians of the medial axis to be as close as possible to rotations. This way we don’t need to compute the desired transformations on the boundary of the domain, as they will be implied from the smoothness of the deformation. The values of these rotations are not known in advance, and will be computed as part of the optimization process.

Hence, we would like to minimize the following rigidity energy:

$$
\min_{a,b,R(\Omega)} E_{\text{Rigid}} (f_{a,b}) = \int_{p \in M(\Omega)} \left\| J_f (p) - R(p) \right\|_F^2 \, d\omega
$$

subject to $\forall p \in M(\Omega)$, $R(p)^T R(p) = I$

The Rigidity Energy

where $M(\Omega)$ is the medial axis of the domain and the norm is the Frobenius matrix norm. The Jacobian is linear in the variables $a$ and $b$, hence if we knew which rotations $R(p)$ each point should undergo, minimizing $E_{\text{Rigid}}$ would be a simple matter of minimizing a quadratic energy. Of course, $R(p)$ are not known in advance, hence the optimization process is non-linear.

Smoothness.

The local transformation of a point $p \in \Omega$ is governed by the Jacobian of the mapping $f$ at $p$. A smooth deformation will have similar transformations for nearby points, and hence a small second derivative. So, to enforce smoothness, we require the Frobenius norm of the Hessian
matrix of each of the deformation mapping components to be as small as possible, by minimizing the following energy:

$$\min_{a,b} E_{\text{Smooth}}(f_{a,b}) = \int_{p \in \Omega} \|H_f(p)\|_F^2 \, d\omega$$

This energy can be simplified using the following observation. Our mapping is harmonic, and hence all the partial and higher derivatives of all its components are also harmonic. According to the maximum principle, a harmonic function on a domain achieves its extremum on the boundary of the domain. Hence, if we minimize the second derivatives on the boundary of the domain, they will also be bounded inside the domain. As a result, we may use the following smoothness energy:

$$\min_{a,b} E_{\text{Smooth}}(f_{a,b}) = \int_{p \in \partial S} \|H_f(p)\|_F^2 \, ds$$

**The Smoothness Energy**

The energy.

Given the points $q_i$ and $t_i$ chosen by the user, and their target positions $f_i$ and target Jacobians $g_i$, respectively, we would like to solve the following optimization problem:

$$\min_{a,b,R(p)} E(f_{a,b}) = \int_{p \in M(\Omega)} \|J_f(p) - R(p)\|_F^2 \, d\omega + \lambda^2 \int_{p \in \partial S} \|H_f(p)\|_F^2 \, ds$$

s.t. $\forall i = 1..r, \; f_{a,b}(q_i) = f_i, \; \forall i = 1..s, \; J_f(t_i) = g_i$

$$\forall p \in M(\Omega), \; R(p)^T R(p) = I$$

**The Continuous Optimization Problem**

where the norms are Frobenius norms, and $R(p)$ are unknown $3 \times 3$ matrices defined on every point $p$ on the medial axis of $\Omega$.

The discrete energy.

Minimizing the energy functional in its current form is difficult, because of the non-linearity of the rotation constraints, and because we do not have closed-form expressions neither for the medial axis, nor for the integrals. Instead, we convert the integrals to a sum of finite samples, as follows. For the smoothness energy, we sample the boundary surface $S$ at $k$ points $w_i$. For the rigidity energy we approximate the medial axis, by sample points on a set of rigidity lines. These lines can be acquired from a skeleton of the deformed shape if it is
available, can be prescribed manually by the user, or can be computed using a skeleton extraction algorithm, such as that in [ATC*08]. Once \( l \) such lines are given, we sample them at \( d \) anchor points \( m_i \), by sampling \( d/l \) points on each rigidity line.

![Image](image_url)

**Figure 4.5:** Deformation using different numbers of anchor points. The leg of the armadillo model (left) was deformed to a bent position, using the specified number of anchor points. The top and bottom rows show different views of the same deformed shape. The source pose shows the five user constraints – red spheres are positional constraints, and black cylinders are orientation constraints.

Since we require smoothness, it is sufficient for the anchor points to be sparsely distributed, so \( d \) can be relatively small. The assumption that a sparse set of rigidity constraints is enough when the deformation is smooth has been used successfully in other deformation methods [WSLG07, SCO04]. Consequently, we only have \( d \) unknown rotation matrices \( R_i \) to solve for. In this setting the optimization problem becomes:

\[
\min_{a,b,R_i} E(f_{a,b}) = \sum_{i=1}^{d} \left\| J_f(m_i) - R_i \right\|^2 + \lambda^2 \sum_{i=1}^{d} \left\| H_f(w_i) \right\|^2
\]

s.t. \( \forall i=1..r, \ f_{a,b}(q_i) = f_i \), \( \forall i=1..s, \ J_f(t_i) = g_i \)

\( \forall i=1..d, \ R_i^T R_i = I \)

**(P1): The Discrete Optimization Problem**

Figure 4.5 shows a comparison of results using a different number of anchor points for the rigidity constraints. The locations of the anchor points were computed using the skeleton extraction algorithm from [ATC*08]. As is evident from the figure, increasing the number of anchors beyond a given point does not significantly improve the results.
Figure 4.6 shows two deformations of the "Armadillo" model. In addition, the figure shows the setup for the deformation – the cage, the original pose, the anchor points and the constraints.

In the following section we describe our optimization scheme for minimizing the deformation energy.

![Figure 4.6: Deformations of the Armadillo model (a) Cage and anchor locations (b) Original pose and constraints (c) Deformed pose (d,e) Another deformed pose from two different viewpoints](image)
4.3 Optimization

To solve the optimization problem (P1) we use the following observation. If the variables $R_i$ are known, then (P1) is a simple linear least-squares problem with linear equality constraints, which has a closed-form global minimum. On the other hand, if $a$ and $b$ are known, then the optimal rotation matrices $R_i$ – those which are closest in Frobenius norm to the Jacobians of the deformation map at $m_i$ - also have a closed-form solution. This solution is a variant of the well-known “Procrustes problem”, obtained using Singular Value Decomposition (SVD). Hence, we can solve (P1) using the alternating least squares method, or "local/global" algorithm [LZX*08, SA07]. In the "local" step, we keep $a$ and $b$ fixed, and solve many small and independent local problems for the $R_i$, while in the "global" step, we keep $R_i$ fixed and solve one global linear system for $a$ and $b$. We repeat these two steps until convergence.

![Figure 4.7](image)

**Figure 4.7:** The "local/global" optimization scheme is robust enough to converge to a good solution from any arbitrary initial configuration. (Left to right) the deformed shape after 1, 3, 17 and 200 iterations, starting from an arbitrary initial configuration. The graphs show the value of the energy functional vs. the number of iterations, starting from different random starting points.
Convergence and robustness.

As was pointed out in previous works [LZX*08, SA07], the "local/global" algorithm is guaranteed to converge, because each step must reduce the energy. In general, the convergence rate depends on the initial configuration. However, since the number of variables is relatively small – the number of anchors for the Jacobian computation is usually smaller than the complexity of the cage – the "local/global" algorithm is robust enough to converge to a good solution from an arbitrary initial configuration. By “arbitrary” - we mean that the Jacobians are initialized to be random $3 \times 3$ matrices.

Figure 4.7 shows the resulting deformation after various numbers of iterations, starting from an arbitrary configuration. In addition, it shows graphs of the value of the energy functional vs. the iteration number using different initial configurations. As can be seen from the graph, our method always converged to the same energy value, no matter which initial configuration was used.

In an interactive modeling environment, the initial configuration can be taken from the values of $a, b$ and $R_i$ in the previous frame. In this case, a small number of iterations of the "local/global" algorithm are usually enough to achieve convergence. This is the initial configuration we used for all the examples in the paper, except the ones in Figure 4.7. In Section 4.4 we provide some more deformation examples, with the timings required to generate them.

Let us now turn to a more detailed description of the minimization process.

Implementation details.

Using the "local/global" approach, the deformation algorithm is relatively simple to implement, and boils down to three steps – the preprocessing step, during which some matrices are pre-computed for later use, the optimization step, where we iterate the "local/global" steps to find the values of $R_i$, and, finally, the deformation step, during which the values of $R_i$ are combined with the user's constraints to generate the final mapping of the input shape. These steps are implemented as a series of matrix operations, on matrices which are "stacked" matrices of $\varphi$, $\psi$ and their derivatives. To avoid clutter in the notation, we redefine (4.3), (4.4) and (4.5) in terms of single matrices as follows:

\[
 f_{a,b}(p) = D_{p}z \quad , \quad J_f(p) = J_{p}z \quad , \quad H_f(p) = H_{p}z
\]
where each matrix represents a concatenation of matrices from (4.3), (4.4) and (4.5) respectively: 

\[ D_p = [\phi, \psi], \quad J_p = [G_{\phi}, G_{\psi}], \quad H_p = [H_{\phi}, H_{\psi}] \]

In addition, \( a \) is a matrix of size \( n \times 3 \), and \( b \) is a matrix of size \( m \times 3 \) (where \( n \) and \( m \) are the number of vertices and faces respectively). \( z \) is the matrix whose first \( n \) rows are \( a \), and last \( m \) rows are \( b \).

Now we can convert the optimization problem to matrix notation using these expressions:

\[
\min_{z, \hat{R}} E(f_z) = \| \hat{J}z - \hat{R} \|_F^2 + \lambda^2 \| \hat{H}z \|_F^2
\]

s.t. \( \hat{D}z = \hat{f}, \quad \hat{J}z = \hat{g} \quad \forall i = 1..d, \quad R_i^T R_i = I \)

where:

\[
\hat{D}_{r(n+m)} = \begin{pmatrix} D_{\phi} & \cdots & D_{\psi} \\ \vdots & \ddots & \vdots \\ D_{\phi} & \cdots & D_{\psi} \end{pmatrix}, \quad \hat{J}_{r(n+m)} = \begin{pmatrix} J_{\phi} & \cdots & J_{\psi} \\ \vdots & \ddots & \vdots \\ J_{\phi} & \cdots & J_{\psi} \end{pmatrix}, \quad \hat{J}_{d(n+m)} = \begin{pmatrix} J_m \\ \vdots \\ J_m \end{pmatrix}, \quad \hat{H}_{r(n+m)} = \begin{pmatrix} H_{\phi} \\ \vdots \\ H_{\psi} \end{pmatrix}
\]

are stacks of deformation, Jacobian and Hessian matrices for the respective points (\( r \) position constraints, \( s \) orientation constraints, \( d \) anchor points and \( k \) Hessian sample points on the boundary of the domain), and:

\[
\hat{f}_{r \times 3} = \begin{pmatrix} f_1 \\ \vdots \\ f_r \end{pmatrix}, \quad \hat{g}_{3 \times 3} = \begin{pmatrix} g_1 \\ \vdots \\ g_s \end{pmatrix}, \quad \hat{R}_{d \times 3} = \begin{pmatrix} R_1 \\ \vdots \\ R_d \end{pmatrix}
\]

are the right hand sides of the linear equations – the user's position and orientation constraints, and the unknown rotation matrices \( R_i \) at the anchor points. The energy can now be written as:

\[
E = \| Az - \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix} \|_F^2, \quad A_{(3d+5k)\times(n+m)} = \begin{pmatrix} \hat{J} \\ \lambda \hat{H} \end{pmatrix}
\]

The constant \( \lambda \) determines the relative weight of the smoothness constraints vs. the rigidity constraints. In our experiments, we took \( \lambda \) to be: \( \lambda = \alpha d |\hat{J}|_F |\hat{H}|_F \). The matrix norms are infinity norms – the maximal \( L_1 \) norms of the rows of the matrix, and \( \alpha \) is a user specified parameter, which can be used to control the stiffness of the deformation. We took \( \alpha \) to be 0.01 in all of our experiments.

Returning to the optimization problem, if \( \hat{R} \) is known, then the minimum of \( E \) is given by:

\[
z_{\text{opt}} = A^T \begin{pmatrix} \hat{R}_{d \times 3} \\ 0_{5k \times 3} \end{pmatrix}, \quad A^+ = (A^T A)^{-1} A^T
\]

Since most of the columns of \( A^T \) are multiplied by zero on the right hand side, we can truncate its last \( 5k \) columns as follows:
\[ z_{\text{opt}} = A_{\text{trunc}}^\top \hat{R} \] , \[ A_{\text{trunc}}^\top = (A^\top A)^{-1} A^\top \]

where \( A_{\text{trunc}}^T \) includes only the first 3\( d \) columns of \( A^T \).

We enforce the user's constraints by removing \( r+3s \) variables from the problem, and computing their value from the remaining variables using the equations:

\[
\begin{bmatrix}
\hat{D} \\
\hat{J}
\end{bmatrix} z_{\text{opt}} = \hat{h} \] , \[ \hat{h}_{(r+3s)+3} = \left( \begin{bmatrix} \hat{f} \\
\hat{g} \end{bmatrix} \right) \]

This can be done, of course, only if the number of hard constraints is less than the number of degrees of freedom in the problem – \( n+m \). However, relatively complicated deformations can be generated with a small number of position and orientation constraints. The resulting system is of the type:

\[ z_{\text{opt}} = B\begin{bmatrix} \hat{h} \\
\hat{R} \end{bmatrix} \]

where \( B \) is computed from partial matrices of \( A \). During the optimization procedure, we need to re-compute the current Jacobian matrices. Hence, we get:

\[ \hat{R}_{\text{new}} = C\begin{bmatrix} \hat{h} \\
\hat{R} \end{bmatrix} \] , \( C = \hat{J}B \)

In addition, once the non-linear iteration has converged, we need to compute the new location of the deformed shape. The new location of the points \( x_1, x_2, ..., x_a \) are given by:

\[
\begin{bmatrix}
\hat{x}_1 \\
\vdots \\
\hat{x}_a
\end{bmatrix} = \begin{bmatrix}
D_{x_1} \\
D_{x_2} \\
D_{x_a}
\end{bmatrix} z_{\text{opt}} = \begin{bmatrix}
\hat{D}_{x_1} \\
\hat{D}_{x_2} \\
\hat{D}_{x_a}
\end{bmatrix} B \begin{bmatrix} \hat{h} \\
\hat{R}_{\text{opt}} \end{bmatrix} = F \begin{bmatrix} \hat{h} \\
\hat{R}_{\text{opt}} \end{bmatrix} \] , \( F = \begin{bmatrix}
D_{x_1} \\
D_{x_2} \\
D_{x_a}
\end{bmatrix} B \)

The matrices \( C \) and \( F \) are pre-computed before the interactive deformation begins. Thus, we have laid out all the building blocks for our algorithm, which can be stated as follows:

**Pre-processing.** Compute the matrices \( C \) and \( F \), given the locations of the user's constraints, the anchor points, the Hessian samples on the boundary and the input shape.

**Optimization.** Select an initial solution \( z, \hat{R} \), and set \( \hat{R}_{\text{global}} = \hat{R} \). Repeat until convergence the following two steps:

1. \( \hat{R}_{\text{local}} = \text{normalize}( \hat{R}_{\text{global}} ) \)

2. \( \hat{R}_{\text{global}} = C\begin{bmatrix} \hat{h} \\
\hat{R}_{\text{local}} \end{bmatrix} \)  \( (4.7) \)
In the local step, the "normalization" of the matrices $\hat{R}_{\text{Global}}$ is done by computing the SVD for each matrix $R_i = USV^T$, and replacing it with $UVT$, up to a change of sign in the last column of $U$, if it has a negative determinant. An additional benefit of this local step, is that since Jacobians with negative determinant are not allowed, the optimization process tends to find a minimum which doesn't contain foldovers. Of course, since this might be overridden by the global step, the occurrence of foldovers depends on the user's constraints. In our experience, for a reasonable set of constraints, foldovers are not likely to appear.

We detect convergence by measuring the amount of change in $\hat{R}_{\text{Global}}$ between two consecutive iterations, which is equivalent to the change in the rigidity energy. It would be better to measure the change in the total energy, however this is more computationally expensive. The computational complexity of each iteration is the complexity of computing $d$ SVD operations, and a matrix-vector multiplication which is $O(d(r+3s+d))$. Detailed performance timings are provided in the next section. Note that the computational complexity of both the local and the global shape do not depend on the complexity of the deformed shape, nor on the complexity of the cage.

**Deformation.** If $\hat{R}_{\text{opt}}$ is the last $\hat{R}_{\text{Global}}$ computed in the optimization step, then the deformed locations are given by:

$$
\begin{bmatrix}
\tilde{x}_i \\
\vdots \\
\tilde{x}_a
\end{bmatrix} = F \begin{bmatrix}
\hat{h} \\
\hat{R}_{\text{opt}}
\end{bmatrix}
$$

(4.8)

The algorithm can be summed up in a few lines of pseudo-code, outlined in Algorithm 4.1.

The pre-process step requires only vector and matrix operations to set up the matrices, and multiply them. Hence, using any efficient linear algebra package, the implementation is relatively straightforward. We provide runtimes of all the steps of the algorithm, for various 3D models, in the next section.

```
Precompute C, F
While (err > threshold) do
    Js_prev = Js
    Js = normalize_jacobians(Js)
    Js = C*[constraints; Js]
    err = norm(Js_prev - Js)
end
new_positions = F*[constraints; Js]
```

**Algorithm 4.1:** Pseudo-code of the deformation algorithm
4.4 Experimental Results

We implemented our “Variational Harmonic Map” (VHM) deformation system as a plugin to the Maya® commercial modeling and animation system. The optimization and deformation step of VHM include two building blocks – SVD computations of $3 \times 3$ matrices, and dense matrix-vector multiply. Dense matrix-vector multiply operations are "embarrassingly parallel" in the sense that they are composed of many independent operations (multiplying one row by one column), which can be performed in parallel. We have exploited this by implementing the computation of Equations (4.7) and (4.8) on the GPU. We used Nvidia’s CUDA programming language with the BLAS library, on an Nvidia Quadro FX 5800 graphics card. Figures 4.1-4.3, 4.6-4.14 demonstrate the application of VHM to different deformation scenarios. In this section we will first compare VHM to two other state-of-the-art deformation methods, and then discuss some of its properties.

4.4.1 Comparison

We compared the performance of VHM to two state-of-the-art deformation methods: "Embedded Deformation" (ED) of [SSP07] and "Adaptive Rigid Cells" (ARC) of [BPWG07]. We compared these methods on three deformation scenarios of a synthetic model, and on one deformation of the beast model – measured by the overall appearance of the deformed shape, the detail preservation and the change in the total volume of the shape. Software was kindly provided by the respective authors.

Before starting the comparison we should state upfront some disadvantages of VHM. Its biggest downside, compared to ED and ARC, is that in additional to the shape to be deformed, the user must also supply a cage bounding the domain, and a set of "rigidity lines". Although generating the rigidity lines is relatively painless (e.g. using a skeleton extraction algorithm such as [ATC*08]), creating a cage is not a trivial problem, and this is mostly understated in existing cage-based deformation methods. In this respect, methods which automatically generate the underlying space representation – the deformation graph for ED and the voxelization for ARC, have an advantage. On the other hand, we believe the benefits of having a cage – a closed form expression for the deformation, faster optimization and separation of unrelated parts of the shape – outweigh the hassle of generating such a cage.
Figure 4.8 shows a comparison between VHM, ED and ARC for the "bar" shape, with three different deformations. For VHM and ED methods, we used the same constraints. For ARC, we achieved the deformation through interactive manipulation. The results shown for ARC are after the final RBF interpolation step.

**Figure 4.8:** Comparison of our deformation method – VHM - with ARC and ED on three deformations of the "bar" model.

**Figure 4.9:** The setup used for the comparisons in Figure 4.8. (top, from left to right) VHM cage and anchors, ARC cells (320) and ED deformation graph (187 vertices). (bottom) The VHM, ARC and ED constraints.
Figure 4.9 shows the setup we used for the deformations in Figure 4.8. Figure 4.10 shows the comparison and setup for the deformation of the beast model. The models were interactively deformed to reach the required pose.

**Figure 4.10:** Comparison of our method (VHM) with ARC and ED on a deformation of the "Beast" model, and the setup used for the deformation. (top row) VHM cage and anchors, ARC cells (2148) and ED deformation graph (300 vertices), (middle row) the VHM, ARC and ED constraints, (bottom row) the deformed models.

To compare the detail preservation of the different models, we computed the *rigidity distortion* of the triangles of the deformed mesh, which is defined similarly to [LZX*08] as:
where $A_i$ is the area of the source triangle, and $\sigma_{1,t}$ and $\sigma_{2,t}$ are the singular values of the Jacobian of the $2 \times 2$ transformation, that transforms the source planar triangle to the deformed planar triangle. Ideally, we would like to compare the singular values of the Jacobian of the 3D transformation, but since for the other two methods we do not have access to the actual deformation function, rather only the end result, this is, unfortunately, not easily done. In addition, we compared the change in the total volume of the deformed shape as:

$$E_{volume} = \left| \frac{vol_{new} - vol_{orig}}{vol_{orig}} \right|$$

The comparison of these errors is given in Table 4.1.

<table>
<thead>
<tr>
<th>Model</th>
<th>$E_{rigid}$</th>
<th>$E_{volume}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VHM</td>
<td>ARC</td>
</tr>
<tr>
<td>A</td>
<td>0.050</td>
<td>0.035</td>
</tr>
<tr>
<td>B</td>
<td>0.069</td>
<td>0.070</td>
</tr>
<tr>
<td>C</td>
<td>0.046</td>
<td>0.043</td>
</tr>
<tr>
<td>Beast</td>
<td>0.022</td>
<td>0.013</td>
</tr>
</tbody>
</table>

**Table 4.1:** Comparison of the rigidity error and volume change of the deformation methods.

As can be seen from Figure 4.8 and Figure 4.10, and Table 4.1, the results of VHM are comparable to those of ARC, however VHM is considerably more efficient (as is shown in Table 4.2), and also simpler to implement. When compared to ED, our method is somewhat better, both in the visual quality of the results – Figure 4.8 shows that the ED method has some noise issues - and in volume preservation.
4.4.2 Locality of the deformation

Figure 4.11 demonstrates that VHM has a local effect, and only regions geodesically close to the manipulated regions are modified, as opposed to unrelated regions which happen to be close in Euclidean distance – the index finger of the hand may be moved without influencing the other fingers.

![Deformation of a tetrahedral mesh model of a hand. One finger is easily moved without influencing the nearby finger, even though it is close in Euclidean distance.](image)

Figure 4.11: Deformation of a tetrahedral mesh model of a hand. One finger is easily moved without influencing the nearby finger, even though it is close in Euclidean distance.

4.4.3 As-Similar-As-Possible deformations

One of the benefits of our "local/global" optimization scheme is that the constraints on the Jacobian matrices of the anchor points can be easily changed from rigidity constraints to other types of constraints, simply by modifying the local step in the optimization algorithm. For example, as was done in previous "local/global" based methods, such as [LZX*08], we can replace the rigidity constraints with similarity constraints by requiring the Jacobian matrices of the anchor points to be similarity transforms. The local step is modified by replacing the "normalization" step of the Jacobian matrices with the following procedure: Compute the SVD for each matrix $R_i = USV^T$, and replace it with $US_{new}V^T$, where $S_{new}$ is a diagonal matrix, whose entries are the average of the diagonal entries of $S$. Such a deformation will not be As-Rigid-As-Possible anymore, as it introduces uniform scale. However, as can be seen in Figure 4.12 interesting exaggeration effects can be generated this way.

For some applications, one might require the deformation to be quasi-conformal, meaning that the condition number of the Jacobian of the deformation is bounded. In these cases, the As-Similar-As-Possible approach is more appropriate than the As-Rigid-As-Possible approach. Figure 4.12 shows the color-coding of the condition number of the Jacobian, for sampled points inside the cage of the Beast model, for the shown deformation. In addition, the
figure shows the histogram of these values. As is evident from the figure, the condition numbers are smaller than 3.5, which indicates that this deformation is quasi-conformal, with a quasi-conformal factor similar to the that of the Green coordinates [LLCO08].

4.4.4 The cage
As opposed to direct manipulation methods [BPWG07, SSP07], which build the underlying representation automatically, cage based methods such as [LLCO08] usually rely on a manually modeled cage. We also use manually modeled cages, but since in our approach the cages are only a mathematical tool, and are not visible to the user, it is important to check
how sensitive the deformation is to the cage used. Specifically, we would like to verify that two reasonable cages result in similar deformations, when the user constraints are identical. To investigate this, we implemented a straightforward algorithm to generate a simple cage by uniform decomposition of space, followed by merging neighboring co-planar faces. Such cages would be somewhat hard to manipulate manually, but since in our method the user does not manipulate the cage directly, this is not an issue. We applied this algorithm to the "Beast" model from Figure 4.1, and using the manually built and automatic cages, we deformed the model interactively. Figure 4.13 shows the two cages, and the deformations resulting from them. As is evident from the figure, the deformations induced by the two cages are very similar, indicating that our method is not very sensitive to the precise cage used.

![Figure 4.13: Two deformations using a manually built cage (left), and an automatic cage (right)](image)

### 4.4.5 Non-articulated shapes

Some objects, such as plate-like objects, do not have an obvious skeleton. In these cases, our method can still be applied by placing the anchors on the medial surface instead of on the medial axis. Figure 4.14 shows two deformation of a "Bumpy plane" model, using different anchors configurations. In both cases, the anchors were placed on the medial surface of the model, but their exact placement was different. As the figure shows, the resulting deformations are very similar, indicating that our method is not very sensitive to the exact
locations of the anchors on the medial surface. The figure also shows the comparison of the results to the deformation of the same model using the ARC method.

Figure 4.14: Deformation of a plate-like object, using two different anchor configurations on the medial surface (left and middle). Deformation using ARC of the same model (right)

4.4.6 Efficiency

Table 4.2 provides the model statistics and the deformation times in milliseconds for our examples. The deformation timing is broken down into the time for one optimization iteration (labeled "Solve") and the time for the matrix-vector multiply which generates the deformation (labeled "Def"). The pre-processing time for all the models was less than a minute. It is clear from the table that our solve times are considerably faster than those reported for ARC [BPWG07] and ED [SSP07], which were run on machines with spec similar to ours. For example, the solve step of ED for the Giraffe model requires 120 milliseconds, using six Gauss-Newton iterations. Using the ARC method, the solve step for a model with 50,000 vertices requires 330 milliseconds for a single Newton iteration. For a larger model of 79,000 vertices, the solve step of VHM requires only 12 milliseconds. The VHM solve includes GPU optimization (for the global part of the "local/global" algorithm), whereas the other methods are implemented on the CPU. However, the ARC and ED optimization algorithms are based on Gauss-Newton iterations using a large sparse matrix. Such algorithms are considerably harder to parallelize than dense matrix-vector multiplies, which can be implemented using off-the-shelf CUDA code. Hence, if one is to compare the best possible implementation of the methods, ours has a distinct advantage. The deformation times are also very fast, with 10 milliseconds for the 170,000 vertex Armadillo model.
<table>
<thead>
<tr>
<th>Model</th>
<th>Vertices</th>
<th>Cage faces</th>
<th>Anchors</th>
<th>Iterations</th>
<th>Solve (ms)</th>
<th>Def (ms)</th>
<th>Total (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar</td>
<td>32,908</td>
<td>208</td>
<td>6</td>
<td>15</td>
<td>0.27</td>
<td>1.84</td>
<td>5.89</td>
</tr>
<tr>
<td>Tet Hand</td>
<td>28,796</td>
<td>288</td>
<td>28</td>
<td>9</td>
<td>0.43</td>
<td>2.27</td>
<td>6.14</td>
</tr>
<tr>
<td>Giraffe</td>
<td>79,226</td>
<td>204</td>
<td>27</td>
<td>33</td>
<td>0.37</td>
<td>3.08</td>
<td>15.29</td>
</tr>
<tr>
<td>Beast</td>
<td>32,311</td>
<td>226</td>
<td>50</td>
<td>10</td>
<td>0.53</td>
<td>2.60</td>
<td>7.90</td>
</tr>
<tr>
<td>Armadillo leg</td>
<td>28,829</td>
<td>68</td>
<td>6</td>
<td>26</td>
<td>0.27</td>
<td>1.87</td>
<td>8.89</td>
</tr>
<tr>
<td>Armadillo</td>
<td>173,101</td>
<td>250</td>
<td>88</td>
<td>13</td>
<td>0.69</td>
<td>10.60</td>
<td>19.57</td>
</tr>
</tbody>
</table>

Table 4.2: Performance measured in milliseconds on an Intel 2.67GHz i7 machine (using a single thread) with 4GB of RAM. "Solve" - time for one optimization iteration, "Def" - time for the matrix multiply in the deformation step. "Iterations" - average number of iterations a typical deformation requires to converge.

4.5 Conclusions and Discussion

In this chapter, we have proposed a new space deformation method ("Variational Harmonic Mapping" – VHM) whose underlying mathematical model is a harmonic mapping. Using this mapping and its derivatives, we defined an energy function whose minimization allows the user to deform the shape using a small number of position and orientation constraints. We showed how to minimize the energy using a very efficient iterative "local/global" algorithm and demonstrated that the resulting deformation is close to an As-Rigid-As-Possible deformation. Its quality is comparable to state-of-the-art space deformation methods, while being considerably faster. Our method is intuitive and easy to use, as it incorporates direct positional constraints.

However user-friendly a deformation method might be, deforming a shape to create poses for a plausible animation sequence is still a time consuming task. Hence, once such an animation is available, it would be useful to be able to transfer it to a different shape. In the next chapter we address this subject and show how the VHM deformation method can be used both for approximating existing animations, and transferring them to new shapes.
In recent years, 3D models have become ubiquitous, and many models are freely available. It would seem, that using existing user-friendly shape deformation methods, such as the VHM method presented in the previous chapter, it is an easy task to download a model of the Web, and "make it move". Unfortunately, creating realistic animations, such as walk-cycles, requires a large amount of skill and expertise, and is quite beyond the reach of a novice user. However, existing animations are available – sample walk cycles, a galloping horse and animations created from motion capture data, to name just a few. In this chapter, we show how to apply the VHM method presented in Chapter 4 in order to "copy" and "paste" animations from an existing model to a new one. This approach is also known as deformation transfer. Unlike existing deformation transfer methods, our approach allows us to transfer deformations between a large variety of shape representations, and not only between closed manifold meshes. Figure 5.1 shows a horse, and two frames from a galloping animation. Given the model of the robot dog, our algorithm creates analog poses of a galloping dog.

5.1 Introduction

Creating a natural deformation of an existing 3D shape is a difficult and time-consuming task. The body of research devoted to 3D shape deformation is huge, and yet the problem is still not considered solved. This problem could be significantly alleviated by reusing existing deformations of similar shapes to generate new deformations. In their influential paper, Sumner et al. [SP04] suggested to reuse deformations using an approach called "deformation transfer."
transfer" (DT). The setup is as follows: Given a source reference shape in some pose, a set of deformed source poses, and a target reference shape in a pose similar to the source reference pose, generate a set of deformed target poses, which are "analogous" in some way to the source deformations. For example, if we are given a horse as the source reference pose, keyframes of a gallop animation as deformed source poses, and a cat as the target reference shape, the output of the deformation transfer are keyframes of a galloping cat. The DT method successfully transfers deformations, however, it is only applicable to single component manifold triangle meshes. As real-life models more than often do not fall into this category – for example: multiple-component meshes, polygon soups, and tetrahedral volumetric meshes, the DT method is somewhat limited. Since realistic characters usually have multiple components – either for separate texture maps, or separate materials, we will concentrate here on this type of models.

Recently, new methods for shape deformation (not deformation transfer) have been proposed [JSW05, LKCOL07, LLCO08, WBGC09, SSP07, BPWG07]. These deform the ambient space the shape “lives” in, instead of the shape itself. Such methods are very versatile, since they can be applied to any shape representation, and are not limited to manifold triangular meshes. In this paper we propose to combine the two ideas, to facilitate spatial deformation transfer. This allows us to transfer a deformation between various shape representations. For example, we can use a skeleton-driven animation as the input deformation, and transfer it to a multiple-component model. Hence, our method is more versatile than the original DT approach.

The deformation transfer process consists of two basic components: 1) Analysis of the source deformation, to extract the "essence" of the deformation, and 2) Synthesis of a deformation of the target shape using the same "essence", thus mimicking the source deformation. In order to transfer the information from the source to the target shapes, some sparse correspondence between the two reference shapes is needed. For example, a foot corresponds to a foot, a nose to a nose and so on.

Given such a correspondence, we propose novel methods for the analysis and synthesis components of the deformation transfer. First, we enclose the source and target reference shapes with two polyhedral domains, cages. Then, to analyze the source deformation, we project the deformation onto a linear space of harmonic maps on the source cage. We use the harmonic basis functions introduced in Chapter 4, and show that many deformations can be
approximated this way with a relatively small error. The main advantage of such a representation, is that the differential properties of the deformation, such as the Jacobians and the Hessians, can be computed analytically and easily transferred to the target deformation. Hence, we consider the Jacobians of the mapping at the correspondence points to be the "essence" of the deformation, and transfer this to the target shape. For the synthesis of the target deformation, we use a method similar to VHM deformation (yet linear), which, in essence is just a reversal of the analysis procedure.

We demonstrate the applicability of our method by transferring deformations to various multiple-components models. We show additionally, that for manifold meshes, our method performs comparably to the original DT method, while avoiding the need to compute a dense correspondence between the meshes. Furthermore, our method is efficiently implemented on the GPU, hence the entire deformation transfer process may be done in real-time.

![Deformation Transfer Illustration](image)

**Figure 5.2:** Terminology of deformation transfer. (top left) Source reference pose. (top right) source deformed pose. (bottom left) target reference pose. (bottom right) target deformed pose – the output of the deformation transfer process.

### 5.1.1 Background and previous work

Since quite a few shapes are involved in the deformation transfer process, some terminology is in order. A deformation transfer method receives as input a *source reference pose*, a *deformed source pose* and a *target reference pose*. The output of the deformation transfer is a *deformed target pose*. Typically many deformed source poses are given, usually the keyframes of an animation sequence. Deformation transfer can be applied to each of them
independently, resulting in keyframes of the target animation sequence. See Figure 1 for an illustration.

The deformation transfer problem can be decomposed into two independent sub-problems – analyzing the source deformation, and synthesizing a new deformation for the target shape. The output of the analysis step is a descriptor of the deformation, having some desirable invariance properties. For example, it can be invariant to global translations and/or rotations of both the reference and deformed poses. Once such a descriptor is generated, it is applied to the target shape to create a new deformation.

One of the most popular deformation descriptors is the so-called deformation gradient. When the deformation is given by two triangular meshes with the same (“compatible”) triangle structure, the deformation gradient of a triangle is the unique affine transform which maps the tetrahedron spanned by the source reference triangle and its normal vector to the tetrahedron spanned by the deformed triangle and its scaled normal. Since, for triangle meshes, the deformation function (on the surface) is a piecewise linear deformation, mapping triangles to triangles, the deformation gradient is just the (piecewise-constant) Jacobian of the deformation function. This deformation descriptor is invariant to global translations, and is widely used for deformation transfer [SP04, ZRKS05, XZY*07], shape editing [YZX*04, SA07] and shape blending [SZGP05]. However, such a deformation descriptor can only be extracted when the source reference and deformed poses are given as compatible triangle meshes. Our method can be considered a generalization of this approach. We approximate the given source deformation by a harmonic mapping of 3D space, whose Jacobian has a closed form expression. Hence, we can compute the Jacobian of the deformation for any point inside the source cage, and transfer it to the target deformation.

Once the deformation gradient has been computed, the new deformation of the target shape can be generated by applying any shape deformation method which takes advantage of these gradients. The most straightforward way to do this on a triangular mesh, which was employed in [SP04, ZRKS05, SZGP05, YZX*04], is to solve a Poisson problem, i.e. to find a mesh whose gradients (relative to the target reference mesh) are as similar as possible, in the least-squares sense, to the deformation gradients of the source deformation. To solve this problem, the gradients of all the target triangles are required. Since usually the correspondence given by the user is a sparse correspondence, only a sparse set of gradients can be transferred from the source to the target mesh. Hence, in order to perform deformation transfer in practice, one
of the following two methods can be used. Either [SP04] generate a dense correspondence from the source to the target reference poses, and transfer all the gradients from the source deformation, or [ZRKS05] transfer the gradients only at the correspondence points, and then propagate these (i.e. interpolate them) to the rest of the target mesh.

The first method is somewhat problematic, as generating a dense correspondence between unrelated meshes is a difficult problem in itself. On the other hand, the second method cannot be applied directly to space deformation transfer, as it will require the computation of harmonic coordinates on the interior of the shape, which is computationally expensive. Hence, we opt for a different approach altogether. We pose the deformation transfer problem as a regular shape deformation problem, where a sparse set of orientation constraints are “learned” from the source deformation, as opposed to a sparse set of positional constraints being obtained interactively from a user. In fact, from this point of view, deformation transfer is even simpler than shape deformation, as one of the most difficult challenges in shape deformation is to infer orientations from positional constraints (which are much more user-friendly than orientation constraints). Moreover, in the deformation transfer setup, there are no positional constraints at all, hence a simpler deformation method can be used. In practice, we generate the target deformation by modifying slightly the space deformation method presented in the previous chapter.

5.1.2 Method overview

The space deformation transfer method receives as input a source reference pose, a deformed source pose, and a target reference pose. The output of the process is a deformed target pose, such that the target deformation mimics the source deformation. See for example, Figure 5.1. The first step in the process is to generate cages for the source and target reference poses. Once the cages are computed, the source poses are analyzed, to compute the best approximating harmonic maps of the deformation. Given a sparse set of corresponding landmarks between the source and target reference poses, we compute the Jacobian of the source deformation at the source landmarks, and transfer them to the corresponding target landmarks. Finally, we seek a harmonic map of the target cage, which best approximates these constraints at the target landmarks, using a variant of VHM.

5.2 Deformation analysis by harmonic projection
The first step in the deformation transfer process is to analyze the source deformation. To be as general as possible, we assume the input poses are given as a collection of points. Hence, 
\[ S = \{ p_1, p_2, ..., p_m \mid p_i \in \mathbb{R}^3 \} \] is the set of \( m \) points of the reference source pose, and \( \tilde{S} = \{ \tilde{p}_1, \tilde{p}_2, ..., \tilde{p}_m \mid \tilde{p}_i \in \mathbb{R}^3 \} \) is the corresponding set of points of some deformed source pose.

Our goal is to find a smooth function \( f \), which maps the reference source pose to the deformed pose: \( f(p_i) = \tilde{p}_i \). If such a function is given analytically, we can compute its Jacobian matrices, and use them as our deformation descriptor.

This analysis can be considered an interpolation problem, and as is common in these cases [Kyt95] we try to approximate \( f \) as a linear combination of a set of basis functions. Inspired by the recent work indicating that harmonic functions generate pleasing deformations [JMD*07, LLCO08, WBCG09], we choose as our basis functions the harmonic functions we used for shape deformation in the VHM method presented in the previous chapter. To compute these functions, the shape should be contained in a polyhedral mesh – a cage, which largely determines the character of the basis functions. Denote by \( C_S = \{ V_S, F_S \} \) a cage enclosing the reference source shape \( S \), where \( V_S \) are the vertices, and \( F_S \) the faces of \( C_S \), and by \( \Omega_S \) the interior of \( C_S \).

Let \( h_j^S : \Omega_S \to R, j = 1..a_S \), be the VHM harmonic basis functions defined in Appendix B, where \( a_S = |V_S| + |F_S| \). We take \( h_j^S \) to be the union of the basis functions for the vertices \( \phi_v \) and the basis functions for the faces \( \psi_f \). The deformation \( f \) is defined by its coefficients \( w_j^S \) on this basis:

\[
    f(p) = \sum_{j=1}^{a_S} w_j^S h_j^S(p)
\]

To project our deformation onto the basis, we must solve the following optimization problem:

\[
    \min_{w_1^S, ..., w_{a_S}^S \in \mathbb{R}^3} E_{\text{Approx}} = \sum_{i=1}^{m} \left( \sum_{j=1}^{a_S} w_j^S h_j^S(p_i) - \tilde{p}_i \right)^2
\]  \( (5.1) \)

This is an over-determined linear least-squares problem, having has a closed-form solution. However, usually \( a_S \ll m \), and we have much more equations than degrees of freedom. Hence, it is not immediately clear why such an approximation would be good, in the sense
that the approximation error $E_{\text{Approx}}$ will be small. To empirically justify the use of these basis functions, we have computed the approximation error (5.1) for a few sets of reference and deformed poses. For each source pose we computed a histogram of the approximation error per vertex from all the poses, given by $E(p_v) = \|f(p_v) - \tilde{p}_v\|_2 \cdot 100 / B$, where $B$ is the size of the bounding box diagonal of the reference pose. Figure 5.3 shows the resulting histograms for two sets of deformations. In addition, the figure shows a few representative deformed poses, overlaid with their "reconstruction" using the basis functions. As is evident from the figure, the error per vertex is quite small, with a mean value of 0.1% of the size of the bounding box diagonal. In addition, Figure 5.4 shows screenshots from a live interaction session, where the projected harmonic map mimics an animation created using a skeleton and linear blend skinning [LCF00], demonstrating that such an animation can be accurately represented using a harmonic map.

**Figure 5.3:** Reconstruction error of harmonic projection, per vertex, as % of the bounding box diagonal, for two sets of poses, including 9 cats, and 48 horses. Also shown - the reference pose within its cage, and a few representative reconstructions (purple), overlaid on the original shape (pink).

Once we have successfully recovered the source deformation $f$ as a linear combination of harmonic basis functions, the Jacobian of the deformation for any point inside the domain $\Omega_S$ can be computed using the gradients of the VHM basis functions, whose expressions are given in Appendix B. The Jacobian of the deformation at a point $p \in \Omega_S$ is:
\[ J^S(p) = \sum_{j=1}^{d_j} w_j^S \nabla h_j^S(p) \]

where \( w_j^S \) is a column vector, and \( \nabla h_j^S \) a row vector.

**Figure 5.4:** Mimicking a linear blend skinning animation using a harmonic maps. (top right) Source shape in reference pose. The shape is "rigged" to be animated in Maya. (top left) target shape in reference pose. (bottom right) A deformed pose created using a skeleton and linear skinning weights. (bottom left) The pose reconstructed using a harmonic map.

Equipped with the Jacobian of the deformation, we can now define our deformation descriptor, which we will later transfer to the target pose. Since we can compute the Jacobian at any point inside the domain, we have the freedom to choose which Jacobians to transfer. An obvious choice would be to densely sample the source domain, and transfer as many Jacobians as possible. However, this would require a dense correspondence between the source and target volumes, which is, in itself, a difficult problem.

Fortunately, transferring Jacobians from the entire volume of the source shape is not necessarily the best approach for deformation transfer, as some of this information may be misleading. Consider the case where the deformation of a bend of a thick bar is transformed onto a thin bar. As was shown in Figure 4.4, to accommodate the bend, the Jacobians on the
boundary of the thick bar must include a large scaling component, whereas the thin bar can bend using less scaling (this phenomenon was investigated in [LCOG*07]). The medial axis of the bars however, will have similar Jacobians. Hence, it is reasonable to transfer only the Jacobians on the medial axis of the source pose to the target pose. Motivated by the same reasons, the VHM shape deformation method presented earlier requires only the Jacobians on the medial axis of the shape to be rotations, in order to generate an As-Rigid-As-Possible deformation.

To summarize, the deformation analysis stage contains the following steps. First, we generate a polyhedral cage enclosing the source reference pose $C_S$ (we elaborate on the cage generation process in Section 5.4). Then we project the deformation on the linear subspace of harmonic maps, spanned by the VHM basis functions, to obtain the coefficients $w_j^S$. Finally, we compute a skeleton of the target cage, which represents its medial axis. This can be done either using a skeleton extraction algorithm (such as [ATC*08]), or, for models such as humans and quadrupeds, using a skeleton embedding algorithm such as [BP07]. Once a skeleton is available, the user maps its joints to the target reference shape. Then we sample each edge of the skeleton, both on the source and target shapes to get the corresponding landmarks $\{ r_i^S \in \Omega_S \mid i = 1..k \}$ on the source, and $\{ r_i^T \in \Omega_T \mid i = 1..k \}$ on the target. Our deformation descriptor is the Jacobian matrices of $f$ at the correspondence landmarks on the source shape: $J(r_i^S)$.

5.3 Deformation synthesis by harmonic reconstruction

Once the deformation descriptor has been extracted from the source deformation, we can apply it to the target reference pose. We would like to create a deformation of the target reference pose, which interpolates a sparse set of Jacobian constraints. Thus, we reverse the analysis process: first, we project our deformation descriptor on the gradients of the VHM harmonic basis functions on the target cage, to find a set of coefficients $w_j^T$. Then, we compute the deformed pose as a linear combination of the VHM basis functions using these coefficients.

Let the target reference shape be given as a set of $n$ points: $T = \{ q_1, q_2, ..., q_n \mid q_i \in \mathbb{R}^3 \}$, and let $C_T = \{ V_T, F_T \}$ be a cage enclosing $T$ with $a_T = |V_T| + |F_T|$. Given the Jacobians $J(r_i^S)$ at the landmarks, we would like to solve the following optimization problem:
\[
\min_{w_1^T,\ldots,w_n^T \in \mathbb{R}^d} \sum_{i=1}^{k} \left\| \sum_{j=1}^{a} w_j^T \nabla h_j^T (r_i^T) - J^S (r_i^S) \right\|_F^2
\]

where \( h_j^T : \Omega \rightarrow R, j = 1..a \) are the VHM harmonic basis functions of the target cage. Unlike in the analysis step, here \( a \gg k \), hence the problem is under-determined. To regularize it, we use the same method as in VHM, requiring, in addition, that the Hessian of the resulting deformation on the boundary of the cage be minimized. In addition, since the gradients determine the deformation only up to a translation, we add one of the landmarks as a single positional constraint. The optimization problem is now:

\[
\min_{w_1^T,\ldots,w_n^T \in \mathbb{R}^d} \sum_{i=1}^{k} \left\| \sum_{j=1}^{a} w_j^T \nabla h_j^T (r_i^T) - J^S (r_i^S) \right\|_F^2 + \lambda \sum_{z \in C} H(\sum_{j=1}^{a} w_j^T h_j^T (z))
\]

As the gradients and Hessians of the basis functions \( h_j^T \) have closed-form expressions, given in Appendix B, solving this optimization problem for the coefficients \( w_j^T \) boils down to solving a linear system of equations. Once we have the coefficients, the deformed target pose is given as a linear combination of the harmonic basis functions, using these same coefficients:

\[
\tilde{q}_i = \sum_{j=1}^{a} w_j^T h_j^T (q_i)
\]

The synthesis step is thus a variant of the VHM deformation method: since we have only gradient constraints, we can avoid the nonlinear step in VHM whose goal is to learn the rotations on the medial axis of the shape, and use a simpler (and more efficient) linear solve.

Figure 5.5 shows poses of a person transferred to the model of a polar bear, and the enclosing cages. The polar bear model has multiple components, but our space deformation method is indifferent to that, and seamlessly transforms the deformation. For this example we sampled 80 gradients from a skeleton with 20 joints.
Figure 5.5: Transferring the poses of a person to a multiple component polar bear, using a skeleton with 20 joints. The reference poses inside their cages are shown in the leftmost column

5.4 Experimental Results

All the steps of our space deformation transfer boil down to solving two sets of linear equations – for analyzing the source deformation, and for synthesizing the target deformation. We have implemented all the required linear algebraic computations using the MKL parallel library, where the multiplication of the large matrices was done on the GPU using the off-the-shelf CUDA BLAS library. The user interaction was implemented as a Maya plug-in.

To evaluate the performance of our deformation transfer method, we have compared it to the original DT method [SP04] which is applicable only to manifold meshes. In this section, we show the results of this comparison, and some additional deformation transfer results. However, first we address some implementation details, which are necessary for using our space deformation transfer algorithm.
5.4.1 Implementation details - Caging

The input to a surface-based deformation transfer application is given by the source and target reference poses, and a deformed source pose. In our case, to apply the space deformation analysis and synthesis we need to envelope the source and target reference shapes with polyhedral cages $C_S$ and $C_T$ respectively. Although cage-based deformation methods are quite popular [JSW05, JMD*07, LKCOL07, LLCO08], to the best of our knowledge there are no published methods for automatically generating a reasonable cage from an input shape.

The cage generation problem can be posed as follows. Given a set of points $p_i \in \mathbb{R}^3$, and a constant integer $\alpha$, find a genus zero closed polyhedron $C = \{V,F\}$, where $V$ are its vertices, and $F$ its faces, which encloses the volume $\Omega$, such that $|F| < \alpha$, $p_i \in \Omega$ for all $i$, and the volume of $\Omega$ is minimal. The volume requirement aims at keeping the cage close to the outer surface of the shape, as this will generally generate pleasing deformations. The requirement on the number of faces is needed, since the complexity of the deformation depends on the complexity of the cage. Hence, for example, a cage with a few thousand vertices is prohibitive in conjunction with existing deformation methods.

Our cage generation method does not solve the posed problem in the general case, but does provide a heuristic for automatic cage generation. To generate a cage, we do the following:

1. **Points and normals.** Create from the input shape a set of points with normal directions which represent the shape. If the shape is given as a collection of triangles (polygon soup, multiple component mesh or manifold mesh), this can be done by sampling the input triangles, and assigning to each sample the normal to the face it was sampled from. If the input shape is a tet-mesh, the same procedure can be done on the outer surface of the volume mesh.

2. **Envelope.** Create an "envelope" $E$ of the input shape, by applying a reconstruction algorithm such as [KBH06] to the points and normals found in step 1.

3. **Simplify.** Simplify $E$, within a tolerance $\varepsilon$.

4. **Offset.** Compute an offset position for each vertex of $E$, by moving it in the normal direction by step size $s$. The normal direction is computed as the area-weighted average of the face normals.
5. **Repeat.** Repeat steps 2,3,4 until the simplification step reaches the required number of faces.

Each time we offset the surface of the cage, the geometry of the shape becomes less complex, and it is easier to correctly approximate it with a small number of faces. The ideal way to solve the problem, would be to compute the minimal offset required, in order to approximate the offset surface within the tolerance $\varepsilon$, with less than $\alpha$ faces. However, this is a difficult problem and our iterated reconstruction/simplification/offset method attempts to heuristically find the solution. For all of the automatically generated cages used in this chapter, a few such iterations were enough to generate the cage. Figure 5.6 shows a few steps in the cage generation process, as well as the final cage. Most of the cages used in the chapter were generated this way (except the cage of the person model, which was generated manually).

![Figure 5.6: Cage generation for the robot-dog model. Every row shows one iteration of the offset-reconstruction-simplification steps. The bottom right model is the cage we used for the deformation transfer in Figure 5.1](image)

### 5.4.2 Deformation Transfer Results

The best state-of-the-art deformation transfer method is the original DT method. Hence, we compare our performance to DT, using source and target meshes which are closed manifold meshes. It is worth noting upfront that there are some classes of shapes for which DT’s results will be superior to ours. For example, our method generates less pleasing results for facial
animations. However, as face models are usually given as a manifold mesh in the first place this is not a major disadvantage.

Figure 5.7 shows the comparison between our method and DT [SP04] for the deformation transfer of the cat to the lion. As is evident in the figure, on a manifold mesh our results are comparable to DT. Our method, however, does not need a full correspondence between the meshes (which the DT method does), and required only a skeleton with 18 joints to transfer the gradient information from the source to the target meshes.

**Figure 5.7:** Comparison of our spatial deformation transfer method, with DT [SP04], on a manifold triangular mesh. (top row) Source poses. (middle row) Result of DT. (Bottom row) Our result. Left most poses on each row are the reference poses enclosed in their cages.

Figure 5.8 shows the transfer of a few poses of a horse to the robot dog multiple-components model. Note that the pose of the head of the reference shapes is slightly different between the horse and the dog. Thus, in all the deformed poses the movement of the head is slightly different, as it is relative to the reference pose. We used 37 joints for the skeleton, and a total of 80 transferred landmarks. In fact, we used 80 landmarks for all the examples in this chapter.

Figure 5.9 shows the transfer of the cat poses to a multiple component model which represents the anatomical skeleton of a cat. Here, the tails of the cats are not aligned in the reference shapes, hence their orientation is different in the target deformed shapes. The skeleton used for the transfer contained only 28 joints. Finally, in Figure 5.10 we transfer the
poses of the person to a multiple component gremlin character using a skeleton with 18 joints. The deformation transfer for this figure was completely automatic – we used the software from [BP07] to embed a generic biped skeleton into both reference shapes, thus creating corresponding skeletons.

**Figure 5.8:** Transferring the poses of a horse to a multiple component robot dog. The reference poses enclosed in their cages are shown in the left most column.

**Figure 5.9:** Transferring the poses of the cat to a multiple component model of the anatomical skeleton of a cat. The reference poses enclosed in their cages are shown in the left most column.
Our deformation transfer method requires only the solution of linear systems of equations, and hence is very efficient. Thus, it is possible to deform a shape using any standard deformation scheme (such as directly manipulating the vertices, or through a skeleton rig), while simultaneously transferring the deformation to another shape, at interactive rates. To give a sense about the timings involved, transforming the deformation from the horse to the robot-dog model, takes 12 ms per deformed pose.

5.5 Conclusions

In this chapter, we have presented a method to extend the basic deformation transfer technique, so that it is applicable to a larger variety of shape representations, and not only to single component manifold triangle meshes. We showed how to analyze a given deformation, by projecting it to a linear subspace of harmonic functions, and how to adapt our space deformation method to generate the target deformation from a sparse set of Jacobian constraints.
6 Conclusions and Discussion

In this work we addressed a few geometry processing applications, and showed their superiority to existing methods. We proposed a novel discrete conformal parameterization method, and showed how it can also be used for shape retrieval. We additionally suggested a novel space deformation algorithm, and applied it further to animation transfer. Our basic tools for all applications were conformal and harmonic maps, both in $\mathbb{R}^3$ and on the surface of a mesh. We applied these maps in the discrete setting – for a piecewise linear surface, and in the continuous setup, for continuous functions in $\mathbb{R}^3$. All our algorithms are efficient, most are based on pure linear formulations, and some are optimized and implemented on the GPU.

We briefly addressed the similarities between deformation and parameterization applications. Studying this issue further is a promising direction for future work. For example, our parameterization method reduces the area distortion inherent to conformal maps by introducing singularities – locations in the map which break the conformality. Can this method be used to generate planar conformal deformations with low area distortion? In addition, to find the parameterization we first go through the conformal factor – Is this method applicable to mesh deformation? And in the other direction – can we adopt deformation paradigms and use them for parameterization? Can the space deformation method introduced in Chapter 4 be used for volumetric harmonic mapping between two shapes?

With such a plethora of existing parameterization and deformation algorithms, one might wonder if these problems have not found yet a satisfying solution. However, if one considers how people interact with real deformable objects, such as a chunk of clay, it is obvious that, at least for deformation, we have yet to find the ultimate algorithm. In the meanwhile, we hope that this work has pushed the state-of-the-art a little further, towards useful and efficient geometry processing algorithms.
Appendix A - The discrete conformal scaling factor

In this section we derive an infinitesimal version of our Eq. (2.1) - the Poisson equation for the conformal scaling factor - the key to our parameterization approach. The derivation is based on differentiating the cosine law, an approach inspired by Luo [Luo06]. Given a mesh $M$ and its embedding $X$, consider a vertex $v \in V$ and its 1-ring neighborhood $\{v_1, v_2, \ldots, v_d \mid (v_i, v) \in E\}$ where $d$ is the degree of $v$. Let $f_i$ be the face $(v, v_i, v_{i+1})$. Then the Gaussian curvature of $v$ is:

$$k_v = 2\pi - \sum_{i=1}^{d} \alpha_{v_i}^f$$

In fact, this Gaussian curvature is a function of only the edge lengths of the faces near $v$. This is because the edge lengths determine the angles, which in turn define the curvature. Considering the Gaussian curvature as a function of the edge lengths, one may wonder how the curvature would change if the edge lengths undergo an infinitesimal perturbation. As the curvature depends only on the angles, we need to compute the partial derivatives of the angles near $v$ with respect to the edge lengths. Luckily, an angle near $v$ depends only on the edge-lengths of the face to which it belongs, so we may consider just a single triangle for the computation of the derivatives.

Let $f$ be the triangle whose edge lengths are $l_1, l_2$ and $l_3$. Denote by $\alpha_1, \alpha_2, \alpha_3$ the angles of this triangle, where $l_i$ is the length of the edge opposite to the vertex $v_i$. Now, how does a small perturbation of the edge lengths $l_i, i \in \{1,2,3\}$ affect $\alpha_i$? To answer this, we differentiate the cosine law, which governs the connection between angles and edge lengths in a triangle:

$$\cos \alpha_i = \frac{l_j^2 + l_k^2 - l_i^2}{2l_j l_k} \tag{A1}$$

where $\{i,j,k\}$ is some cyclic permutation of $\{1,2,3\}$.

Differentiating (A1) with respect to $l_j$ and using the cosine law again we get:

$$-\sin \alpha_i \frac{\partial \alpha_i}{\partial l_j} = \frac{l_i}{l_j l_k} \cos \alpha_k \tag{A2}$$

Applying the sine law to (A2) we have:
\[
\frac{-\partial \alpha_i}{\partial l_j} = \cot \alpha_k \frac{1}{l_j} \tag{A3}
\]

Now, let \( u_j = \log(l_j) \):

\[
\frac{\partial \alpha_i}{\partial u_j} = -\cot \alpha_k \tag{A4}
\]

Note, that we can similarly differentiate the cosine law for \( \alpha_j \) with respect to \( l_i \) and obtain the symmetric relation:

\[
\frac{\partial \alpha_i}{\partial u_j} = -\cot \alpha_k \tag{A5}
\]

Since the sum of the angles is the constant \( \pi \), we have:

\[
0 = \frac{\partial}{\partial u_i} \left( \alpha_i + \alpha_j + \alpha_k \right) = \frac{\partial \alpha_i}{\partial u_i} + \frac{\partial \alpha_j}{\partial u_i} + \frac{\partial \alpha_k}{\partial u_i}
\]

\[
\frac{\partial \alpha_i}{\partial u_i} = -\left( \frac{\partial \alpha_j}{\partial u_i} + \frac{\partial \alpha_k}{\partial u_i} \right)
\]

Using (A5) gives:

\[
\frac{\partial \alpha_i}{\partial u_i} = -\left( \frac{\partial \alpha_j}{\partial u_j} + \frac{\partial \alpha_k}{\partial u_k} \right) \tag{A6}
\]

Finally, we may compute the change \( d\alpha_1 \) in the angle \( \alpha_1 \) as a result of the changes \( du_i \) using the chain rule and (A5) and (A6):

\[
d\alpha_1 = \frac{\partial \alpha_1}{\partial u_1} du_1 + \frac{\partial \alpha_1}{\partial u_2} du_2 + \frac{\partial \alpha_1}{\partial u_3} du_3
\]

\[
= \frac{\partial \alpha_1}{\partial u_2} (du_2 - du_1) + \frac{\partial \alpha_1}{\partial u_3} (du_3 - du_1) \tag{A7}
\]

\[
= -\cot \alpha_2 (du_2 - du_1) - \cot \alpha_3 (du_3 - du_1)
\]

Now, let us define the change in the edge length caused by the scaling function \( \phi \) as follows:

\[
l_i^{\text{new}} = l_i^{\text{old}} \exp(0.5 \left( \phi_j + \phi_k \right)) \tag{A8}
\]

which means an edge is scaled using the mean of the scaling function at its two endpoints. Using (A8) we get the following:

\[
du_j = \log l_i^{\text{new}} - \log l_i^{\text{old}} = \log \left( \frac{l_i^{\text{new}}}{l_i^{\text{old}}} \right) = 0.5 \left( \phi_j + \phi_k \right)
\]

hence:
\[ du_j - du_i = 0.5(\phi_i + \phi_k) - 0.5(\phi_j + \phi_k) = 0.5(\phi_i - \phi_j) \quad (A9) \]

Plugging (A9) into (A7) we have:

\[ d\alpha_i = -0.5\cot\alpha_3(\phi_1 - \phi_2) - 0.5\cot\alpha_2(\phi_1 - \phi_3) \]

Returning to the original problem of the 1-ring neighborhood, let us compute the change in the curvature \( dk_v \) as a function of the changes \( du_i \). Let \( f_i \) be the face \((v, v_i, v_{i+1})\), then:

\[ dk_v = -\sum_{i=1}^{d} d\alpha_{fi} = -\sum_{i=1}^{d} \left(-0.5\cot\alpha_{fi}^L(\phi_{v_i} - \phi_{v_{i+1}}) - 0.5\cot\alpha_{fi}^L(\phi_{v_{i+1}} - \phi_{v_{i-1}})\right) \]

Since on neighboring faces \( f_i \) and \( f_{i-1} \), the same factor appears, this simplifies to:

\[ dk_v = 0.5\sum_{i=1}^{d} \left(\cot\alpha + \cot\beta\right)(\phi_{v_i} - \phi_{v_{i+1}}) \quad (A10) \]

where \( \alpha \) and \( \beta \) are the angles opposite the edge \((v, v_i)\) on the faces \( f_i \) and \( f_{i-1} \). Note that the right hand side of (A10) is exactly the discrete Laplace-Beltrami operator (with cotangent weights).

To summarize, we have the following relation between the scaling function and the resulting curvature change:

\[ \nabla^2\phi_v = dk_v \approx k_v^{new} - k_v^{orig} \quad (A11) \]

This implies that given a mesh, a small enough curvature change \( dk_v \) can be achieved by solving the above equation for \( \phi \), and modifying the edge lengths using (A8).

The update method based on the FEM approach introduced in Section 2 is different from the one in (A8). However, a simple Taylor expansion for small \( \phi \) shows that the two are equivalent.

Eq. (A11) proven here is correct for small curvature changes, and we use it for general curvature changes, large and small alike. As it turns out, and as is shown in the results section, in practice one can prescribe relatively large curvature changes and still get reasonable results.
Appendix B – The VHM basis functions and their derivatives

B.1 The mappings $\varphi$ and $\psi$

The mappings $\varphi$ and $\psi$ are defined on the vertices and faces of the mesh, respectively, so we must provide for each face a scalar value $\psi_t$ and for each vertex a scalar value $\varphi_v$. The values of $\varphi_v$ are determined as a sum of values on the faces neighboring $v$. Given a point $p \in \Omega$, and a face $t = (u,v,w) \in F$, we define a tetrahedron $T$ spanned by these four points, as in Figure B.1.

Figure B.1: Notations for the definitions of $\varphi_v(p)$, $\psi_t(p)$ and their derivatives.

On this tetrahedron, $4\pi \omega_t$ is the signed solid angle at the point $p$, subtended by the face $t$, and $vol_t$ is its signed volume. $N_t$ is the normalized outward pointing normal of $t$, and $A_t$ is the area of the face $t$. Following [Ura00] we obtain:

$$\psi_t = - \sum_{i \in t} C'_i \left( J'_i \cdot N_i \right) - \frac{3}{A_t} \omega_t vol_t$$

$$\varphi_v = \sum_{t \in N(v)} \frac{1}{2A_t} P_t \cdot J'_v$$
B.2 The gradients

Again following [Ura00], and taking the derivative of $\varphi_v$:

$$\nabla \varphi_v = \sum_{t \in N(v)} \frac{1}{2A_t} P_t \times d^t_v$$

B.3 The Hessians

To derive the Hessian matrices for $\psi_t$ and $\varphi_v$ we need the Jacobian matrix of $P_t$, and the gradient vector of $\omega_t$. These are:

$$\nabla \omega_t = \sum_{i=1}^3 J_{v_i} \tilde{C}_{v_i} \left( \|e_{v_{i+1}}\|^{-1} + \|e_{v_{i+2}}\|^{-1} \right)$$

$$J(P_{t(u,v,w)}) = \begin{pmatrix} \tilde{e}_v^t + \tilde{e}_w^t \tilde{C}_u^t \\ \tilde{e}_u^t + \tilde{e}_w^t \tilde{C}_v^t \\ \tilde{e}_u^t + \tilde{e}_v^t \tilde{C}_w^t \end{pmatrix}_{3x3} \left( \begin{array}{c} d_u^t \\ d_v^t \\ d_w^t \end{array} \right)_{3x3} + \nabla \omega_t^T N_t$$

where, given a vector $v$, $[v]_x$ is the skew symmetric matrix, such that for any vector $w$, $[v]_x w = v \times w$. Finally the Hessian matrices of $\varphi_v$ and $\psi_t$ are:

$$H(\psi_t) = -J(P_t)$$

$$H(\varphi_v) = -\sum_{t \in N(v)} \frac{1}{2A_t} [d^t_v]_x J(P_t)$$
Bibliography


האלגוריתם שביצעו קונפורמי ל槭תי מודל: היכולת של המודל לשפר את תוצאות המשימה (בצורה אידיאלית) היא למרות עבור איך הקונפורמי במלואו של מודל משמש. על המתקבל המשימה (בצורה אידיאלית) היא למרות עבור איך הקונפורמי במלואו של מודל משמש. על המתקבל המשימה (בצורה אידיאלית) היא למרות עבור איך הקונפורמי במלואו של מודל משמש. על המתקבל המשימה (בצורה אידיאלית) היא למרות עבור איך הקונפורמי במלואו של מודל משמש. על המenerima המשימה (בצורה אידיאלית) היא למרות עבור איך הקונפורמי במלואו של מודל משמש. על המenerima המשימה (בצורה אידיאלית) היא למרות עבור איך הקונפורמי במלואו של מודל משמש. על המenerima המשימה (בצורה אידיאלית) היא לмар...
קירות האטמוספרות, והן מביעות את המ destructor של הנפש гарなくなる, כדי לבליה
ולעשותBushfeld התפקיד התפקיד של דמותدعימה. בפרק?

ל撕וס, בבעד צור הMosser אל מערכת Flow. המעבדה על כולם מתכתיים דימוי –

טרופים ומזכיות וקואופerties. או מוששים במודולר אל שבבראש ורצוע, ובראש הגזים, גם הפרה.

והם על מורים מונופולים. כל המגדליים המגדלים מינרלים והיצורים את זהות, ייעל, שמיים וקלים לופטרו.
ה כעת נעשתה בבניהית פרופ' חיות גוטסמן בפקולטה למדעי המחשב.

אני מודה לטךיני על תמיכה הספרות והדרכה בהשתלבותי.

Technion - Computer Science Department - Ph.D. Thesis PHD-2009-09 - 2009
אלגוריתמים גיאומטריים בדידים
ליעקב רשתות

törör על מחקר

לשם מילוי תחילק של תדרישות קבילה התואר
דוקטור לפילוסופיה

ופרליה בק-מק

هوוש ל㌧ שכנורי - מכון טכנולוגי liéוראל
סיוון 2009
אלגוריתמים נדירים בידידים
לעיבוד רשתות

מירה בן-חן