Please cite as: T. Winkler, K. Hormann, and C. Gotsman. Mesh massage: A versatile mesh optimization framework. *The Visual Computer*, 24(7–9):775–785, July 2008. Best Papers of CGI 2008.

# Mesh Massage: A Versatile Mesh Optimization Framework

Tim Winkler Kai Hormann Craig Gotsman

Department of Informatics
Clausthal University of Technology

Computer Science Department
Technion

**Abstract**

We present a general framework for post-processing and optimizing surface meshes with respect to various target criteria. On the one hand, the framework allows us to control the shapes of the mesh triangles by applying simple averaging operations, on the other we can control the Hausdorff distance to some reference geometry by minimizing a quadratic energy. Due to the simplicity of this setup, the framework is efficient and easy to implement. Yet it constitutes an effective and versatile tool with a variety of possible applications. In particular, we use it to reduce the texture distortion in animated mesh sequences, to improve the results of cross-parameterizations, and to minimize the distance between meshes and their remeshes.

## 1 Introduction

Over the last decade, triangle meshes have become the most common representation of 3D surfaces in computer graphics and geometric modelling, and there exists an abundance of geometry processing tools for creating, editing, and modifying them. In many applications, however, the result is not optimal and can further be improved by post-processing the mesh. For example, Hoppe et al. [12] optimize meshes from surface reconstruction, Ohtake and Belyaev [20] improve triangulated iso-surfaces, and Surazhsky and Gotsman [27] optimize the regularity of a mesh.

The general idea behind mesh optimization is the following: a triangle mesh $M$ is a piecewise linear approximation of some smooth surface $S$, but there are of course many other meshes that may represent $S$ equally well. Therefore, we can try to modify $M$ such that it is still a good approximation of $S$, but is better than $M$ in some way. The optimization methods then typically differ in the optimization criterion used and the type of admissible mesh modifications. For example, the main goal of mesh simplification algorithms (see [11] for a survey) is to reduce the number of triangles while preserving the overall shape. These methods modify the number and positions of the mesh vertices as well as the way in which they are triangulated. Other methods minimize the overall discrete curvature of a mesh [30, 6] and are more restrictive as they change only the connectivity without moving the vertices. In contrast, mesh smoothing algorithms that eliminate noise [28, 13, 8] usually optimize only the vertex positions while keeping the triangulation of the mesh fixed.

Our approach falls in the last category in the sense that we also modify only the vertex positions and do not change the connectivity of the mesh. However, we do not have a single optimization criterion, but rather provide a general framework that can be adapted to various settings. The method is somewhat similar to those of Nealen et al. [19] and Liu et al. [16] as it also uses two competing energies, one to control the shape of the triangles (Section 2.1) and another to minimize the distance to some reference geometry (Section 2.2). However, our approach can be used for a broader range of possible applications as it differs from [19, 16] in the following way: First, we use mean value coordinates [9] instead of Laplacian coordinates to control the local shape of the mesh triangles, which allows us to also use the triangle shapes from a different reference mesh as target templates (Sections 3.1 and 3.2). Second, we do not consider distances between the original and modified vertex positions, but more generally minimize the approximate Hausdorff distance between the optimized mesh and some reference geometry. The latter can either be the mesh before optimization or something completely different (Section 3.3).

### 1.1 Contributions

We present a versatile yet simple framework that can be used to post-process the results of current geometry processing algorithms and optimize them with respect to various criteria. At the heart of our framework are two mechanisms, one to control the shapes of the mesh triangles by enforcing convex combinations with certain weights, another to constrain the optimized mesh to be geometrically close to a given reference geometry. We can describe the optimization procedure as a variational problem and determine the vertices of the optimized mesh by iteratively solving a sparse linear system. The details of our method are described in Section 2 and in Section 3 we show the effectiveness of this framework by applying it to several different settings.

*Animated Mesh Sequences:* Suppose we have a set of meshes that constitute an animation sequence and that all meshes have the same connectivity but different geometry. In this setting it is natural to use a single set of texture coordinates for all meshes. If the sequence was generated by hand, e.g. by animating a character, this works out nicely. But if the sequence was extracted from the acquisition data of a real world dynamic scene [2], then it may happen that the relative positions of the vertices change from frame to frame, making the texture bounce around. We can use our framework to drastically reduce this effect by aligning the relative vertex positions in all meshes to those in the first frame (see Section 3.1).

*Cross-Parameterizations:* Kraevoy and Sheffer [14] and Schreiner et al. [24] described how to construct a mapping from one mesh to another. This cross-parameterization can be used, e.g. to transfer attributes and to morph between the two meshes. Both applications rely on the mapping to have as low distortion as possible and we found that our framework can improve the initial parameterizations considerably (see Section 3.2).

*Remeshing:* Another class of algorithms that take a given mesh $M$ and generate another mesh $N$ that represents the same surface as $M$, are remeshing algorithms (see [1] for a survey). Unlike mesh optimization methods, these algorithms do not modify $M$ to get $N$, but rather compute the new mesh from scratch. Usually, the vertices of the remesh $N$ lie on the triangles of the initial mesh $M$, thus leading to a rather large Hausdorff distance between both meshes. With our framework, we can roughly halve this distance (see Section 3.3).

## 2 Mesh Optimization Framework

In this section we explain the different components of our framework for optimizing a mesh $M$ with vertices $v_1, \ldots, v_m$. The mesh can have arbitrary topology and be with or without boundaries. We first describe how to control the triangle shapes by convex combinations, then discuss how to approximate the Hausdorff distance between $M$ and a reference mesh $N$, and finally explain how to combine both components.

### 2.1 Controlling the Triangle Shape

The basic idea for controlling the shapes of the mesh triangles is to iteratively apply averaging operations to its vertices. Assume that we have for every vertex $v_i$ and each of its neighbours $v_j$ a weight $\lambda_{ij}$, then the optimized position $\tilde{v}_i$ of the vertex $v_i$ is given by

$$\tilde{v}_i = \sum_{j \in N_i} \lambda_{ij} v_j, \tag{1}$$

where $N_i$ is the index set of all neighbours of $v_i$. More precisely, we require that all weights $\lambda_{ij}$ are positive and sum to unity,

$$\sum_{j \in N_i} \lambda_{ij} = 1, \qquad \text{for all } i,$$

so that (1) describes a *convex combination* and hence $\tilde{v}_i$ always lies inside the convex hull of its neighbours $v_j$, $j \in N_i$.

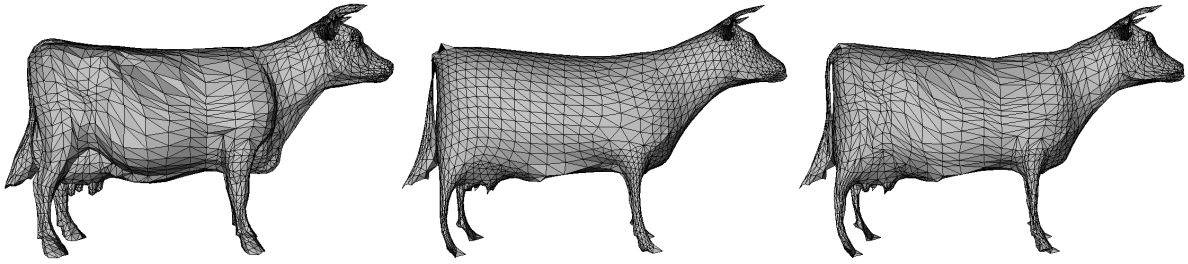

Figure 1: Applying averaging operations to a mesh (left) with uniform (centre) and mean value weights (right).

For example, if all weights $\lambda_{ij}$, $j \in N_i$ are identical, then $\tilde{v}_i$ will be located at the barycentre of its neighbours and iteratively applying (1) to all vertices of $M$ will lead to a uniform distribution of points and tends to create equilateral triangles. Instead, if we let $\lambda_{ij}$ be the mean value coordinates [9] of $v_i$ with respect to its neighbours $v_j$, then the shapes of the triangles are nicely preserved (see Figure 1).

This effect is well known from mesh parameterization (see [10] for a survey) and stems from the fact that mean value coordinates are a generalization of barycentric coordinates to arbitrary polygons. A similar effect can be achieved with other barycentric coordinates, but the discrete harmonic coordinates [22, 7] may be negative (thus violating the convex hull property) and the Wachspress coordinates [31, 17] are not necessarily well-defined.

Inspired by the deformation transfer method of Sumner and Popović [25], we realized that instead of computing the mean value weights from $M$, we can also compute the weights $\lambda_{ij}$ from a second mesh $N$ and thus "copy-and-paste" the shapes of the triangles in $N$ to the triangles in $M$. Of course, this requires $N$ to have the same connectivity as $M$, but this setting is less restrictive than it seems and can be found in several applications, two of which we discuss in Sections 3.1 and 3.2.

Taking averages as described above can actually be seen as repeatedly applying smoothing with either the standard Laplacian (uniform weights) or the discrete Laplace-Beltrami operator (mean value or discrete harmonic weights) [5]. Unfortunately, this kind of smoothing is well-known to have an undesired shrinking effect on the shape of the mesh (see Figure 1), so it cannot be used aggressively unless a complementary mechanism is used to combat the shrinking. We show how to do this in Section 2.3.

### 2.2 Controlling the Distance

Besides controlling the triangle shapes, our framework should also allow to control the distance between the optimized mesh $M$ and some reference geometry. In the following we focus on the case that this reference geometry is a second triangle mesh $N$ with vertices $w_1, \ldots, w_n$, but in principle it could be of any kind, e.g. a point cloud, an implicit surface, or a parametric surface. Ideally, we would like to measure the Hausdorff distance between the two meshes $M$ and $N$, but as that is rather costly to compute, we will use the following approximation instead.

Let $A$ and $B$ be two non-empty compact sets. The (minimum) distance from some $a \in A$ to the set $B$ is defined as

$$d(a, B) = \min_{b \in B} \|a - b\|$$

and the one-sided Hausdorff distance $d_{\mathrm{h}}(A, B)$ between $A$ and $B$ is the maximum of all distances,

$$d_{\mathrm{h}}(A, B) = \max_{a \in A} d(a, B). \tag{2}$$

The (symmetric) Hausdorff distance $d_{\mathrm{H}}(A, B)$ between $A$ and $B$ is finally defined as

$$d_{\mathrm{H}}(A, B) = \max\{d_{\mathrm{h}}(A, B), d_{\mathrm{h}}(B, A)\}.$$

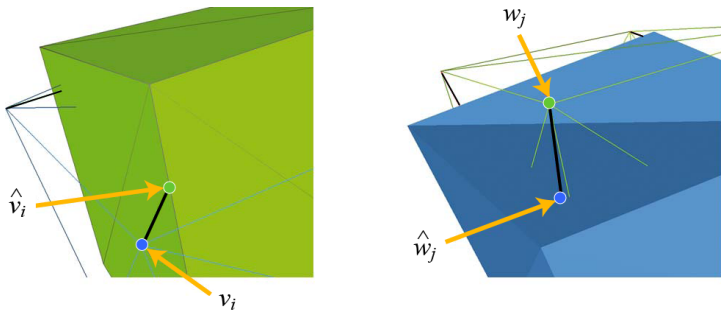


Figure 2: Distances between vertices $v_i$ of $M$ (blue) and corresponding points $\hat{v}_i$ in $N$ (green) and vice versa.

If $A$ and $B$ are two meshes $M$ and $N$, then the Hausdorff distance is usually simplified as follows. First, we determine for any point $v$ in $M$ (where $v$ is not necessarily a vertex, but more generally inside one of $M$'s triangles) the corresponding closest point $\hat{v}$ in $N$ (which is again inside one of $N$'s triangles) so that

$$d(v, N) = \|v - \hat{v}\|.$$

In Section 2.2.1 we describe in detail how to find and maintain these corresponding points. Secondly, we do not consider all possible points of $M$ in (2), but only a finite set (usually the vertices $v_i$ of $M$) and replace the max-norm by the 2-norm,

$$d_{\mathrm{h}}(M, N) \approx \left( \sum_{i=1}^{m} \|v_i - \hat{v}_i\|^2 \right)^{1/2}$$

and similarly for the Hausdorff distance, so that minimizing $d_{\mathrm{H}}(M, N)$ is then simplified to minimizing

$$d_{\mathrm{H}}(M, N)^2 \approx \sum_{i=1}^{m} \|v_i - \hat{v}_i\|^2 + \sum_{j=1}^{n} \|w_j - \hat{w}_j\|^2, \tag{3}$$

where $\hat{w}_j$ are the points in $M$ that correspond to the vertices $w_j$ of $N$ (see Figure 2).

Although simple to minimize, this approximation of the Hausdorff distance suffers from the fact that the max-norm is replaced by the 2-norm which allows for rather large maximum distances if they are counterbalanced by a lot of small distances. Therefore, we improve the approximation in (3) using a *weighted* 2-norm,

$$\sum_{i=1}^{m} \alpha_i \|v_i - \hat{v}_i\|^2 + \sum_{j=1}^{n} \beta_j \|w_j - \hat{w}_j\|^2. \tag{4}$$

A theorem by Motzkin and Walsh [18] states that there exists a set of weights $\alpha_i$ and $\beta_j$ such that minimizing (4) is equivalent to minimizing the maximum distance

$$\max \left\{ \max_{i=1,\ldots,m} \|v_i - \hat{v}_i\|, \max_{j=1,\ldots,n} \|w_j - \hat{w}_j\| \right\}$$

which in turn is a much better approximation of the Hausdorff distance $d_{\mathrm{H}}(M, N)$ than (3) and can further be improved by considering not only the vertices $v_i$ of $M$ and $w_j$ of $N$ but also additional sample points on the edges and inside the triangles of both meshes.

It remains to find the optimal weights $\alpha_i$ and $\beta_j$, but this can be done iteratively using Lawson's algorithm [15]. This algorithm suggests to start with initial weights $\alpha_i^{(0)} = 1$ and to update them after the $k$-th iteration by the rule

$$\alpha_i^{(k+1)} = \alpha_i^{(k)} \|v_i^{(k)} - \hat{v}_i^{(k)}\|, \tag{5}$$

and likewise for the weights $\beta_j$. The idea behind this strategy is that vertices with a large distance contribute more to the weighted 2-norm (4) in the next iteration than those with a small distance. Although the theoretical convergence of this algorithm to the optimal weights requires infinitely many iterations, we observed that in practice a few iterations (5 to 10) usually suffice to get very close to the optimum.

Finally note that our weighted 2-norm, like the approximation in (3), is quadratic in the vertices $v_i$ of $M$, because $\hat{v}_i$ and $w_j$ are points in $N$ and thus fixed and any point $\hat{w}_j$ in $M$ can be written as a barycentric combination of the three vertices $v_i$ of $M$ that constitute the triangle which contains $\hat{w}_j$. Thus, minimizing (4) amounts to solving a sparse linear system of normal equations. An alternative but less efficient approximation of the Hausdorff distance was presented in [3].

### 2.2.1 Finding the Corresponding Points

In our framework, the initial corresponding points $\hat{v}_i$ in $N$ for the vertices $v_i$ of $M$ (and similarly the $\hat{w}_j$ for $w_j$) are set to the closest vertex $w_j$ of $N$. This requires a global search over all vertices of $N$, but can be done efficiently by using an appropriate spatial data structure, e.g. a $k$-$d$-tree. Note that this simple strategy requires the mesh $M$ and the reference geometry $N$ to be similar in shape and aligned, which is the case in all the examples that we discuss in Section 3; see [23] and the references therein for a recent overview of alignment algorithms and their performances.

After this initialization, we perform a local search in the triangles around the initial corresponding point $\hat{v}_i$ to find the point closest to $v_i$ among all points in these triangles (see [21] for details) and use it as the correct corresponding point. This closest point usually lies inside one of $N$'s triangles, but can also fall on a vertex or an edge of $N$ (see Figure 2). The same local search procedure is carried out after each iteration in order to always keep the distances between $v_i$ and $\hat{v}_i$ minimal.

## 2.3 Combined Optimization

When controlling the triangle shape as described in Section 2.1, what we ideally want (instead of iteratively applying the averaging steps) is to place the vertices $v_i$ of $M$ such that the convex combinations (1) are satisfied for all $i$ with $\tilde{v}_i$ replaced by $v_i$, which is somehow equivalent to doing infinitely many averaging iterations. This amounts to solving the linear system

$$A\boldsymbol{v} = \boldsymbol{0} \tag{6}$$

where $\boldsymbol{v}$ is the vector of all vertices $v_i$ and $A$ is the sparse Laplacian matrix with elements $A_{ii} = 1$, $A_{ij} = -\lambda_{ij}$ if $v_i$ and $v_j$ are neighbours, and $A_{ij} = 0$ otherwise. The problem is that this system is solved only if all $v_i$ are set to a constant point, which is a rather degenerate solution, but conforms with the remark on the shrinking effect of this kind of smoothing.

In order to get a reasonable solution one can constrain the system (6) by fixing the positions of a few vertices (e.g. the vertices at the boundary of a mesh), but this will result in a kind of minimal surface spanned between these constrained vertices [22]. Again, this is not exactly what we want and we rather choose to constrain it with the Hausdorff distance from Section 2.2. Similar to Equation (6), the vertex positions that minimize the approximate Hausdorff distance (4) can also be found by solving the linear system, namely the normal equation

$$B\boldsymbol{v} = \boldsymbol{c},$$

where $B$ is a sparse, symmetric, and positive definite matrix and the right hand side $\boldsymbol{c}$ depends on the fixed vertices $\hat{v}_i$ and $w_j$. In our framework, we simply take a weighted average of both systems and solve

$$\big(\mu A + (1-\mu)B\big)\boldsymbol{v} = (1-\mu)\boldsymbol{c} \tag{7}$$

with $\mu \in [0, 1)$ in order to get the new vertex positions of $M$. In this approach, $\mu$ is a tradeoff-parameter between the distance and the shape control. A small value of $\mu$ emphasizes the distance term and leads

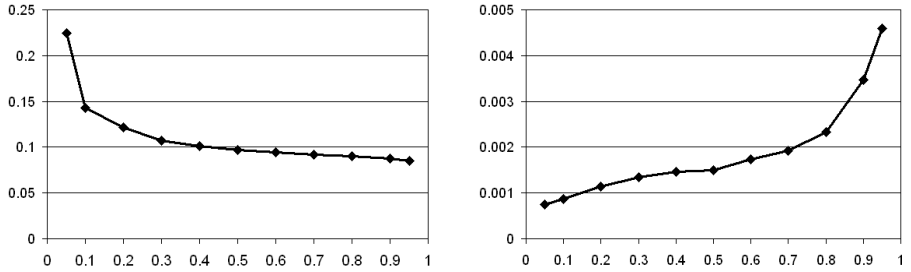


Figure 3: Influence of $\mu$ on the triangle shape (left) and the Hausdorff distance (right) for the example in Figure 9.

to very small Hausdorff distances, whereas a larger value of $\mu$ gives a better control on the triangle shape at the cost of larger distances (see Figure 3). In practice we found that setting

$$\mu = \frac{\|B\|}{3\|A\| + \|B\|}$$

with the Frobenius matrix norm gave satisfactory results in all our examples. Moreover, we found that performing 10 to 15 iterations of solving (7), updating the weights $\alpha_i$ and $\beta_j$ as in (5) and the positions of the corresponding vertices $\hat{v}_i$ and $\hat{w}_j$ (see Section 2.2.1) after each iteration, is usually enough to give nearly optimal results, i.e. neither the distance nor the triangle shapes are further optimized significantly by additional iterations.

Instead of solving (7), we could also minimize the Laplacian system (6) and the approximate Hausdorff distance (4) together in a least squares sense, as done in [19, 16], but the resulting normal equation is less sparse than our system (7) and takes longer to solve.

# 3 Applications

## 3.1 Texture Transfer

Suppose we have a mesh $M_0$ that approximates some surface $S$. If we now move the vertices of $M_0$ slightly in the local tangent planes, then we get a mesh $M_1$ that also approximates $S$ and looks almost identical to $M_0$ if rendered with flat or smooth shading. However, if we texture both meshes and use the same texture coordinates for corresponding vertices, than we will clearly see the difference as the shapes of the triangles in $M_1$ are no longer similar to those in $M_0$, thus leading to texture distortion.

These kinds of artifacts occur, e.g. when a compatibly meshed sequence of meshes is extracted from some unstructured acquisition data of a dynamic scene. Such an approach has been described by Anuar and Guskov [2]. They start with an initial template mesh that is then propagated through the frames of the animation, based on an adaptation of the Bayesian multi-scale differential optical flow algorithm. Since the flow is invariant to motion in the tangent plane, the artifacts described above tend to occur and as a result the texture seems to wobble over the animated mesh during the sequence.

The top row in Figure 4 shows the texture distortion for several meshes $M_i$ from the dancing man sequence in [2] with blue signifying low and red indicating high distortion. The distortion is measured with respect to the first mesh $M_0$ of the sequence and using the symmetric maximum shear,

$$\max\left\{\sigma_1 + \frac{1}{\sigma_1}, \sigma_2 + \frac{1}{\sigma_2}\right\} - 2, \tag{8}$$

as a distortion measure. More precisely we consider for each triangle in $M_0$ the linear map to the corresponding triangle in $M_i$ and compute the singular values $\sigma_1$ and $\sigma_2$ of this mapping, which capture the distortion of this triangle in the principal directions. In order to penalize over- and undersampling in the same way, we symmetrize these values by adding their inverse, so that shrinking by a factor of $1/2$ is considered equally bad as expanding by a factor of 2, and finally take the worst of both values.

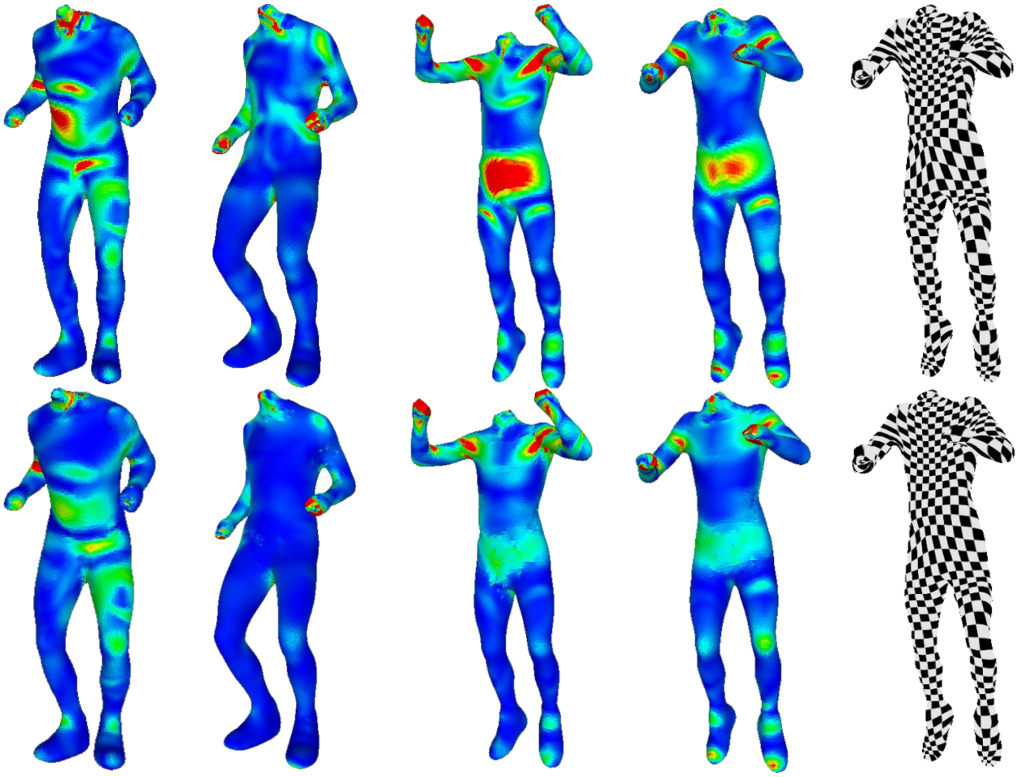

Figure 4: Texture distortion for several meshes selected from the animation sequence of a dancing man and textured version of the last mesh. Top: original meshes, bottom: optimized meshes.

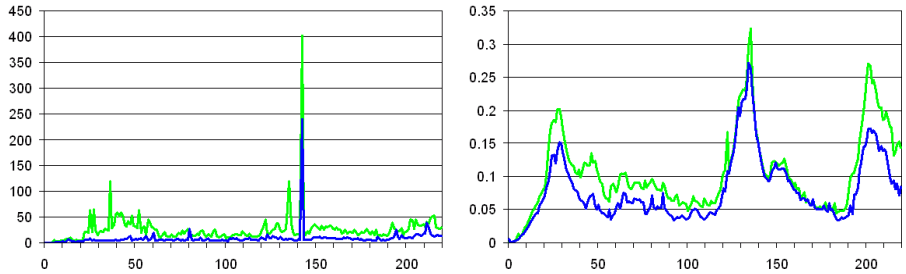


Figure 5: Maximum (left) and average (right) distortion for all meshes in the original (green) and the optimized (blue) dancing man sequence.

We can apply our framework to optimize the meshes $M_i$ in the sequence by taking $M_i$ itself as the reference geometry and using $M_0$ as the template for the triangle shapes, i.e. we use the mean value weights $\lambda_{ij}$ that were computed from $M_0$ to optimize $M_i$. As a result, the vertex positions in the flat surface regions are corrected and the texture is fixed throughout the sequence. This results in improved visual quality and is shown in the bottom row of Figure 4. The rightmost column shows the last mesh again, but this time textured to emphasize the influence of the distortion to the texture. The original sequence as well as the optimized one can be seen in the additional videos.

Figure 5 further shows the maximum and the average distortion for all meshes in the sequence and confirms that our method is able to reduce both for all meshes. Interestingly, the peaks in the average distortion plots correspond to the jumps of the dancing man, where the triangle distortion tends to increase due to the extreme global shape deformations in the mesh.

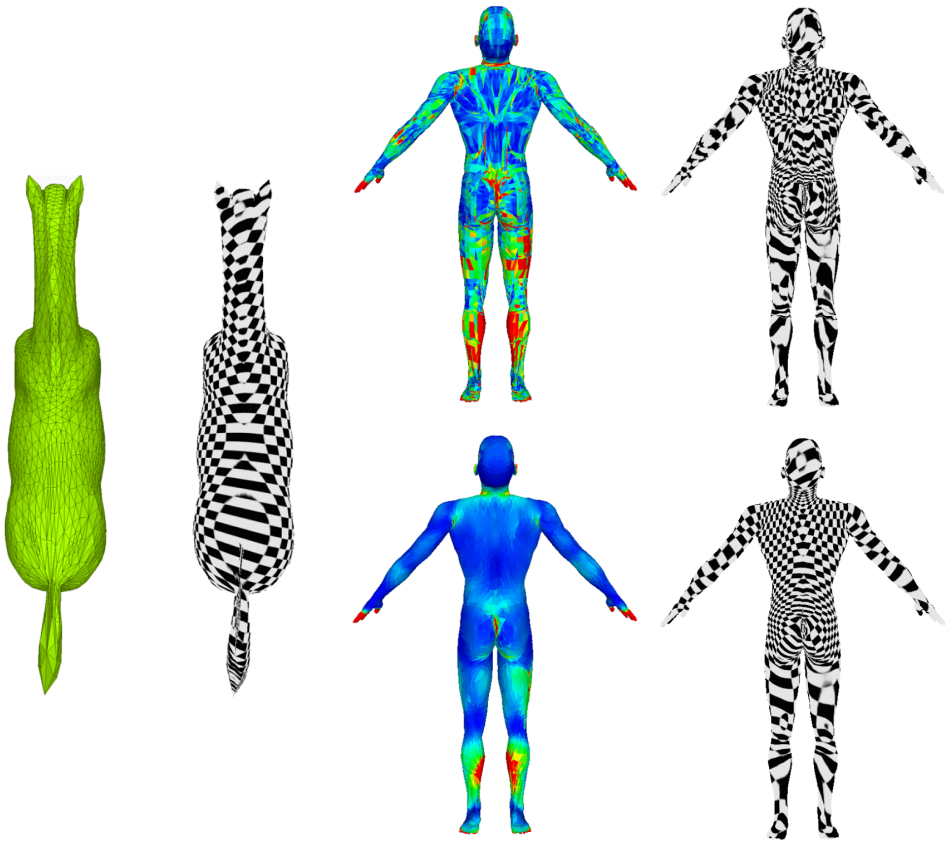

Figure 6: Left: shaded and textured version of the first frame from the horse-to-man sequence that serves as a reference frame for the triangle shape. Right: distortion of the cross-parameterization between the first and the last frame of the sequence before (top) and after (bottom) optimization.

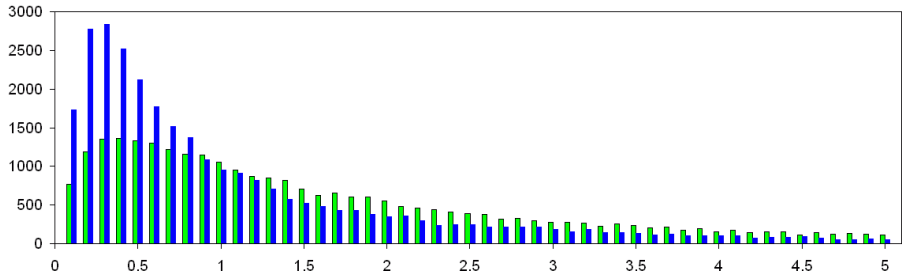


Figure 7: Distribution of the distortion per triangle before (green) and after (blue) the optimization.

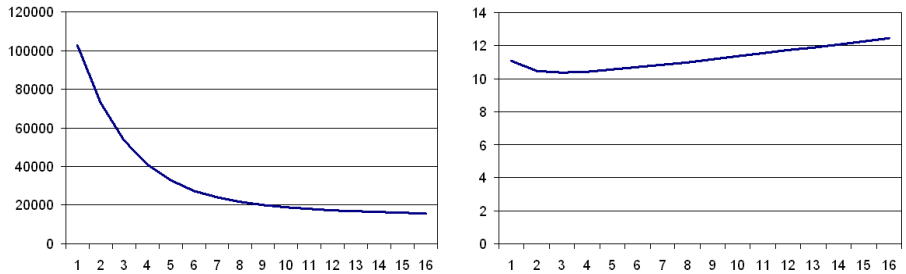


Figure 8: Maximum (left) and average (right) distortion over 16 iterations of the optimization process.

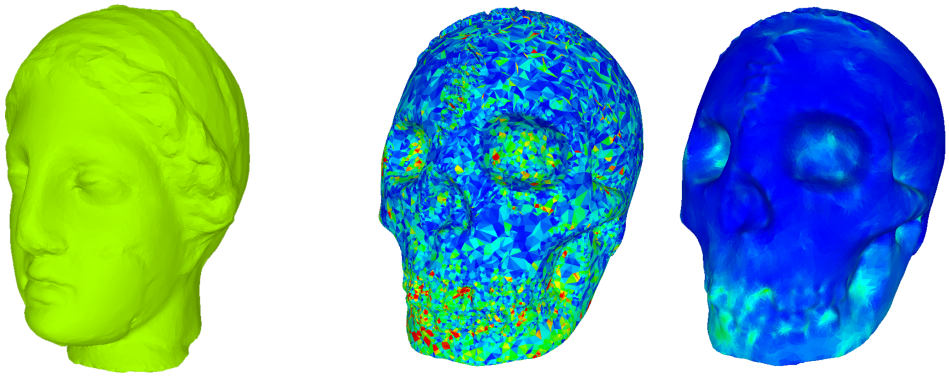

Figure 9: Reference mesh (left) and target model without (centre) and with optimization (right).

### 3.2 Cross-Parameterization

We can also apply our framework to reduce the distortion of cross-parameterizations, i.e. mappings between two meshes with different shapes but compatible connectivity. In particular we used our method to optimize the horse-to-man morphing sequence and the cross-parameterization between the venus head and the skull model, where both examples were generated by the algorithm of Kraevoy and Sheffer [14].

The setup for our framework is similar to the one used in the previous section, i.e. the horse and the venus head are used as reference meshes for the triangle shapes, while the reference geometry is provided by the same mesh that is also optimized (the mesh of the man and the skull model, respectively).

Figure 6 visualizes the distortion of the cross-parameterization between the first and the last mesh of the horse-to-man morphing sequence for the original (top) and the optimized mesh (bottom) in two ways: by colour-coding the distortion (8) per triangle and by showing how the parameterization maps a regular texture from the first to the last mesh. Both plots show that our framework successfully reduces the distortion, which is also confirmed by the clear shift towards small values in the histograms of the distortion distribution shown in Figure 7. Figure 8 further shows how the maximum distortion is reduced during the iterations of our optimization process. Note that this comes at the price of a slight increase in average distortion.

Figure 9 finally shows the distortion of the parameterization between the venus head and the skull model before and after our optimization.

### 3.3 Remeshing Revised

Remeshing is an essential part of the geometry processing toolbox. Although the output generated by modern acquisition tools like 3D scanners captures 3D objects well in terms of geometry, the quality of the generated mesh is often bad with respect to the triangle shapes or the connectivity. To compensate for these disadvantages, many algorithms have been proposed to generate for a given source mesh $M_0$ a remesh $M_1$ that has the same shape but is superior, e.g. by having a more uniform or curvature-adaptive sampling density, better triangle shapes, or more regular vertex connectivities.

In almost all remeshing methods, the vertices of the remesh $M_1$ lie on the triangles of the source mesh $M_0$, which is far from optimal regarding the Hausdorff distance between both meshes. An exception is the approach of Surazhsky et al. [26] that places the vertices of $M_1$ on interpolating PN-triangles over $M_0$. However, in both cases we found that we can use our framework to reduce the Hausdorff distance by 50% and more.

Note that this setting does not require $M_0$ and $M_1$ to have compatible connectivity because we do not want to "copy-and-paste" the triangle shapes from the source mesh to the remesh. Instead, $M_0$ serves only as the reference geometry and $M_1$ itself as its triangle shape reference, i.e. we compute the mean value weights $\lambda_{ij}$ from the same mesh that we also optimize.

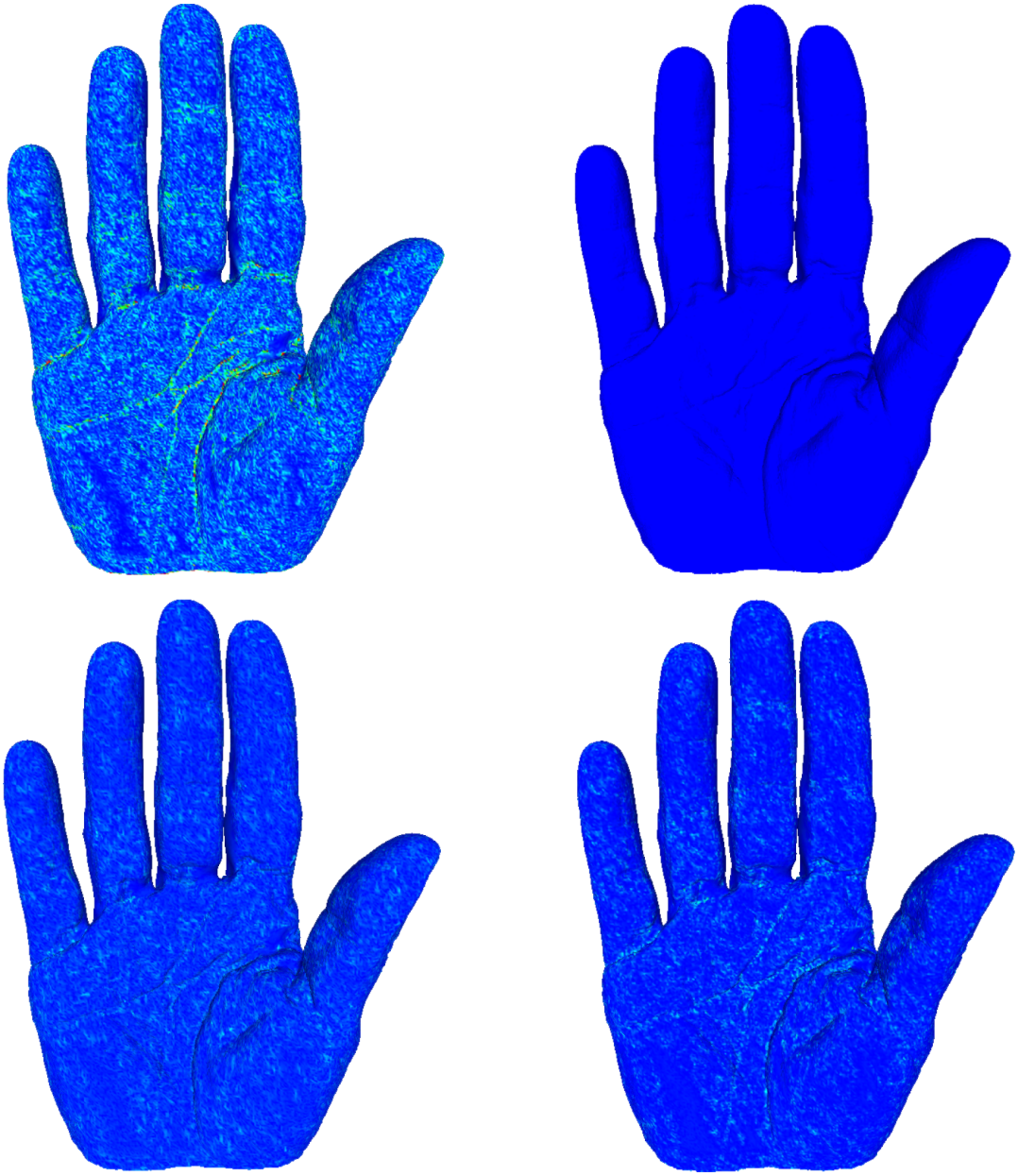

Figure 10: Optimization of the hand model: one-sided distances between the source mesh $M_0$ (left) and the remesh $M_1$ (right) for the original (top) and the optimized (bottom) remesh.

| | max | mean | RMS |
|---|---|---|---|
| $M_0 \rightarrow M_1$ | | | |
| original | 0.605037 | 0.010509 | 0.017324 |
| optimized | 0.258184 | 0.010494 | 0.014429 |
| $M_1 \rightarrow M_0$ | | | |
| original | 0.256721 | 0.009208 | 0.014566 |
| optimized | 0.206491 | 0.010114 | 0.014120 |

Table 1: One-sided distances between the source mesh $M_0$ and the original and the optimized remesh $M_1$ for the hand model in Figure 10.

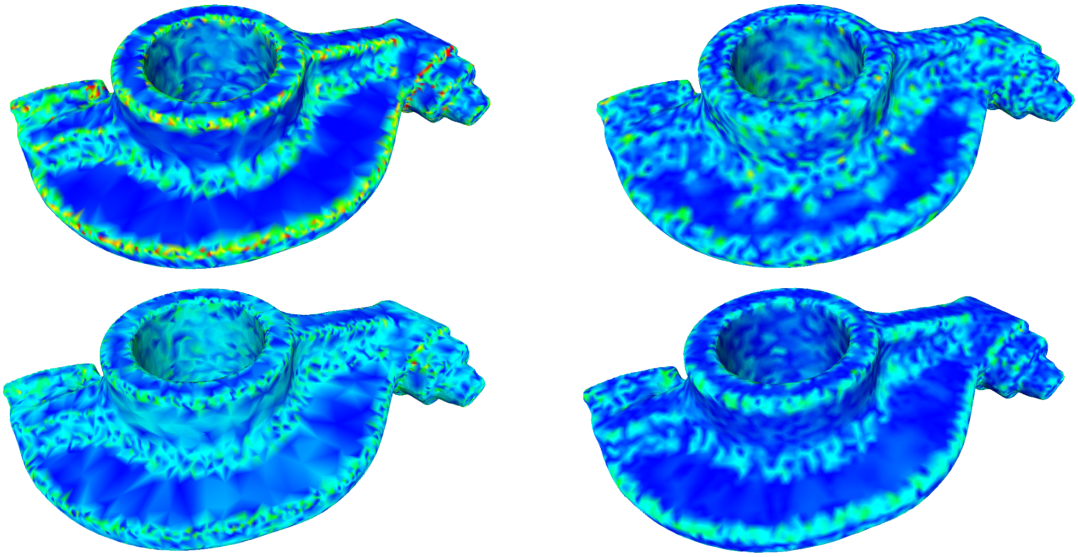

Figure 11: Optimization of the rockerarm: one-sided distances between the source mesh $M_0$ (left) and the remesh $M_1$ (right) for the original (top) and the optimized (bottom) remesh.

| | max | mean | RMS |
|---|---|---|---|
| $M_0 \rightarrow M_1$ | | | |
| original | 0.046016 | 0.001729 | 0.002432 |
| optimized | 0.022029 | 0.001508 | 0.001970 |
| $M_1 \rightarrow M_0$ | | | |
| original | 0.020577 | 0.001572 | 0.002185 |
| optimized | 0.024118 | 0.001411 | 0.001882 |

Table 2: One-sided distances between the source mesh $M_0$ and the original and the optimized remesh $M_1$ for the rockerarm model in Figure 11.

Figures 10 and 11 show the results of the optimization process by colour-coding the one-sided distances between the source mesh and the original as well as the optimized remesh and Tables 1 and 2 list the numerical values. The vertices of the remeshed hand model were sampled from the triangles of the source mesh and we can see that this gives a relatively small distance from the remesh $M_1$ to $M_0$ but a relatively large distance in the other direction, and thus a large Hausdorff distance. By optimizing the mesh we distribute the distance more evenly and thus reduce the Hausdorff distance by about 60%. This is reduced to 50% in the example of the rockerarm, where the vertices of the remesh lie on the PN-triangles over the source mesh. All distances were measured with the Metro tool [4].

## 4 Discussion & Conclusion

We showed that our framework is simple and flexible enough to be applied to a wide range of applications. For example, the methods of Nealen et al. [19] and Liu et al. [16] cannot be used for optimizing remeshes (see Section 3.3) as they can handle only meshes with compatible connectivity. The advantage of [19, 16] is that they need to solve only one $n \times n$ system, whereas we have to do this iteratively (10 to 15 times). However, our system can be solved more efficiently because it is sparser and if additional constraints (e.g. feature preservation) are required, then [19, 16] need to solve a $3n \times 3n$ system, which is as expensive as our iterative approach.

Still our method is able to preserve features well, as shown in Figures 12 and 13. In these examples, we optimized the given mesh (left) using uniform weights $\lambda_{ij}$ and the given mesh itself as the reference

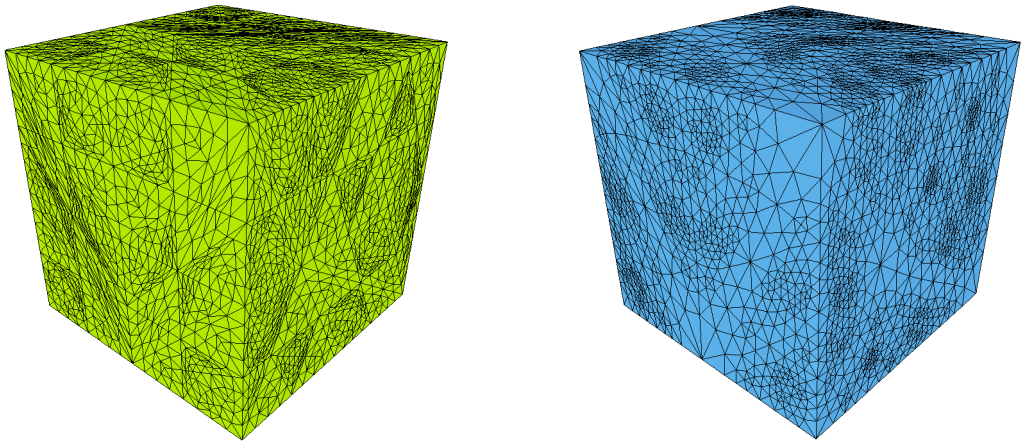

Figure 12: Initial mesh (left) and optimized mesh (right).

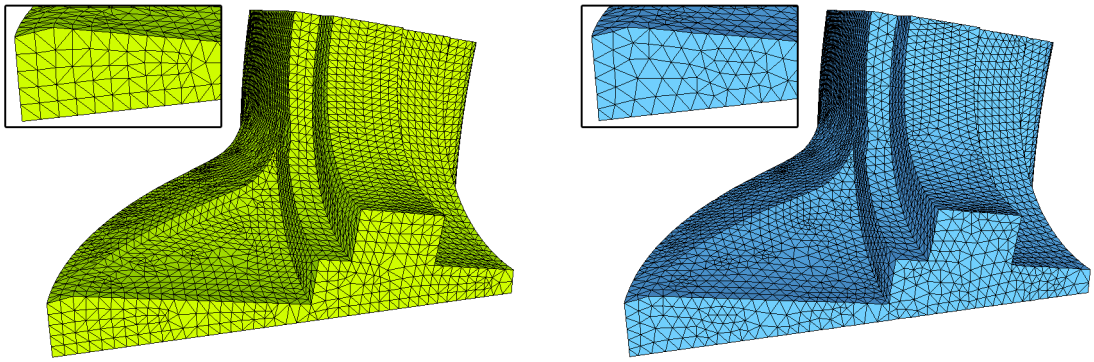

Figure 13: Initial mesh (left) and optimized mesh (right).

geometry. During the optimization, the features of the mesh are nicely preserved, because the approximate Hausdorff distance (4) always pulls the vertices close to the feature lines. But apart from this constraint, all vertices are free to move and yield very uniform triangle shapes (right).

Table 3 compares our results to those of [19]. Although their linear reduced Laplacian approach gives slightly smaller distances, the triangle shapes (measured by the radius ratio, i.e. twice the ratio of the circumradius to the inradius of the triangle) are worse because they need to fix the feature vertices in order to preserve the mesh features. If we use our framework for this kind of shape optimization on the armadillo dataset, we achieve better results than [19] both in terms of distance ($2.45 \cdot 10^{-3}$ vs. $2.63 \cdot 10^{-3}$) and triangle shapes (min: 0.112 vs. 0.091, mean: 0.892 vs. 0.868).

The most time consuming parts of our framework are the factorization of the system (7) using TAUCS [29] and updating the corresponding points. The timings in Table 4 were measured on an Intel Core2 Duo E6400 with 2GB of RAM. One potential drawback of our framework is the calculation of the corresponding points. If the distance between the meshes is too large, the corresponding points cannot be determined correctly anymore.

## Acknowledgements

This joint research project was financially supported by the State of Lower-Saxony and the Volkswagen Foundation, Hannover, Germany. We would like to thank Andrew Nealen for providing us with the data sets from [19].

| | min | mean | distance |
|---|---|---|---|
| original | 0.3235 | 0.8505 | |
| lin. + tplane [19] | 0.2770 | 0.9089 | $1.87 \cdot 10^{-3}$ |
| lin. + red. Laplacian [19] | 0.1525 | 0.9095 | $1.98 \cdot 10^{-4}$ |
| Mesh Massage | 0.3700 | 0.9434 | $1.06 \cdot 10^{-3}$ |

Table 3: Radius ratio and Hausdorff distance for the fandisk model in Figure 13.

| | vertices | factorization | update |
|---|---|---|---|
| cube | 3860 | 0.07 s | 0.21 s |
| fandisk | 6475 | 0.18 s | 0.40 s |
| rockerarm | 10044 | 0.25 s | 0.48 s |
| dancing man | 15830 | 0.46 s | 0.93 s |
| armadillo 17k | 17297 | 0.44 s | 0.98 s |
| horse-to-man | 17489 | 0.48 s | 1.00 s |
| venus-to-skull | 23908 | 0.71 s | 1.10 s |
| hand | 147634 | 14.70 s | 7.69 s |
| armadillo | 172974 | 16.14 s | 10.36 s |

Table 4: Run times for one iteration.

## References


[1] P. Alliez, M. Attene, C. Gotsman, and G. Ucelli. Recent advances in remeshing of surfaces. In L. De Floriani and M. Spagnuolo, editors, *Shape Analysis and Structuring*, Mathematics and Visualization, pages 53–82. Springer, Berlin, Heidelberg, 2008.

[2] N. Anuar and I. Guskov. Extracting animated meshes with adaptive motion estimation. In B. Girod, M. A. Magnor, and H.-P. Seidel, editors, *Proceedings of Vision, Modeling, and Visualization 2004*, pages 63–71, Stanford, CA, Nov. 2004. IOS Press.

[3] G. Charpiat, P. Maurel, J.-P. Pons, R. Keriven, and O. Faugeras. Generalized gradients: Priors on minimization flows. *International Journal of Computer Vision*, 73(3):325–344, July 2007.

[4] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, June 1998.

[5] M. Desbrun, M. Meyer, P. Schröder, and A. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of SIGGRAPH 99*, pages 317–324, Los Angeles, CA, Aug. 1999. ACM Press.

[6] N. Dyn, K. Hormann, S.-J. Kim, and D. Levin. Optimizing 3D triangulations using discrete curvature analysis. In T. Lyche and L. L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces: Oslo 2000*, Innovations in Applied Mathematics, pages 135–146. Vanderbilt University Press, Nashville, TN, 2001.

[7] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proceedings of SIGGRAPH 95*, pages 173–182, Los Angeles, CA, Aug. 1995. ACM Press.

[8] S. Fleishman, I. Drori, and D. Cohen-Or. Bilateral mesh denoising. *ACM Transactions on Graphics*, 22(3):950–953, July 2003. Proceedings of SIGGRAPH 2003.

[9] M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, Mar. 2003.

[10] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, Mathematics and Visualization, pages 157–186. Springer, Berlin, Heidelberg, 2005.

[11] M. Garland. Multiresolution modeling: Survey & future opportunities. In *Proceedings of Eurographics*, STAR – State of The Art Reports, pages 111–131, Milano, Sept. 1999. Eurographics Association.

[12] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Proceedings of SIGGRAPH 93*, pages 19–26, Anaheim, CA, Aug. 1993. ACM Press.

[13] T. R. Jones, F. Durand, and M. Desbrun. Non-iterative, feature-preserving mesh smoothing. *ACM Transactions on Graphics*, 22(3):943–949, July 2003. Proceedings of SIGGRAPH 2003.

[14] V. Kraevoy and A. Sheffer. Cross-parameterization and compatible remeshing of 3D models. *ACM Transactions on Graphics*, 23(3):861–869, Aug. 2004. Proceedings of SIGGRAPH 2004.

[15] C. L. Lawson. *Contributions to the Theory of Linear Least Maximum Approximation.* PhD thesis, University of California, Los Angeles, CA, 1961.

[16] L. Liu, C.-L. Tai, Z. Ji, and G. Wang. Non-iterative approach for global mesh optimization. *Computer-Aided Design*, 39(9):772–782, Sept. 2007.

[17] M. Meyer, H. Lee, A. Barr, and M. Desbrun. Generalized barycentric coordinates for irregular polygons. *Journal of Graphics Tools*, 7(1):13–22, 2002.

[18] T. S. Motzkin and J. L. Walsh. Polynomials of best approximation on a real finite point set. *Transactions of the American Mathematical Society*, 91(2):231–245, May 1959.

[19] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. Laplacian mesh optimization. In *Proceedings of GRAPHITE 2006*, pages 381–389, Kuala Lumpur, Nov./Dec. 2006. ACM Press.

[20] Y. Ohtake and A. Belyaev. Mesh optimization for polygonized isosurfaces. *Computer Graphics Forum*, 20(3):368–376, Sept. 2001. Proceedings of Eurographics 2001.

[21] S. J. Owen, D. R. White, and T. J. Tautges. Facet-based surfaces for 3D mesh generation. In *Proceedings of the 11th International Meshing Roundtable*, pages 297–312, Ithaca, NY, Sept. 2002.

[22] U. Pinkall and K. Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

[23] H. Pottmann, Q.-X. Huang, Y.-L. Yang, and S.-M. Hu. Geometry and convergence analysis of algorithms for registration of 3D shapes. *International Journal of Computer Vision*, 67(3):277–296, May 2006.

[24] J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe. Inter-surface mapping. *ACM Transactions on Graphics*, 23(3):870–877, Aug. 2004. Proceedings of SIGGRAPH 2004.

[25] R. W. Sumner and J. Popović. Deformation transfer for triangle meshes. *ACM Transactions on Graphics*, 23(3):399–405, Aug. 2004. Proceedings of SIGGRAPH 2004.

[26] V. Surazhsky, P. Alliez, and C. Gotsman. Isotropic remeshing of surfaces: a local parameterization approach. In *Proceedings of the 12th International Meshing Roundtable*, pages 215–224, Santa Fe, NM, Sept. 2003.

[27] V. Surazhsky and C. Gotsman. Explicit surface remeshing. In *Proceedings of SGP 2003*, pages 20–30, Aachen, June 2003. Eurographics Association.

[28] G. Taubin. A signal approach to fair surface design. In *Proceedings of SIGGRAPH 95*, pages 351–358, Los Angeles, CA, Aug. 1995. ACM Press.

[29] S. Toledo. TAUCS: A library of sparse linear solvers, version 2.2. Available online at `www.tau.ac.il/~stoledo/taucs/`, 2003.

[30] R. van Damme and L. Alboul. Tight triangulations. In M. Dæhlen, T. Lyche, and L. L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces*, Innovations in Applied Mathematics, pages 517–526. Vanderbilt University Press, Nashville, TN, 1995.

[31] E. L. Wachspress. *A Rational Finite Element Basis.* Academic Press, New York, 1975.