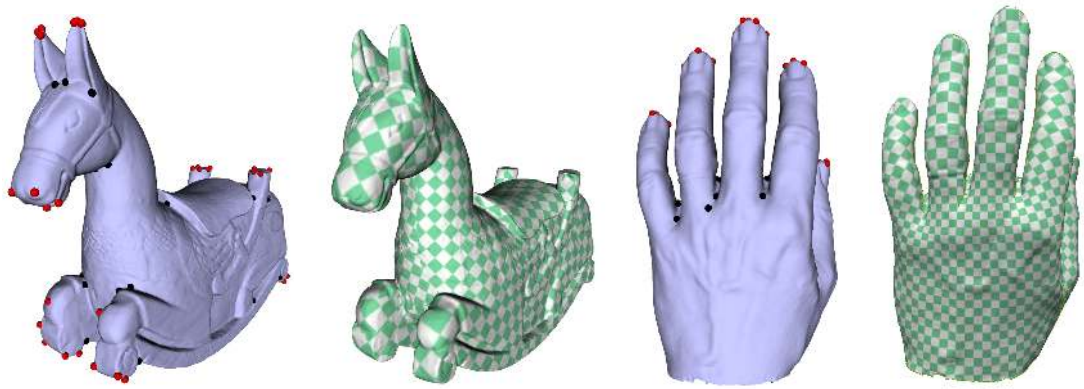
EUROGRAPHICS 2008 / G. Drettakis and R. Scopigno
(Guest Editors)

Volume 27 (2008), Number 2

Conformal Flattening by Curvature Prescription and Metric Scaling

Mirela Ben-Chen, Craig Gotsman and Guy Bunin

Technion – Israel Institute of Technology



Abstract

We present an efficient method to conformally parameterize 3D mesh data sets to the plane. The idea behind our method is to concentrate all the 3D curvature at a small number of select mesh vertices, called cone singularities, and then cut the mesh through those singular vertices to obtain disk topology. The singular vertices are chosen automatically. As opposed to most previous methods, our flattening process involves only the solution of linear systems of Poisson equations, thus is very efficient. Our method is shown to be faster than existing methods, yet generates parameterizations having comparable quasi-conformal distortion.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

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 *conformal*, and area preserving parameterization *authalic*. Most surfaces do not have a parameterization which is *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 *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 *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 π. 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 *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 *cone singularities* and the entire Gaussian curvature of the mesh is concentrated at those singularities. Once this abstract sparse curvature distribution is computed, a *metric* (i.e. edge lengths) having this target curvature is found, and this is subsequently realized by an *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 in this paper is closest in spirit to that of [KSS06] and improves upon it in a number of ways.

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 the scope of this paper. 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. The inset figure illustrates this in 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 *cone singularities*. These are vertices which, at the end of the parameterization process, do *not* have zero Gaussian curvature as the other vertices of the mesh. Their method is based on the concept of *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.

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 *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.

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.

The rest of the paper is organized as follows. In the next section, we present our parameterization method. In Section 3 we show how to leverage our parameterization method in order to automatically find the location and curvature of the cone singularities. Section 4 describes the full algorithm for the parameterization of an arbitrary mesh, including some technical implementation details. We conclude with some results in Section 5 and a short discussion in Section 6.

2. Metric scaling

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:

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

The *angles induced by a metric* L_M are:

$$A_{L_M} = \left\{ \begin{array}{l} \alpha_v^f = \arccos\left(\dfrac{l_{vu}^2 + l_{vw}^2 - l_{uw}^2}{2 l_{uv} l_{uw}} \right) \\ f = (u,v,w) \in F \end{array} \right\}$$

The (discrete) *Gaussian curvature induced by a metric* L_M is:

$$K_{L_M} = \left\{ k_v = 2\pi - \sum_{f \in F_v} \alpha_v^f \mid v \in V \right\}$$

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. 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 2π. 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 these approaches, 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.

2.3. 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]:

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

where ∇^2 is the Laplace-Beltrami operator of the manifold, K^{orig} is the Gaussian curvature of the original manifold, and K^{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 ϕ to the change of Gaussian curvature, in the special case that the new Gaussian curvature distribution is a sum of delta functions, is:

$$\nabla^2\phi = K^{orig} - K^{new}$$

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 matter. 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):

$$\nabla^2\phi_v \approx k_v^{new} - k_v^{orig}$$

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^{ϕ} are computed by first solving the following discrete Poisson equation on the mesh vertices:

$$\nabla^2\phi = K^T - K^{orig} \qquad (1)$$

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

ϕ 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:

$$s_{ij} = \int_0^1 e^{\phi(t)}dt = \begin{cases} \dfrac{e^{\phi_j} - e^{\phi_i}}{\phi_j - \phi_i} & \phi_i \neq \phi_j \\ e^{\phi_i} & \phi_i = \phi_j \end{cases}, \quad (i,j) \in E$$

The target metric is then computed by multiplying the original edge lengths of the embedding by the edge's scaling factors:

$$L^T = \left\{ l_{ij}^T = l_{ij} \cdot s_{ij} \mid (i,j) \in E, l_{ij} \in N_X \right\}$$

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 1: *Flattening and texture mapping of parameterized meshes.*

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 1 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.

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.

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 χ 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:

$$P_{ij} = \begin{cases} w_{ij} & (i,j) \in E, i \notin S, \sum_j w_{ij} = 1 \\ 1 & i = j, i \in S \\ 0 & Otherwise \end{cases} \quad (2)$$

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).

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:

$$P = \begin{pmatrix} S_{nxn} & T_{nxs} \\ 0_{sxn} & I_{sxs} \end{pmatrix}$$

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:

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

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

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

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

$$G = (I-S)^{-1}T \quad (3)$$

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:

$$K_S^{New} = K_S^{Orig} + G^T K_{V \setminus S}^{Orig} \quad (4)$$

In order to compute G as in (3), observe that P has the structure of the incidence matrix of the mesh, thus it is easy to check that (3) is equivalent to the following *Poisson* equation:

$$L\hat{G}_i = \delta_i \quad (5)$$

where L is a generalized Laplacian operator of the mesh, defined with weights w_{ij} as in P, and δ_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].

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 ϕ 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^{ϕ} will preserve the Gaussian curvature, ϕ is unique only up to an additive constant. Hence, we may assume that ϕ has zero mean. Now, let us consider how the parameterization will be distorted. If ϕ 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 ϕ 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, ϕ 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 ϕ is obtained by a global computation on the entire mesh. The inset figure shows the set of vertices with highest absolute curvatures in the hand model, which are obviously not a good selection as singularities.

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 (4)
3. Compute ϕ using the Poisson equation (1). If $max(\phi)-min(\phi) > \varepsilon$, where ε 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 ε, 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 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 2π 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.

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 ϕ as a solution to the Poisson equation.
3. Compute the new 2D edge lengths using ϕ.
4. Compute the 2D coordinates from the new lengths.

4.1. Computing ϕ

Computation of ϕ involves solving the Poisson equation (1). We use the discrete symmetric Laplacian with cotangent weights derived from the input 3D mesh [PP93]: $w_{ij}=0.5(\cot(\alpha)+\cot(\beta))$, where α and β are the angles opposite edge (i,j). Since the co-rank of the Laplacian of a connected mesh is 1, ϕ 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. (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. (1) can be solved very efficiently, e.g. with Matlab's *mldivide* operator, which uses the CHOLMOD package [Dav05].

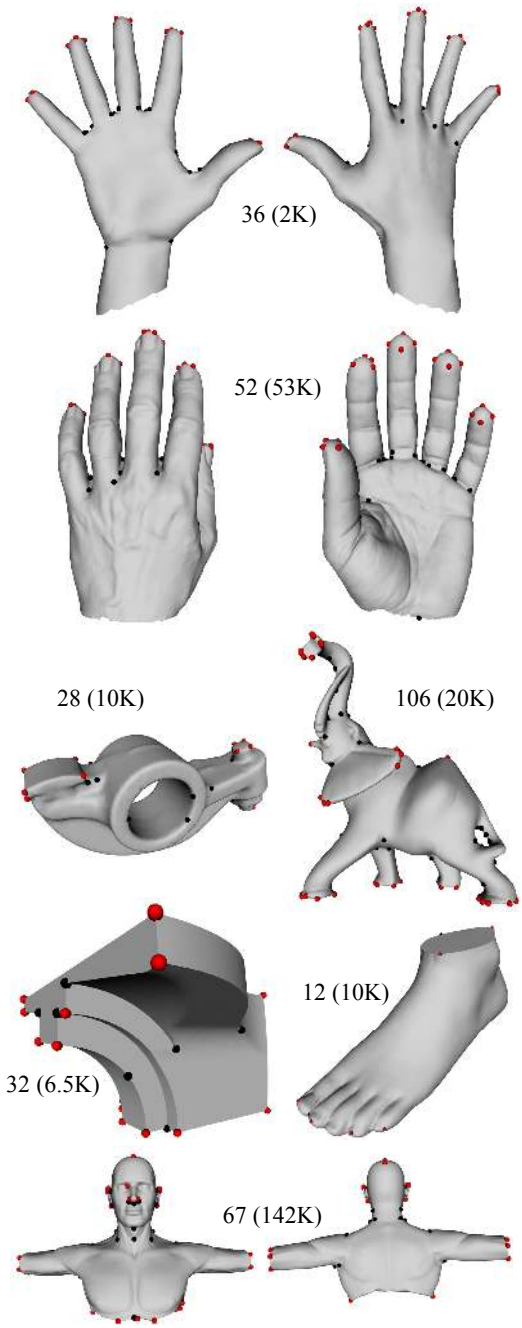


Figure 2: *Cone singularities. Red (black) spheres indicate positive (negative) singularities. The number of singularities is stated, along with the total number of vertices (in parentheses).*

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 (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 (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 (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.

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.

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 1 this error is quite small, on the order of 1e-5 π. 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.

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.

Model	#Δ	Curvature Error (units of π)
fandisk	13K	4.6e-6
rockerarm	20K	1.4e-5
foot	20K	2.7e-5
gargoyle	20K	2.9e-5
elephant	40K	1.6e-5
horse	40K	1.8e-5
hand	106K	3.2e-6
torso	284K	1.4e-6
isi-horse	358K	8.4e-7

Table 1: *Curvature error relative to target curvature.*

Table 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.

Model	#Δ	Quasi-conformal distortion			Runtime (sec)		
		CP	LABF	CF	CP	LABF	CF
fandisk	13K	1.013	1.007	1.012	20	0.9	0.5
rockerarm	20K	1.045	1.027	1.028	21	1.5	0.4
foot	20K	1.029	1.016	1.020	29	1.4	0.6
gargoyle	20K	1.213	1.032	1.037	29	1.6	0.7
elephant	40K	1.028	1.022	1.027	44	4.8	1.1
horse	40K	1.035	1.017	1.077	62	4.1	1.3
hand	106K	1.009	1.008	1.009	902.7	26.0	3.8
torso	284K	-	1.004	1.005	-	86.2	24.3
isi-horse	358K	-	-	1.009	-	-	31.6

Table 2: *Comparison between parameterization methods*

As is evident from Table 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 3 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 4 compares the flattening of the foot mesh generated by the three methods. Finally, Figure 5 shows the models listed in Table 2 texture mapped and color-coded with the quasi-conformal distortion generated by CF.

Figure 3: *Quasi-conformal distortion color coded over the mesh*

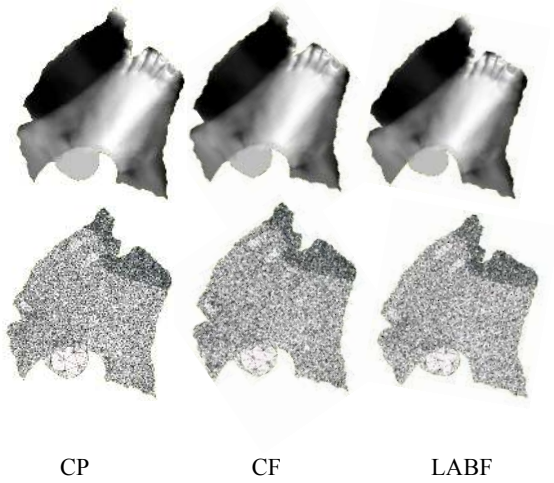


Figure 4: *Flattening of the foot model*

6. Conclusions and Discussion

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 effi-

cient. Despite its simplicity, resulting in runtimes which are much faster than other state-of-the-art methods, it still yi elds results comparable to those methods.

Figure 5: *Texture mapping and color coding of quasi-conformal distortion, for the meshes of Table 2.*

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 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 ϕ as a piecewise linear function.

Since mesh parameterization is a very fundamental operation, a prerequisite to other more sophisticated geometry processing techniques, we are sure that our approach, and its underlying theory, will find more applications. Manipulating the intrinsic geometry (the curvature distribution and its associated metric) seem to be a promising alternative to processing the extrinsic geometry (e.g. the vertex coordinates) directly. We are currently exploring the implications of this for applications such as quad remeshing of 3D mesh datasets. For quad remeshing to be applicable, applying a checkerboard texture to the parameterization should generate a seamless texture. As our method only takes care of the scale of the texture across the cut boundary, and not of the directions of the iso-lines of the parameterization, this problem remains open.

Acknowledgments

All the models except the cut cow and cut bunny were obtained from the AIM@SHAPE shape repository. The cut cow and bunny are courtesy of Alla Sheffer.

We thank Pierre Alliez and Camille Wormser for thought provoking discussions, Tamal Dey and Alla Sheffer for providing us with software, and the authors of the LinABF and CP algorithms for providing us with comparison data. Thanks also to the anonymous reviewers for their many useful comments.

This work was partially supported by European FP6 IST NoE grant 506766 (AIM@SHAPE), the Israel Ministry of Science, and the INRIA-Technion Associate Team GEOTECH project.

References

[Bun07] BUNIN, G.: A continuum theory for unstructured mesh generation in two dimensions. *Computer Aided Geometric Design* (2007), to appear.

[CY00] CHUNG, F., YAU, S.: Discrete Green's functions. *J. Comb. Theory Ser. A* 91, 1-2 (2000).

[Dav05] DAVIS T.-A.: CHOLMOD Version 1.0 User Guide (http://www.cise.ufl.edu/research/sparse/cholmod). (2005).

[DBG*06] DONG S., BREMER P.-T., GARLAND M., PASCUCCI V., HART J.: Spectral surface quadrangulation. *ACM SIGGRAPH* (2006).

[DLS07] DEY T., LI K., SUN J.: On computing handle and tunnel loops. *Proc. IEEE NASAGEM workshop* (2007), to appear.

[GGH02] GU X., GORTLER S., HOPPE H.: Geometry images. *ACM SIGGRAPH* (2002).

[HG00] HORMANN K., GREINER G.: MIPS: An efficient global parameterization method. *Curve and Surface Design* (2000).

[JKG07] JIN M., KIM J., GU X.-D.: Discrete surface Ricci flow: theory and applications. *Mathematics of Surfaces XII* (2007).

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

[KNP07] KÄLBERER F., NIESER M., POLTHIER K.: QuadCover - Surface parameterization using branched coverings. *Computer Graphics Forum,* 26(3), 375-384, (2007).

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

[Luo06] LUO F.: Some applications of the cosine law. *Discrete Differential Geometry conference, Oberwolfach. http://www.math.tu-berlin.de /geometrie/ps/DDGWorkshopSlides/DDG_talk_Luo.pdf* (2006).

[KSS06] KHAREVYCH L., SPRINGBORN B., SCHRÖDER P.: Discrete conformal mappings via circle patterns. *ACM Transactions on Graphics* 25(2) (2006).

[PP93] PINKALL U., POLTHIER K.: Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics,* 2 (1993).

[RLL*06] RAY N., LI W. C., LÉVY B., SHEFFER A., ALLIEZ P.: Periodic global parameterization. *ACM Trans. Graph*. 25(4), 1460–1485, (2006).

[SH02] SHEFFER A., HART J.: Seamster: Inconspicuous low distortion texture seam layout. In *Proceedings of IEEE Visualization* (2002).

[SLMB05] SHEFFER A., LÉVY B., MOGILNITSKY M., BOGOMJAKOV A.: ABF++: Fast and robust angle based flattening. *ACM Trans. Graph.* 24(2), (2005).

[SPR06] SHEFFER A., PRAUN E., ROSE K.: Mesh parameterization methods and their applications. *Foundation and Trends in Computer Graphics and Vision 2*(2), (2006).

[SY94] SCHOEN R., YAU S.-T.: *Lectures on Differential Geometry*. Intl Press (1994).

[TACSD06] TONG Y., ALLIEZ P., COHEN-STEINER D., DESBRUN M.: Designing quadrangulations with discrete harmonic forms. In *Proc. Eurographics/ACM Symp. on Geom. Proc.* (2006), pp. 201–210.

[WC07] WANER S., COSTENOBLE S.: *Finite Mathematics and Applied Calculus, 4th Edition*. Thomson Brooks/Cole, 2007.

[ZLS07] ZAYER R., LÉVY B, SEIDEL H.-P.: Linear angle based parameterization. In *Proc. ACM/Eurographics Symposium on Geometry Processing* (2007).

Appendix A: The discrete conformal scaling factor

In this section we derive an infinitesimal version of our Eq. (1) - the Poisson equation for the conformal scaling factor - the key to our 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 $\{v_1, v_2, \dots, v_d \mid (v_i,v) \in E\}$ 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:

$$k_v = 2\pi - \sum_{i=1}^{d} \alpha_v^{f_i}$$

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 α_1, α_2, α_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 α_i? To answer this, we differentiate the cosine law, which governs the connection between angles and edge lengths in a triangle:

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

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 α_j with respect to l_i and obtain the symmetric relation:

$$\frac{\partial \alpha_i}{\partial u_j} = \frac{\partial \alpha_j}{\partial u_i} = -\cot\alpha_k \tag{A5}$$

Since the sum of the angles is the constant π, we have:

$$0 = \frac{\partial(\alpha_i + \alpha_j + \alpha_k)}{\partial u_i} = \frac{\partial \alpha_i}{\partial u_i} + \frac{\partial \alpha_j}{\partial u_i} + \frac{\partial \alpha_k}{\partial u_i}$$

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

Using (A5) gives:

$$\frac{\partial \alpha_i}{\partial u_i} = -\left(\frac{\partial \alpha_i}{\partial u_j} + \frac{\partial \alpha_i}{\partial u_k}\right) \tag{A6}$$

Finally, we may compute the change $d\alpha_1$ in the angle α_1 as a result of the changes du_i using the chain rule and (A5) and (A6):

$$\begin{aligned} d\alpha_1 &= \frac{\partial \alpha_1}{\partial u_1} du_1 + \frac{\partial \alpha_1}{\partial u_2} du_2 + \frac{\partial \alpha_1}{\partial u_3} du_3 \\ &= \frac{\partial \alpha_1}{\partial u_2}(du_2 - du_1) + \frac{\partial \alpha_1}{\partial u_3}(du_3 - du_1) \\ &= -\cot\alpha_3 (du_2 - du_1) - \cot\alpha_2 (du_3 - du_1) \end{aligned} \tag{A7}$$

Now, let us define the change in the edge length caused by the scaling function ϕ as follows:

$$l_i^{new} = l_i^{old} \exp(0.5(\phi_j + \phi_k)) \tag{A8}$$

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

$$du_i = \log l_i^{new} - \log l_i^{old} = \log\left(\frac{l_i^{new}}{l_i^{old}}\right) = 0.5(\phi_j + \phi_k)$$

hence:

$$du_j - du_i = 0.5(\phi_i + \phi_k) - 0.5(\phi_j + \phi_k) = 0.5(\phi_i - \phi_j) \tag{A9}$$

Plugging (A9) into (A7) we have:

$$d\alpha_1 = -0.5\cot\alpha_3(\phi_1 - \phi_2) - 0.5\cot\alpha_2(\phi_1 - \phi_3)$$

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:

$$dk_v = -\sum_{i=1}^{d} d\alpha_v^{f_i} = -\sum_{i=1}^{d}\left(-0.5\cot\alpha_{v_{i+1}}^{f_i}(\phi_v - \phi_{v_i}) - 0.5\cot\alpha_{v_i}^{f_i}(\phi_v - \phi_{v_{i+1}})\right)$$

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

$$dk_v = 0.5\sum_{i=1}^{d}(\cot\alpha + \cot\beta)(\phi_v - \phi_{v_i}) \tag{A10}$$

Where α and β 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:

$$\nabla^2\phi_v = dk_v \approx k_v^{new} - k_v^{orig} \tag{A11}$$

This implies that given a mesh, a small enough curvature change dk_v can be achieved by solving the above equation for ϕ, 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 ϕ 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.