An Algorithm for the Construction of Intrinsic Delaunay Triangulations with Applications to Digital Geometry Processing

Matthew Fisher
Caltech

Boris Springborn
TU Berlin

Peter Schröder
Caltech

Alexander I. Bobenko
TU Berlin

Abstract

The discrete *Laplace-Beltrami* operator plays a prominent role in many Digital Geometry Processing applications ranging from denoising to parameterization, editing, and physical simulation. The standard discretization uses the cotangents of the angles in the immersed mesh which leads to a variety of numerical problems. We advocate the use of the *intrinsic* Laplace-Beltrami operator. It satisfies a local maximum principle, guaranteeing, *e.g.*, that no flipped triangles can occur in parameterizations. It also leads to better conditioned linear systems. The intrinsic Laplace-Beltrami operator is based on an *intrinsic* Delaunay triangulation of the surface. We detail an incremental algorithm to construct such triangulations together with an overlay structure which captures the relationship between the extrinsic and intrinsic triangulations. Using a variety of example meshes we demonstrate the numerical benefits of the intrinsic Laplace-Beltrami operator.

1 Introduction

Delaunay triangulations of domains in $\mathbb{R}^2$ play an important role in many numerical computing applications because of the quality guarantees they make, such as: among all triangulations of the convex hull of a set of points in the plane the Delaunay triangulation maximizes the minimum angle. Many error estimators in finite element approaches to the solution of partial differential equations, *e.g.*, are directly related to the minimum angle and are more favorable if this minimum is maximized (for an extensive discussion see [Shewchuk 2002]). The construction of such triangulations for domains in $\mathbb{R}^2$ is now standard textbook material and a basic ingredient in many meshing algorithms. For immersed surfaces in $\mathbb{R}^3$, which are given as simplicial 2-manifold meshes (possibly with boundary), the picture is not nearly as clear.

Algorithms which numerically manipulate such meshes are of great importance in Digital Geometry Processing applications. Examples include surface denoising [Desbrun et al. 1999], thin-shell simulation [Grinspun et al. 2003], construction of discrete minimal surfaces [Pinkall and Polthier 1993], surface parameterization [Desbrun et al. 2002; Lévy et al. 2002], computation of discrete conformal structures [Gu and Yau 2003; Mercat 2001], and geometric modeling [Botsch and Kobbelt 2004], among many others. Similar to the setting of domains in $\mathbb{R}^2$ one also finds that meshes which satisfy an *intrinsic* Delaunay criterion give rise to better numerical behavior in all the above geometry processing examples.

A classic algorithm to convert a given planar triangulation into a Delaunay triangulation involves edge flipping, whereby an edge which violates the local Delaunay criterion is flipped until no such edge remains. In the same vein, one can construct an *intrinsic Delaunay triangulation* (see Figure 1) of a simplicial surface in $\mathbb{R}^3$ by performing *intrinsic edge flips* (see Figure 2). Importantly, because the edge flips are intrinsic the shape of the original embedded mesh *does not change*. This notion of intrinsic Delaunay triangulation (iDT) takes into account only the intrinsic geometry of the mesh, *i.e.*, the mesh is considered as an abstract surface with a metric that is locally Euclidean except at isolated points (the vertices of the mesh) where it has cone-like singularities. The relevant data read off from the input mesh are the combinatorics of its edge graph as well as the length of each edge. With this data alone one may now ask of each interior edge whether it satisfies the local Delaunay condition since the associated predicate can be based solely on the observed edge lengths and local combinatorics (and thus entirely on intrinsic data). The intrinsic flip algorithm proceeds as follows: While there is an edge that violates the local Delaunay criterion, perform a combinatorial flip on it and update its length. Note that this procedure does not change the intrinsic geometry of the input mesh at all. One may visualize the iDT of such a mesh as a graph of geodesic edges drawn on the original simplicial surface, as shown in Figure 1. (Here *geodesic* denotes a straightest edge, not necessarily a shortest edge [Polthier and Schmies 2002].) It is not hard to see that the intrinsic flip algorithm terminates, thus producing an intrinsic triangulation with all interior edges satisfying the local Delaunay criterion [Indermitte et al. 2001]. Recently, Bobenko and Springborn [2005] have shown that as in the planar case the iDT is essentially unique and satisfies a *global* empty circumcircle property. A technical difficulty that one encounters when dealing with iDTs is that they are not necessarily *regular* triangulations. A triangulation is called *regular* if each triangle is incident with three different edges and three different vertices. It is called *strongly regular* if it is regular and the intersection of two triangles is either empty or *one* edge or *one* vertex, and if the intersection of two edges is either empty or *one* vertex. The usual definition of the term *triangulation* implies strong regularity. This is too narrow for our purposes. For example, edges of an iDT may form loops (see Figure 3). Therefore, we do not require triangulations to be regular.

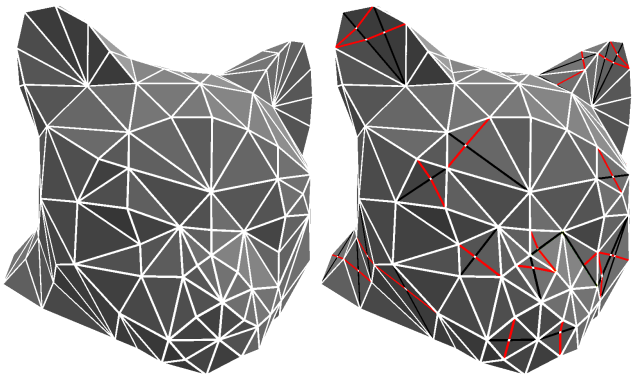

Figure 1: *Left: carrier of the (cat head) surface as defined by the original embedded mesh. Right: the intrinsic Delaunay triangulation of the same carrier. Delaunay edges which appear in the original triangulation are shown in white. Some of the original edges are not present anymore (these are shown in black). In their stead we see red edges which appear as the result of* intrinsic flipping. *Note that the red edges are straight,* i.e.*, geodesic, within the original surface.*

With an iDT in hand one may define an *intrinsic Laplace Beltrami* (iLB) operator [Bobenko and Springborn 2005]. In contrast to the *extrinsic Laplace Beltrami* (eLB) operator, which is based on the original triangulation, the iLB operator has many favorable numerical properties. For example, in the construction of discrete harmonic parameterizations one can guarantee that the computed pa-

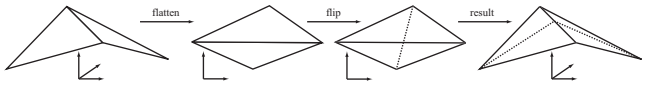

Figure 2: *Given a pair of adjacent triangles in $\mathbb{R}^3$ their intrinsic geometry can be visualized by isometrically mapping them to the plane. If the edge is not Delaunay it is flipped inside the original surface (original edges: solid; flipped edge: dotted). The flipped edge is a geodesic segment in this surface and the surface remains unchanged.*

rameterization are free of flipped triangles because a discrete maximum principle holds for the iLB while this is generally not the case for the eLB. (These issues with the eLB have been the cause for many proposals to numerically "fix" the eLB.) More generally, the iLB is numerically better conditioned than the eLB leading to more efficient applications in particular when higher powers of the iLB are required. The iDT is also useful in other settings. For example, Circle Patterns [Kharevych et al. 2006], used in the construction of discrete conformal maps, require the intrinsic Delaunay property of each edge. In that setting the use of iDTs leads to results with far lower quasi-conformal distortion (see Figure 10).

Contributions In this paper we describe an algorithm for the construction of iDTs for immersed simplicial 2-manifold meshes (possibly with boundary) of arbitrary topology. First, we detail the basic intrinsic edge flipping algorithm which requires little more than a standard edge based data structure. Then we describe an extension of this flip algorithm that maintains a data structure encoding the overlay of the original triangulation and the current triangulation with flipped edges. This is necessary if one wants to interpolate some data that is given on vertices with respect to the original triangulation and other data with respect to the iDT. For example, the 3D vertex coordinates should be interpolated with respect to the original mesh, but texture coordinates which were calculated using the iLB operator should be interpolated with respect to the iDT. The overlay data structure is maintained using only combinatorial predicates.

It is well known that edge flipping can have a long running time (edge flipping in planar triangulations, *e.g.*, takes $\Theta(n^2)$ time). Empirically we find the algorithm to run quite fast however, and its runtime to be easily dominated by subsequent numerical computing tasks. We present statistics for a variety of input meshes and show, among other observations, that the condition number of the associated Laplace-Beltrami operator can be noticeably reduced when using the iDT. Coupled with the guarantee of satisfying a discrete maximum principle, this results in more robust and efficient numerical behavior for a host of DGP applications.

2 Intrinsic Delaunay Triangulations

We begin this section with a brief recall of the relevant intrinsic geometry of piecewise linearly immersed simplicial meshes, before describing the edge flipping algorithm without (Section 2.2) and with (Section 2.3) overlay maintenance.

2.1 Piecewise Flat Surfaces

A *piecewise flat surface* (PF surface) is a 2-dimensional manifold equipped with a metric such that every interior point of the manifold has a neighborhood that is isometric to a neighborhood in the plane, *except* for a number of isolated points, the *cone points*. Each cone point has a neighborhood that is isometric to a neighborhood of the apex of a Euclidean cone. Small circles of radius r around a cone point have length αr with $\alpha \neq 2\pi$. The number α is the *cone angle* of the cone point (which may be smaller or larger than 2π). Its *Gaussian curvature* is $2\pi - \alpha$. We always assume that the boundary of a PF surface, if present, is piecewise geodesic.

A concrete way to construct PF surfaces is through gluing polygons: take a set of planar polygons together with a partial isometric edge pairing, *i.e.*, a set of pairs of different polygon edges such that the edges in each pair have the same length and every edge is contained in at most one pair. Gluing the polygons along the paired edges one obtains a PF surface. The unpaired edges make up the boundary. We emphasize that the notion of a PF surface belongs to the *intrinsic geometry of surfaces*: here we are not interested in whether or how a PF surface can be isometrically embedded in $\mathbb{R}^3$. When we speak of gluing polygons together, we mean the abstract construction of identifying corresponding points along the edges.

A possible representation which describes a *triangulation of a PF surface* consists of a graph structure to describe an abstract surface triangulation (which need not be regular) together with a labeling of the edges by positive numbers signifying edge length. The only constraint on the edge lengths is that they must satisfy the triangle inequalities for each face. For if that is the case, one can construct all the triangles and glue them to obtain a PF surface. All cone points are vertices of the triangulation. Such a triangulation is a *Delaunay triangulation of a PF surface* if for each interior edge the local Delaunay criterion is satisfied: the sum of the opposite angles in the adjacent triangles is less than π. For more on Delaunay triangulations of PF surfaces see [Bobenko and Springborn 2005] and references therein.

Note: Since a Delaunay triangulation of a PF surface is not necessarily regular, it is essential that the data structure which is used to represent the abstract surface triangulation can represent non-regular triangulations. For example, winged-edge, half-edge, or quad-edge data structures may be used. A data structure based on triangle-vertex incidences is not suited.

2.2 Intrinsic Delaunay Flipping Algorithm

Any 3D surface mesh with flat faces is intrinsically a PF surface. In general, every vertex of such a mesh is a cone vertex. Suppose an immersed surface (in $\mathbb{R}^3$) is given in the form of a 2-manifold triangle mesh. More precisely, we have a 2-manifold complex $K = (V,E,T)$ of vertices, edges, and triangles together with the point positions $P(v_i) = p_i \in \mathbb{R}^3$ (one for each vertex $v_i \in V$). Piecewise linear (PL) interpolation over each triangle then defines the *carrier* of the surface. For this surface we seek an iDT. This triangulation depends only on the complex K and metric data associated with each edge $e_{ij} \in E$. This metric data is the Euclidean length $L(e_{ij}) = l_{ij} = \|p_i - p_j\|$ of each embedded edge $e_{ij} \in E$. The pair (K,L) constitutes the input to the iDT algorithm which returns a complex with corresponding *intrinsic* lengths $(\tilde{K},\tilde{L})$. Note that once L is computed P will play no further role in the algorithm.

For each edge in $\tilde{E}$ we will also report all edges in E crossed by it (if any). Note that an edge in $\tilde{E}$ may cross a given edge in E multiple times.

While K is generally strongly regular, we do not require regularity. In fact, the output of the algorithm will in general not be regular (see Figure 3).

The transformation of (K,L) into $(\tilde{K},\tilde{L})$ is straightforward and based on the classic edge flipping algorithm:

Require: (K,L)
Ensure: $(\tilde{K},\tilde{L})$ is intrinsic Delaunay
  $\forall e \in E$: Mark(e)
  Stack $s \leftarrow E$
  while !Empty(s) **do**
    $e_{ij} \leftarrow$ Pop(s) and Unmark(e_{ij})

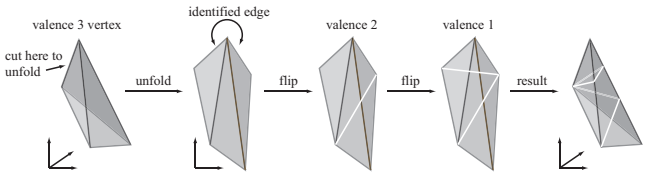

Figure 3: *Even if the input mesh is strongly regular it is quite easy to arrive at non-regular configurations through intrinsic edge flipping. Here a vertex of valence* 3 *is reduced to a valence* 1 *vertex through two intrinsic flips (original edges: black; flipped edges: white).*

if !Delaunay(e_{ij}) **then**
  $e_{kl} \leftarrow$ Flip(e_{ij}) and compute l_{kl}
  for all $e \in \{e_{kj}, e_{jl}, e_{li}, e_{ik}\}$ **do**
    if !Mark(e) **then**
      Mark(e) and Push(s, e)
    end if
  end for
end if
end while
return $(\tilde{K}, \tilde{L}) \leftarrow (K, L)$

The fact that this algorithm terminates is shown in [Indermitte et al. 2001]. The proof is essentially the same as for planar triangulations. Infinite loops cannot occur because a suitable function on the set of triangulations decreases with every edge flip. In the case of PF surfaces an added complication results from the fact that the number of triangulations on a fixed vertex set may be infinite. So an additional requirement for a "suitable function" is that it is unbounded on any infinite set of triangulations. That the output $(\tilde{K}, \tilde{L})$ is globally intrinsic Delaunay, *i.e.*, that the local Delaunay property implies a global empty circumcircle property, requires some more work [Bobenko and Springborn 2005].

To implement the algorithm two essential functions are required, (1) evaluation of the Delaunay(e_{ij}) predicate, and (2) computation of the new edge length l_{kl} if edge e_{ij} is flipped. Both require only the edge lengths of the incident triangle(s) t_{ijk} and t_{lij}. Many implementations are possible. Here we describe one which uses only the four basic arithmetic operations and square roots, in particular, no inverse trigonometric functions. It also has the added benefit that it computes the coefficients of the Laplace-Beltrami operator (see Section 3).

(1) First, compute the tangents of the half angles of the triangles using the half-angle theorem: If a, b, and c are the side lengths of a triangle, and α is the angle opposite the side with length a, then

$$\tan\frac{\alpha}{2} = \sqrt{\frac{(a-b+c)(a+b-c)}{(a+b+c)(-a+b+c)}}.$$

From these half-angle tangents one may compute the cotan weights using the identity

$$\cot\alpha = \frac{1-\tan^2\frac{\alpha}{2}}{2\tan\frac{\alpha}{2}}.$$

For a boundary edge, Delaunay(e_{ij}) returns true. For an interior edge, Delaunay(e_{ij}) returns true if the cotan weight $w_{ij} = \cot\alpha_{ij}^k + \cot\alpha_{ij}^l$ on the edge e_{ij} is nonnegative, false otherwise. (Here α_{ij}^k denotes the angle opposite edge e_{ij} in triangle t_{ijk}.)

(2) We need to compute the other diagonal f in a quadrilateral with known sides a, b, c, d and diagonal e as shown in Figure 4. Again

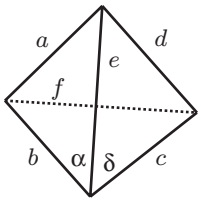

Figure 4: *We need to compute the length of f from the given lengths of a, b, c, d, and e.*

we base our computation on half-angle tangents $\tan\frac{\alpha}{2}$ and $\tan\frac{\delta}{2}$ that we obtain using the half angle theorem. From these we compute

$$\tan\frac{\alpha+\delta}{2} = \frac{\tan\frac{\alpha}{2}+\tan\frac{\delta}{2}}{1-\tan\frac{\alpha}{2}\tan\frac{\delta}{2}},$$

then

$$\cos(\alpha+\delta) = \frac{1-\tan^2\frac{\alpha+\delta}{2}}{1+\tan^2\frac{\alpha+\delta}{2}},$$

and finally

$$f = \sqrt{b^2+c^2-2\cos(\alpha+\delta)}.$$

The correct execution of the algorithm depends both on the correct evaluation of the Delaunay predicate as well as on the correct computation of the lengths of flipped edges since later flips can depend on earlier flips in this manner. This is quite different from standard flipping algorithms which work with an embedding. There the data (vertex positions) upon which the predicate depends does not change as the algorithm proceeds. In our current implementation we have used only double precision floating point computations and not observed any ill effects in our examples, though correctness cannot be assured in this setting.

2.3 Incremental Overlay

The information contained in $(\tilde{K}, \tilde{L})$ is sufficient to assemble the iLB operator and perform computations with it. In some applications the resulting values (at vertices) are to be interpolated across the carrier surface. Recall that the carrier surface is defined through PL interpolation with respect to the input triangulation. However, the data arising from a computation using the iLB operator is most naturally PL interpolated with respect to the iDT. (For a quantitative comparison of the resulting distortion errors see Section 3 and Figure 9.) To be able to perform both types of interpolation simultaneously, as is required in texture mapping, for example, one needs a graph structure representing the overlay of both triangulations K and $\tilde{K}$. We describe an incremental algorithm which maintains this overlay during flipping.

We maintain a graph structure (*e.g.*, a half-edge data structure) which describes, at each stage of the flip algorithm, the overlay of the original triangulation and the current one. The vertices of the original and of the current triangulation (they have the same vertex set) are also vertices of the overlay graph. Additionally the overlay graph contains vertices corresponding to points where an edge of the original triangulation is crossed by an edge of the current triangulation. These we will not call vertices but *crossings*. The overlay graph structure distinguishes between vertices and crossings so that we can tell them apart. The edges of the overlay graph we will call *segments* because they are segments of edges of the original and current triangulations. Each segment is labeled **c**, **o**, or **oc**: **o** if it is part of an edge that belongs to the original triangulation but not the current one (we will also call such an edge an **o** edge), **c** if it is part of an edge that belongs to the current triangulation but not the original one (a **c** edge), and **oc** if it is part of an edge that belongs to

both the original and current triangulation (an **oc** edge). Note that from this overlay graph structure one can reconstruct both the original and the current triangulation. Edges labeled **o** (**c**) correspond to sequences of more than one **o** (**c**) segment because an edge of the current triangulation that does not belong to the original triangulation must cross an original edge (and vice versa). A **oc** edge on the other hand corresponds to a single **oc** segment, because such an edge cannot cross any other edges. Each crossing has four adjacent segments which are alternatingly labeled **o** and **c**.

Initially, the original triangulation is also the current one, so it is also the overlay graph, with all edges being **oc** segments. Performing a flip now requires updates of the overlay graph. We will see that this requires only combinatorial information. After the new combinatorics are established, the segment lengths can be updated independently (if desired).

Updating the Overlay Topology We illustrate this using the examples shown in Figure 5. First, consider the flip shown on the left. The horizontal **c** edge (consisting of four segments) which is to be flipped, is removed. This requires merging of pairs of **o** segments (previously crossed by the **c** edge) incident to **c/o** crossings. This leaves us with a **c** quadrilateral (formed by the two triangles incident on the edge) with five **o** segments on its interior. Inserting the (vertical) flipped **c** edge will lead to some new crossings with **o** segments. Importantly, we can tell which of these **o** segments will be crossed and in what order: The **o** segments that are crossed are those that are incident to the left as well as right boundary of the quadrilateral. (In this accounting the top and bottom vertices do not belong to either boundary.) This eliminates two of the five segments from further consideration. The remaining segments are incident to the boundaries *in the same order* as they are crossed by the flipped **c** (vertical) edge. This is so because there are no vertices in the interior of the quadrilateral and crossings cannot occur between **o** segments. So we know which **o** segments to split and how to insert the flipped **c** edge. None of these considerations required any coordinates or lengths.

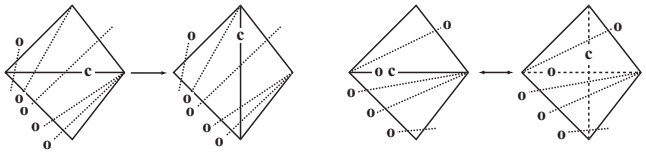

Figure 5: *Maintaining the overlay topology. Left: the common case when a* **c** *edge is flipped and remains* **c**. *Right: the special cases when an* **oc** *edge is flipped, or, reversely, a* **c** *edge is flipped onto an* **o** *edge. Each case also illustrates a variety of possible* **o/c** *crossings.*

The flip shown on the right of Figure 5 illustrates the special case when an **oc** edge is flipped, or, vice versa, a **c** edge is flipped *onto* an **o** edge. The procedure is the same as described above, except that flipping a **oc** edge does not lead to its removal but rather only to a label change (making it **o**). Reversely, if a **c** edge is flipped onto an **o** edge, the result is an **oc** edge.

In either example, some or all of the four boundary edges of the quadrilateral might be **oc** instead of **c**. Such edges do not contain any crossings. The procedure remains the same. This concludes the description of all possible cases.

Updating Lengths First note that during the algorithm, lengths of segments are not required, only lengths of edges. If actual crossing intersection points, *e.g.*, in terms of segment lengths, are required, one can find these by laying out the quadrilateral hinge of a given edge. We may think of segment lengths as barycentric coordinates with respect to their containing edge (and its length). Assuming that all segment lengths are known before an edge flip, one can lay out an isometric copy of the **c** quadrilateral in the plane and obtain coordinates for the four vertices and all crossings of **o** segments with the boundary. These can be used to obtain coordinates for the new crossings of **o** edges with the flipped edge, and thus to calculate the new segment lengths. The layout process must be able to cope with identified edges/segments or vertices, since the triangulation is in general not regular. In particular a given vertex in the mesh may have multiple different (u,v) coordinates in the layout.

3 IDT in Applications

The discrete Laplace Beltrami operator is central in applications ranging from denoising/smoothing (intrinsic mean curvature flow [Desbrun et al. 1999]), to editing (using the bi-Laplace-Beltrami operator [Botsch and Kobbelt 2004]), texture mapping (using a pair of discrete harmonic functions [Desbrun et al. 2002]), and construction of discrete minimal surfaces [Pinkall and Polthier 1993], to give but a few references (see also the references in [Grinspun et al. 2005]). In all of these cases one needs to solve (sequences of) linear problems involving a system matrix Δ with off-diagonal entries

$$\Delta_{ij} = -(\cot \alpha_{ij}^k + \cot \alpha_{ij}^l)$$

for edge e_{ij} where α_{ij}^k is the angle opposite to edge e_{ij} in triangle t_{ijk} (and similarly for α_{ij}^l). The diagonal Δ_{ii} holds the negative sum of the off-diagonal entries in that row. (We ignore here scaling factors which arise in various applications.) Δ is symmetric and positive definite, given appropriate boundary conditions. These are typically given as desired values (Dirichlet data) or cross boundary derivatives (Neumann data). Non-Delaunay edges in the mesh give rise to negative cotan weights, which leads to a loss of the local (discrete) maximum principle. One symptom of this is the occurrance of flipped triangles when computing texture maps. This issue is entirely avoided if we use the iLB operator, *i.e.*, Δ_{ij} depends on the iDT. At vertices of valence 1 or 2 (which can occur in the iDT) injectivity of a discrete conformal mapping is lost though, since the single incident triangle to a vertex of valence 1 is collapsed to a point while in the case of valence 2 the two incident triangles are collapsed to a line. In effect the Jacobian of the mapping from the surface to the texture plane can become singular, but its determinant *cannot* become negative. While this induces (unavoidable) local distortion it does not cause any problems in the texture mapping stage itself.

The *number* of non-zero entries in the matrix is the same for both eLB and iLB, thus having no impact on the operations count in a single application of the matrix to a vector.

Table 1 gives some statistics of the edge flipping algorithm for a number of representative meshes. Comparing the total number of flips with the number of iDT edges which cross exactly two original triangles ("simple") we see that this simplest case is by far the most common. The other extreme is the iDT edge consisting of the most segments, *i.e.*, crossing the longest ("lgst.") chain of original triangles. These are generally also short with the notable exception of the Camel where the longest chain of segments is length 22. An aggregate measure is the total number of crossings generated for all iDT edges: Max Planck (7167), Bunny (2434), Camel (25043), Horse (4027), and Feline (13317). Here the Camel stands out with more than half as many crossings as original vertices. More typical are cases such as Horse and Feline.

Figure 6 shows a histogram of coefficients for the iLB versus eLB operator for the Hygeia model (other models have similar histograms). Obviously there are no negative coefficients anymore,

Model	V	flips	simple	lgst.	κ_i/κ_e
Cat hd.	131	45	40	3	0.8114
Bny hd.	741	380	275	6	0.6139
Bty. Frk.	1042	560	364	12	0.1841
Hygeia	8268	4017	2861	6	0.1053
Planck	25445	6417	5584	5	0.7746
Bunny	34834	2365	2283	4	0.0758
Camel	40240	17074	12618	22	0.7218
Horse	48485	3581	3127	7	0.6734
Feline	49864	12178	10767	7	0.5746

Table 1: *Statistics for some representative meshes. Number of: vertices; edge flips; flipped edges crossing only two original triangles; maximal number of segments in an iDT edge; and condition number improvement as ratio for iLB and eLB). All runs were well under 1 second on a 2GHz Athlon.*

but we also see a noticeable decrease in the number of large coefficients.

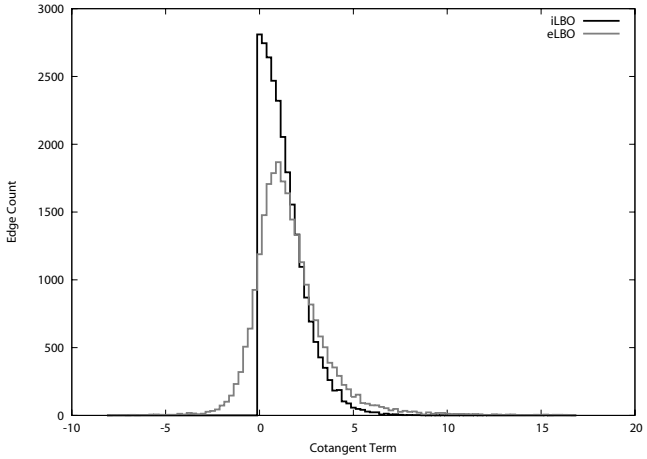

Figure 6: *Histogram of coefficients in the eLB versus iLB operator for the Hygeia model.*

For numerical considerations the most relevant figure of merit is the condition number of the Laplace-Beltrami operator, which we evaluate by considering the ratio of iLB (κ_i) to eLB condition number (κ_e). Generally we find an improvement between 20% and 40%, due to the largest eigenvalue being decreased (as predicted by theory [Rippa 1990]). Notable outliers are the "Beautiful Freak" dataset [Desbrun et al. 2002], which was specifically engineered to be challenging (80% improvement) and the Bunny (over 90% improvement).

The numerical improvements in the iLB operator over the eLB are also noticeable in the quality of discrete harmonic parameterizations [Desbrun et al. 2002; Lévy et al. 2002; Mercat 2001; Gu and Yau 2003]. In this case the overlay graph is required to properly interpolate the induced texture mapping functions. Figure 7 shows the original triangulation (left column) and iDT (right column) for the "Beautiful Freak" dataset when mapped to the texture plane using Dirichlet (disk) boundary conditions. The resulting checker board pattern when mapped onto the surface is shown below. (Note that because of the Dirichlet boundary conditions we do not expect the resulting texture to be conformal.) The distortion in each triangle can be visualized by plotting the ratio of largest to smallest singular value of the Jacobian. In the case of conformal parameterizations this ratio can be as low as unity. Figure 8 compares the results for original triangulation (left column) and iDT (right column) with Dirichlet (top) boundary conditions to the disk (see Figure 7) and

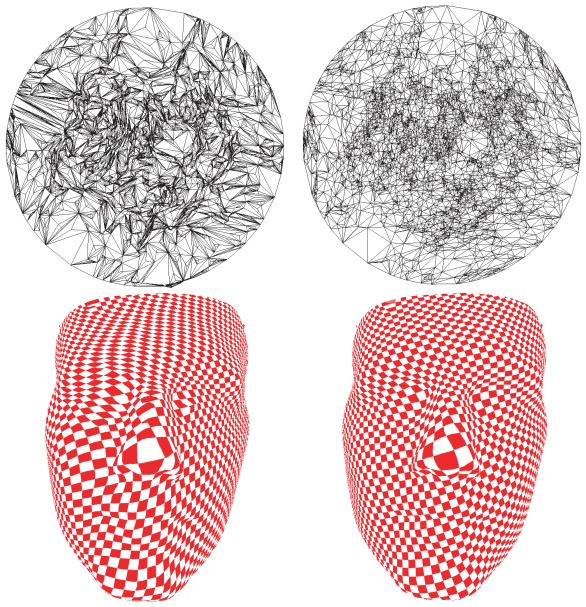

Figure 7: *Original (left column) and iDT (right column) of the Beautiful Freak dataset (Dirichlet boundary conditions). Texture plane image and resulting checker board mapping onto the surface.*

natural boundaries (bottom). As expected the distortion is overall lower for natural boundaries, but even in that case there is still a marked difference between original triangulation and iDT.

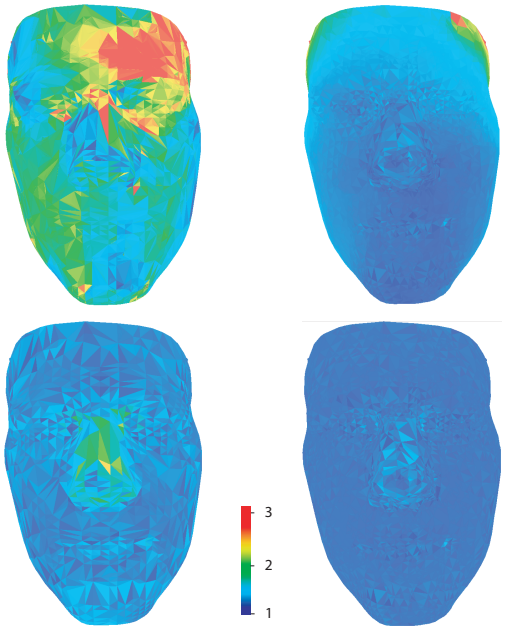

Figure 8: *Comparison of distortion (ratio of larger to smaller singular value of the Jacobian) between original triangulation (left column) and iDT (right column) for Dirichlet (top row) and natural (bottom row) boundary conditions.*

When the iLB operator is used to compute a discrete harmonic parameterization one can interpolate the resulting (u,v) parametric assignments at the vertices with respect to the original triangulation or with respect to the iDT. The latter requires the overlay graph. In Figure 9 the distortion error between the two alternatives is visualized. We can clearly see that the error is lower when the parameter-

ization is interpolated with respect to the overlay graph, justifying the extra expense of computing the overlay graph if the lowest possible distortion is to be achieved.

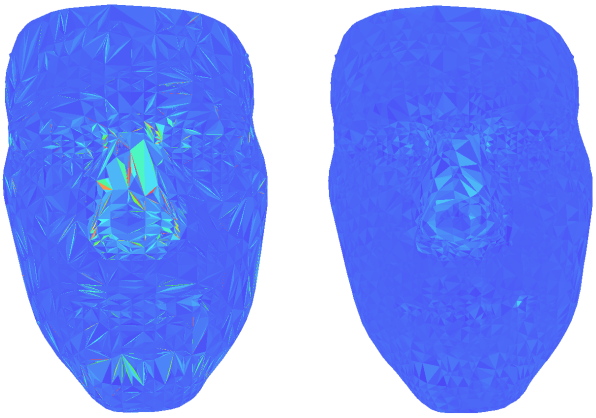

Figure 9: *Comparison of the distortion error when using the iLB operator to compute harmonic parameterizations (here with natural boundary conditions). On the left the parametric assignments at vertices are interpolated with respect to the original triangulation while on the right the iDT is used.*

The lowering of distortion is also noticeable in other parameterization applications. Kharevych *et al.* [2006] used circle patterns to compute discrete conformal mappings for embedded meshes. Briefly, in this approach discrete conformal mappings for triangle meshes (of arbitrary topology) are computed via triangle circumcircles and the angles they make with one another at shared edges. A requirement of the underlying theory is that all these angles be in $[0, \pi]$. While this can be enforced through clipping illegal values to the nearest legal value, a better approach is to change the combinatorics of the triangulation so that it is iDT. Figure 10 compares the distortion for two datasets when using the original triangulation (left column) and iDT (right column).

4 Conclusion

Given a triangle mesh annotated with lengths for every edge (assuming these lengths satisfy the triangle inequality) an intrinsic Delaunay triangulation is well defined and can be found with the help of a simple and low cost edge flipping algorithm. If explicit representations of the iDT edge crossings are required, an overlay graph can be incrementally maintained, using only combinatorial considerations. For applications which require a discrete Laplace-Beltrami operator (*e.g.*, denoising, parameterization, editing, simulation), defining it over the iDT has numerical advantages from improved condition numbers to lower error in the computations, and a reduced need for special case handling.

Clearly an (embedded) mesh which is intrinsically Delaunay to begin with is most desirable. It would be interesting to explore this as a constraint in remeshing algorithms, be they for static, or dynamically deforming, surfaces. Finally, we mention the challenge of making the algorithm robust. While we have not experienced any problems in our experiments, correct execution (and termination) of the implementation of the algorithm will likely require more than simple floating point arithmetic.

Acknowledgments This work was supported in part by NSF (CCF-0528101), DFG Research Center MATHEON "Mathematics for Key Technologies," DFG research unit "Polyhedral Surfaces," DOE (W-7405-ENG-48/B341492), the Alexander von Humboldt Stiftung, the Caltech Center for the Mathematics of Information, nVidia, Autodesk, and Pixar Animation Studios. Special thanks to Mathieu Desbrun, Yiying Tong, Liliya Kharevych, Herbert Edelsbrunner, and Cici Koenig.

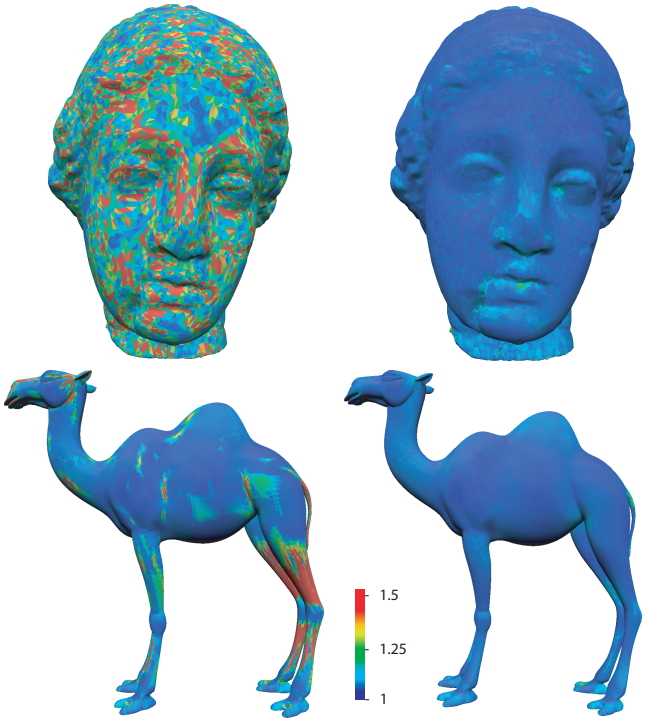

Figure 10: *Two examples (Hygeia, genus zero; cut camel with free boundaries) comparing distortion for original triangulation (left column) and iDT (right column). (Used with permission from [Kharevych et al. 2006].)*

References

BOBENKO, A. I., AND SPRINGBORN, B. A., 2005. A discrete Laplace-Beltrami operator for simplicial surfaces. Preprint http://www.arxiv.org/math/0503219.

BOTSCH, M., AND KOBBELT, L. 2004. An Intuitive Framework for Real-Time Freeform Modeling. *ACM Trans. on Graphics 23*, 3, 630–634.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. 1999. Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow. In *Comp. Graphics (Proc. of SIGGRAPH)*, 317–324.

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic Parameterizations of Surface Meshes. *Comp. Graph. Forum (Proc. of EG) 21*, 3, 209–218.

GRINSPUN, E., HIRANI, A., DESBRUN, M., AND SCHRÖDER, P. 2003. Discrete Shells. In *Symp. on Comp. Anim.*, 62–67.

GRINSPUN, E., SCHRÖDER, P., AND DESBRUN, M., Eds. 2005. *Discrete Differential Geometry*. Course Notes. ACM SIGGRAPH.

GU, X., AND YAU, S.-T. 2003. Global Conformal Surface Parameterization. In *Symp. on Geom. Proc.*, 127–137.

INDERMITTE, C., LIEBLING, T. M., TROYANOV, M., AND CLÉMENÇON, H. 2001. Voronoi Diagrams on Piecewise Flat Surfaces and an Application to Biological Growth. *Th. Comp. Sci. 263*, 263–274.

KHAREVYCH, L., SPRINGBORN, B., AND SCHRÖDER, P. 2006. Discrete Conformal Mappings via Circle Patterns. *ACM Trans. on Graphics 25*, 2, 412–438.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *ACM Trans. on Graphics 21*, 3, 362–371.

MERCAT, C. 2001. Discrete Riemann Surfaces and the Ising Model. *Comm. in Math. Physics 218*, 1, 177–216.

PINKALL, U., AND POLTHIER, K. 1993. Computing Discrete Minimal Surfaces and Their Conjugates. *Exp. Math. 2*, 1, 15–36.

POLTHIER, K., AND SCHMIES, M. 2002. Straightest Geodesics on Polyhedral Surfaces. In *Mathematical Visualization*, H. C. Hege and K. Polthier, Eds. Springer Verlag, 135–152.

RIPPA, S. 1990. Minimal Roughness Property of the Delaunay Triangulation. *CAGD 7*, 6, 489–497.

SHEWCHUK, J. 2002. What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. In *Proc. of Intl. Meshing Roundtable*, 115–126.