Consistent Mesh Parameterizations

Emil Praun
Princeton University

Wim Sweldens
Bell Labs

Peter Schröder
Bell Labs

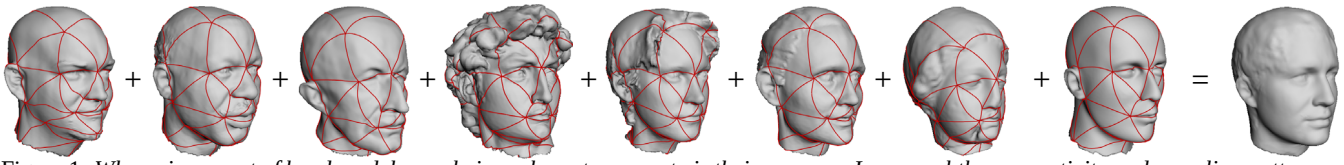


Figure 1: *When given a set of head models an obvious shape to compute is their average. In general the connectivity and sampling patterns of the models are different and computing the average is non trivial. After computing consistent mesh parameterizations (red patch boundaries) and remeshing, all models have the same connectivity and sampling pattern so computing the average becomes trivial.*

Abstract

A basic element of Digital Geometry Processing algorithms is the establishment of a smooth parameterization for a given model. In this paper we propose an algorithm which establishes parameterizations for a set of models. The parameterizations are called consistent because they share the same base domain and respect features. They give immediate correspondences between models and allow remeshes with the same connectivity. Such remeshes form the basis for a large class of algorithms, including principal component analysis, wavelet transforms, detail and texture transfer between models, and n-way shape blending. We demonstrate the versatility of our algorithm with a number of examples.

1 Introduction

Digital Geometry Processing (DGP) is the field concerned with the construction of signal processing style algorithms for geometry. Due to the non-Euclidean nature of surface geometry the construction of DGP algorithms is fundamentally more difficult than the construction of classical signal processing algorithms. Sound (1D), images (2D), and video (3D), are readily parameterized onto a Euclidean space, e.g., an image is given by the irradiance function over a section of the plane. In addition images are always sampled using a Cartesian grid. As a result, simple operations such as averaging two images or computing the norm of their difference are easy. The same is not true for geometry. There are two causes for this: (a) the non-Euclidean nature of geometry and (b) the generally differing sampling patterns and connectivity of meshes describing geometry. DGP algorithms involving multiple models require a common parameterization and a common sampling pattern. Computing a global parameterization and remeshing for a single model is a difficult problem in itself and has received considerable attention as it is a fundamental step in many algorithms from texture mapping and shape blending to physical simulation, compression, and data analysis.

In this paper we will go beyond the usual parameterization problem and compute parameterizations for a *group* of models. We call a set of parameterizations consistent when they share the same base domain and respect features. Note that this implies that all models need to have the same genus. In this paper we will focus on the genus zero, orientable manifold case, although many of the techniques carry over to higher genus. Consider for example a set of head scans; we say that their parameterizations are consistent if they all use the same base domain, such as a low polygon-count head model, and if all parameterizations respect previously defined head features such as eyes, nose, mouth, etc.

Consistent parameterizations give immediate point correspondences between all the models and allow us to remesh each model with the *same* connectivity. Therefore, every vertex in one mesh has a *unique* corresponding vertex in every other mesh. This in turn enables a whole series of applications ranging from n-way shape blending to the transfer of attributes, such as textures, details, or animation controls, from one model to a whole set of models. Additionally a number of new geometry processing algorithms which involve many models simultaneously, e.g., principal component computations, become possible for the first time. This is illustrated in Figure 2: multiple models, which may be geometrically quite dissimilar, get parameterized onto the same base domain and remeshed with identical connectivity for subsequent DGP processing.

One problem with existing parameterization algorithms is that even two very similar models can easily end up with different base domains and hence inconsistent parameterizations. There is no fundamental reason for this and our algorithm overcomes this problem.

For our method, the base domain may be user specified or one can be found automatically, for example, by applying an existing method to one of the models. Each of the models needs to have features outlined and annotated either manually or through automatic feature identification. Once our algorithm computes consistent parameterizations, all models can be remeshed using identical connectivity. Because of their obvious advantages such as simple data structures, easy filtering, wavelet transforms, and excellent compressibility, we consider here only *semi-regular* meshes, i.e, meshes formed by recursive (possibly adaptive) regular refinement from some base domain.

Contributions Given a base domain and a set of orientable genus zero models with identified features, we present an algorithm that computes a consistent parameterization for all of them. This is done by tracing on each mesh a net of curves that is provably topologically equivalent to the connectivity of the base domain. The models are not required to be geometrically close. Subsequently we use the parameterizations to compute remeshes with identical connectivity and give examples of how such meshes can be used in a variety of DGP applications.

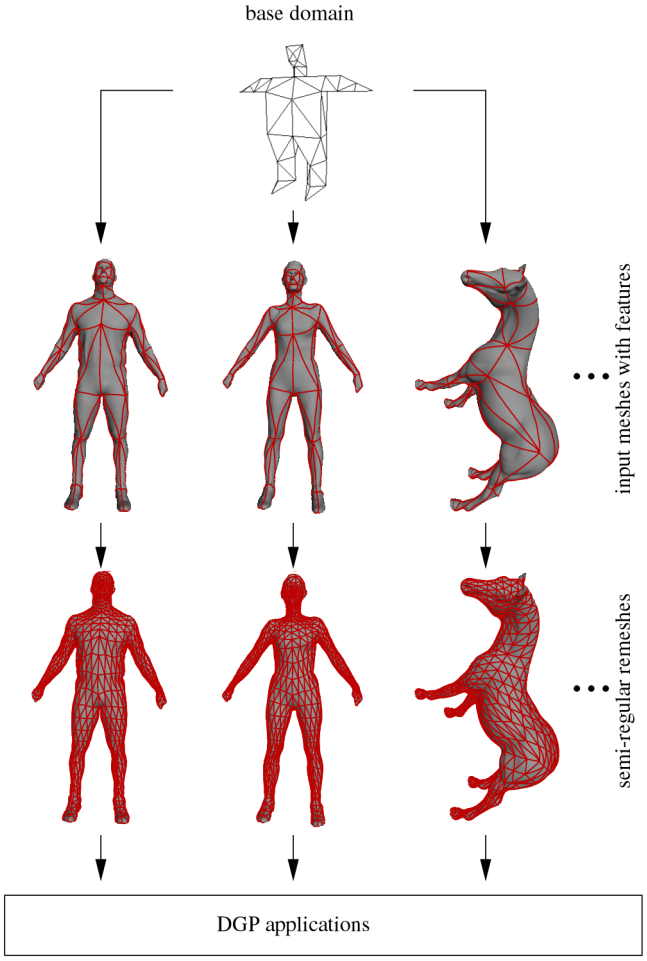


Figure 2: *Multiple models are parameterized with respect to the same base domain using feature point (and edge) mappings. Subsequent remeshes ensures consistent sampling patterns for downstream DGP applications.*

Related Work Parameterization algorithms typically start with a given model in the form of a triangle mesh with irregular connectivity and construct a set of patches or a semi-regular mesh on output (note that the two are equivalent). Eck and co-workers [2] produced a semi-regular mesh fully automatically, while Krishnamurthy and Levoy [8] gave the user responsibility for laying out a set of NURBS patch boundaries. In contrast, Lee and co-workers [10] employed mesh simplification with constraints to accommodate any user supplied data in the construction of the parameterization during an otherwise fully automatic process.

None of these approaches considered building parameterizations for multiple models simultaneously. This problem arises naturally in the context of morphing when a mapping correspondence between two models is the explicit goal. Lee and co-workers [9] use their previous MAPS work to independently establish parameterizations for two models followed by solving the correspondence problem on the base domains. The two parameterizations are not consistent as the base domains are different. This creates the need for a common "meta-mesh" able to realize the simultaneous parameterization of the two original models. Unfortunately this algorithm does not scale, since the meta-mesh typically has much higher complexity (a reported $10\times$) than either original mesh. Additionally it is not clear how it might generalize to n-way simultaneous parameterizations.

Marschner and co-workers [11] confront a problem very close to our setting when they seek to animate a number of different faces via a single, prototype patch layout. Since their prototype face, i.e., the *embedding* of the prototype layout, is already very close to a given face they report a simple least squares fitting procedure to work well. In contrast we aim to concurrently parameterize models which may be fairly different geometrically, for example a horse and a human body. In that case there is no embedding of some prototype layout that is simultaneously close to both shapes. Hence a simple least squares matching procedure will likely not succeed.

2 Algorithms

We begin by establishing terminology. A triangle mesh $\mathcal{M}$ is a pair $(\mathcal{P}, \mathcal{K})$, where $\mathcal{P}$ is a set of N point positions $\mathcal{P} = \{\mathbf{p}_i = (x_i, y_i, z_i) \in \mathbf{R}^3 \mid 1 \leq i \leq N\}$, and $\mathcal{K}$ is an *abstract simplicial complex* which contains all the topological, i.e., adjacency information. The complex $\mathcal{K}$ is a set of subsets of $\{1, \ldots, N\}$. These subsets come in three types: vertices $\{i\}$, edges $\{i, j\}$, and faces $\{i, j, k\}$. Two vertices $\{i\}$ and $\{j\}$ are *neighbors* if $\{i, j\} \in \mathcal{K}$. The 1-ring neighbors of a vertex $\{i\}$ form a set $\mathcal{V}(i) = \{\{j\} \mid \{i, j\} \in \mathcal{K}\}$.

As argued above we will work with semi-regular meshes. For concreteness we will further assume that such meshes are built by repeated triangle quadrisection starting from a coarse irregular mesh $\mathcal{N}_0 = (\mathcal{Q}_0, \mathcal{L}_0)$ with the finer meshes denoted $\mathcal{N}_j = (\mathcal{Q}_j, \mathcal{L}_j)$. Nothing in the method prevents the use of quadrilaterals instead of triangles or regular refinement procedures other than quadrisection.

A typical remeshing procedure starts from an irregular input mesh $\mathcal{M}$ and computes a base domain with connectivity $\mathcal{L}_0$ as well as a bijective mapping between the base domain and $\mathcal{M}$. This mapping is subsequently used to build the remeshes $\mathcal{N}_j$ for $j > 0$. Note that this can be done in an adaptive fashion if needed.

In our setting, we are given a *set* of meshes $\mathcal{S} = \{\mathcal{M}^m \mid 0 \leq m < M\}$. The purpose is to compute a semi-regular remeshing for all of them with a *common* base domain connectivity $\mathcal{L}_0$ (see Figure 2 top). We also need to ensure that the individual parameterizations respect features as desired. As an example, consider a set of heads. Typically one would seek mappings that will respect features such as eyes, nose, ears, etc.

For convenience, we treat the common base domain as a special mesh $\mathcal{B} = (\mathcal{P}, \mathcal{L}_0)$. Since this base domain does not have associated geometry, its vertex locations do not need to be in $\mathbf{R}^3$. To each of the F feature vertices $\mathbf{b}_f$ we simply give canonical locations in $\mathbf{R}^F$. Let $\mathbf{b}_f$ be the f-th basis vector, i.e., the f-th component is one and all others are zero.

Basic Setup Assume we have identified, either by hand or through some feature detection algorithm, F feature points in each of the meshes, with $\mathbf{p}_f^m$ denoting the f-th feature point in the m-th mesh[1]. Furthermore the connectivity $\mathcal{L}_0$ between the feature points is also given. Each triangle in $\mathcal{L}_0$ corresponds to a patch on each of the meshes $\mathcal{M}^m$, while edges in $\mathcal{L}_0$ correspond to patch boundaries (middle part of Figure 2). We will refer to the layout of patch boundaries on the meshes $\mathcal{M}^m$ as "nets." The task now is to trace these patch boundaries on each of the meshes $\mathcal{M}^m$ between the points $\mathbf{p}_f^m$ in a manner such that the resulting net is (1) topologically equivalent to $\mathcal{L}_0$ and (2) outlines fair patches. The first constraint is easy to state and our algorithm guarantees that it is satisfied (Section 2.1). The second condition is more difficult to capture in an objective criterion and our algorithm uses a number of heuristics that produce fair patches in practice (Section 2.2). Once we have computed a net, the surface is parameterized using Normal Meshes [4], but without necessarily imposing the normality constraint.

[1] We only discuss feature points although feature lines can be treated similarly. If the features are points inside a triangle, the triangle is split to produce a feature vertex.

2.1 Topologically Equivalent Patch Boundaries

Given that both $\mathcal{B}$ and $\mathcal{M}$ are assumed to be orientable manifolds, two conditions need to be satisfied for the patch boundaries to be topologically equivalent with $\mathcal{L}_0$:

1. Two patch boundaries may only intersect at a feature vertex.
2. Each feature vertex has a consistent cyclical ordering of its edges in both $\mathcal{B}$ and $\mathcal{M}$ [1].

It is tempting to simply trace shortest paths on $\mathcal{M}$ for each edge in $\mathcal{L}_0$ (e.g., by employing a standard "brush fire" algorithm [7]). Unfortunately this can lead to intersecting patch boundaries and may not respect the vertex edge ordering (see Figures 3 and 7). Therefore we use a *restricted* brush fire algorithm to trace a path

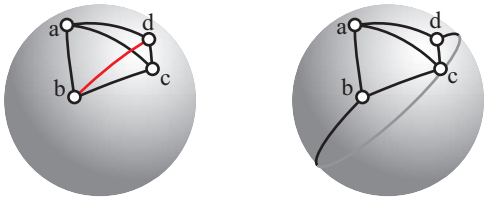


Figure 3: *Example patch boundaries of a sphere with four fully connected feature vertices. A naïve shortest path strategy does not lead to a topologically correct net (left), violating both the crossing property and the vertex edge ordering. On the right is a topologically correct net.*

between two feature vertices in a topologically equivalent manner. Patch boundary crossings are avoided by making sure that the brush fire never crosses a previously traced path. Once a feature vertex is incident to at least two traced paths, we also have to make sure that any new paths respect the cyclical ordering of edges at that vertex, i.e., new paths need to be attached in the appropriate sector at both vertices. This is ensured by starting the brush fire in the correct sector at the source and only terminating when it reaches the destination in the correct sector, see Figure 4.

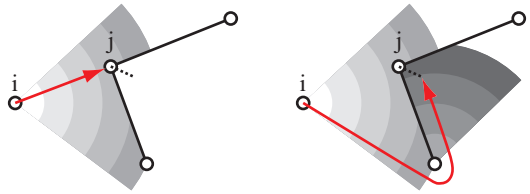
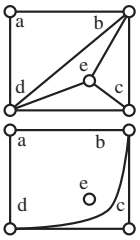


Figure 4: *To find a curve for edge $\{i, j\} \in \mathcal{L}_0$ a brush fire is started at $\{i\}$. When it reaches $\{j\}$ a check is performed to see whether it reaches $\{j\}$ in the correct sector (not true in the example shown on the left). The brush fire continues and will eventually reach $\{j\}$ in the correct sector defining a topologically equivalent curve from $\{i\}$ to $\{j\}$ (right side). Already drawn paths act as fire walls during the brush fire propagation.*

Additionally, the order in which paths are traced is important. For a random ordering, even the restricted brush fire algorithm does not guarantee termination. For example, it is possible to "encircle" a vertex with paths, making it unreachable. In the adjacent figure, the top diagram shows the desired connectivity. In cannot be achieved, however, in the lower setup, since the vertex $\{e\}$ has been encircled in the patch $\{a, b, d\}$, and cannot be linked with $\{c\}$.



To solve the encircling problem, we do not trace any paths that would complete cycles until a spanning tree of $\mathcal{L}_0$ has been traced. This guarantees that no vertex can be encircled during tree building. Once we have a tree, we can complete the net by adding the remaining patch boundaries in any order. To prove this claim we only need to show that the completion algorithm cannot get stuck.

Proof of Correctness We prove by contradiction that completion of the net after building a spanning tree always terminates and produces a topologically equivalent net. Assume that one of the invocations of the brush fire algorithm, say for edge $\{a, b\} \in \mathcal{L}_0$, could not complete; let this be the first such occurrence. This implies the existence of a cycle C of paths on the model separating $\{a\}$ from $\{b\}$ (i.e., the fire gets caught). Given that the model and base domain are both genus zero, C also splits the base domain in two regions. Since the set of already traced curves contains a spanning tree of $\mathcal{L}_0$, it forms a connected graph, and therefore must contain a path P from $\{a\}$ to $\{b\}$. Since $\{a\}$ and $\{b\}$ are in different model regions, P and the cycle C have to cross. Curves can only cross at vertices, so P and C must cross at a subset of the vertices of P. Follow P vertex by vertex and at every intersection vertex, record whether the base domain or model region change. Given that neighborhood ordering has to be the same on the base domain and model at each vertex, region changes have to occur simultaneously on the base domain and model. Since $\{a\}$ and $\{b\}$ are in the same region on the base domain ($\{a, b\} \in \mathcal{L}_0$), they have to be in the same region on the model. This contradicts our assumption and proves our claim: we are assured that the brush fire can always terminate in the correct sector and the net can be completed.

2.2 Tracing Fair Boundary Curves

The above algorithm guarantees a topologically equivalent mapping of the base domain to the desired model, however, it could still be greatly distorted. We now present a modified algorithm that has the same correctness guarantee and attempts to ensure a fair mapping. In general we want the following quality criteria:

1. equal distribution of surface area amongst patches;
2. smooth patch boundaries;
3. fair patch boundaries; in particular they should not "swirl."

The first two criteria are easy to understand and intuitively clear and can be achieved through relaxation [3]. The third one is more difficult. In essence we want to avoid unnecessary "winding" or "swirling" of the curves. The swirling phenomenon leads to particularly nasty patches that cannot be fixed through relaxation. A simple example of the "swirl" operator, a bijective map from a mesh onto itself is show in Figure 5.

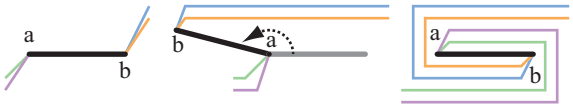


Figure 5: *An example of the swirl operator: b turns around a pushing patch boundaries ahead of itself until b comes back to its original position (left to right) and a topologically equivalent arrangement.*

Clearly we want to avoid swirling and choose the least distorted or fairest amongst all topologically equivalent nets. Such a net can be defined as the one which achieves a global minimum of the fairness functional

$$\sum_{\mathbf{C} \in \mathbf{N}} \int_{s \in \mathbf{C}} \|g'\| + \|g''\| \, ds,$$

where the summation is over all curves $\mathbf{C}$ in the net $\mathbf{N}$, and g is the arc length parameterization of $\mathbf{C}$. This leads to a non-convex, mixed discrete-continuous global optimization problem, which seems to be intractable at this point. Instead we exploit a number of heuristics that do not require a global solve and lead to good patch layouts in practice.

We discuss each of these algorithm modifications in turn. Note that none impact the correctness guarantee.

Parameterization In preparation for path tracing we compute a parameterization of the model. Each vertex of the mesh $\mathcal{M}$ is parameterized onto the base domain $\mathcal{B}$. The parameter value of a vertex $\{i\}$ is π_i, which is an F-vector. The feature points $\mathbf{p}_f$ are already associated with the coarsest level vertices $\mathbf{b}_f$: $\pi_f = \mathbf{b}_f$. Parameter values for the other vertices of $\mathcal{M}$ are computed by solving a linear system, using conjugate gradients:

$$\pi_i = \sum_{\{j\} \in \mathcal{V}(i)} w_{ij} \pi_j .$$

The weights w_{ij} are computed using Floater's shape-preserving scheme [3]. As a result $w_{ij} \geq 0$ and their sum over j equals 1 for each i, implying that the π_i are convex combinations of the π_j. Taken together with the fact that the boundary conditions $\pi_f = \mathbf{b}_f$ for the linear system are F-vectors with one non-zero component equal to 1, the components of each of the computed π_i will all be non-negative and sum to 1.

Tracing Curves Assume we want to trace the curve between feature points f and f'. The main idea is to set an objective function and trace a curve along the local minimum of the objective function. The objective function at a vertex $\{j\}$ is given by

$$K_j = 1 - \pi_j[f] - \pi_j[f'],$$

where $[f]$ denotes the f-th component of an F-vector. The algorithm is the same as before but only with K_j as the priority in the queue. Clearly $K_f = K_{f'} = 0$ while $0 < K_j \leq 1$ for other vertices j. Since the sum of π_j's components is 1, our objective function tries to seek f and f', minimizing the influence of other feature vertices, and making the traced curves repel each other. This helps avoid swirling.

Priority Queues and Spanning Tree Construction For all the edges of $\mathcal{L}_0$ we first trace corresponding tentative curves using the constrained brush fire algorithm described earlier. The constraints are given by the curves that have already been inserted into the net (initially, none). We record the tentative length and path of these curves and insert them into the priority queue based on length. Upon removal from the priority queue—during spanning tree construction—we check to see if the recorded (tentative) path is still valid and does not violate constraints provided by other curves recently inserted into the net. If it is not topologically valid, we retrace the curve and update its priority. If it is topologically valid, we run the swirl detector (Figure 6) for both adjacent $\mathcal{L}_0$ triangles. If at least one fails, we put the edge back into the queue with a penalty. If both pass we add the curve to the net.

Swirl Detection Consider tracing a path for $\{a, b\} \in \mathcal{L}_0$ (Figure 6) with $\{a, b, c\}$ and $\{a, d, b\}$ in $\mathcal{L}_0$. To detect a possible swirl, we parametrically trace a line from $\{c\}$ to the closest point on the image of $\{a, b\}$, i.e., the first point that the brush fire meets on the path from $\{a\}$ to $\{b\}$. This path should arrive on the left side of the oriented path from $\{a\}$ to $\{b\}$ (Figure 6, right). If it arrives on the wrong side, then the mapping of the triangle $\{a, b, c\}$ appears flipped. Consequently, one of the traced edges $\{a, b\}$, $\{b, c\}$ or $\{c, a\}$ would have to be routed around an opposing vertex (the figure shows $\{c, a\}$ "taking the detour"). These three configurations are equivalent to each other under a sequence of swirls. We can decide which is the best configuration only after curves linking $\{a\}$, $\{b\}$, and $\{c\}$ to the rest of the net have been added. Therefore, we postpone tracing of $\{a, b\}$ by introducing it back into the priority queue with a penalty. The same test is applied starting from $\{d\}$.

Complete Net After the spanning tree has been constructed we complete the net as before except that a priority queue based on length is used.

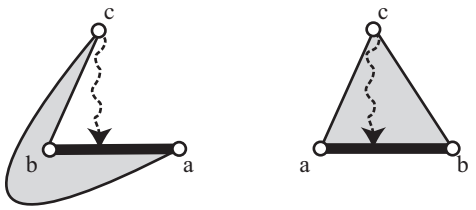


Figure 6: *Swirl detector: If the shortest path from $\{c\}$ to $\{a, b\}$ falls on the wrong side (left) the triangle $\{a, b, c\}$ is considered flipped and may lead to swirls. Adding the path $\{a, b\}$ to the net is postponed. On the right the trace reaches $\{a, b\}$ on the correct side and the path is accepted.*

Edge Straightening After tracing the patch boundaries we obtain a net of curves topologically equivalent to $\mathcal{L}_0$. These curves are composed of segments residing on the mesh edges. Next we straighten the curves, allowing them to cross the interior of triangles, and ensuring that all patches have approximately equal area.

For all feature vertices $\{i\}$ all incident curves $\{i, j\}$ are straightened in a single operation. First, gather all triangles on the mesh which are interior to the image of all $\{i, j, k\} \in \mathcal{L}_0$. This submesh of the model is parameterized onto a convex region in the plane as follows. Map the feature vertices $\{j\}$ belonging to $\mathcal{V}(i)$ to the vertices of a planar n-gon inscribed in the unit circle. The angles subtended by each side $\{j, k\}$ of the n-gon are proportional to the length of the corresponding net curves. Boundary curves $\{j, k\}$ are mapped to straight lines, assigning coordinates for the interior curve points as convex combinations of the curve endpoints. Finally, we solve for the coordinates of patch interior points using Floater's scheme. Once we have this 2D parameterization we can replace the curves $\{i, j\}$ incident to $\{i\}$, by mapping the respective 2D line segments back onto the 3D model. In general these lines will cut across existing triangles which must be split accordingly to maintain a valid triangulation. The straightening process can be iterated, although in practice this did not seem necessary.

Implementation Note For efficiency, we run our net tracing algorithm on a simplified version of the model and later transfer the net to the original model. For the examples in this paper, we take a simplified model with approximately one thousand vertices.

During path tracing we must allow multiple paths to traverse the same edge and impose an ordering on them. We do this with a special "lane" data structure which allows several paths to pass through the same mesh edge, while maintaining a conceptual epsilon interval separation between them. In effect an edge is treated as a highway with many ordered lanes. This requires a modification of the queue in the brush fire graph traversal. Ordinarily this queue controls which vertex is visited next. In our setting we must also maintain a notion of which lane the fire is traveling on to properly respect fire walls. Thus, the priority queue maintains not just vertices but vertices and their associated lanes.

Non-Feature Base Domain Vertices So far we assumed that all vertices in the base domain are features and have correspondences with vertices in the meshes $\mathcal{M}^m \in \mathcal{S}$. However, in some applications not all base domain vertices are features. A detailed face animation model can have up to a thousand vertices, but only about a hundred feature vertices. We can handle this case in two stages. Say, the base domain $\mathcal{B}$ has V vertices, F of which are features vertices and $B - F$ are not. In the first stage, we construct a smaller base domain $\mathcal{B}^0$ by removing the $B - F$ non feature vertices. We now apply our algorithm with $\mathcal{B}^0$ as the base domain and compute consistent parameterizations for the meshes $\mathcal{M}^m$ *as well as* the original base domain $\mathcal{B}$. (Note that their is no need for remeshing at this point.) Next we can use the resulting bijection between $\mathcal{B}$ and the meshes $\mathcal{M}^m$ to place the remaining $V - F$ vertices of $\mathcal{B}$ on the meshes $\mathcal{M}^m$. Using these new correspondences,

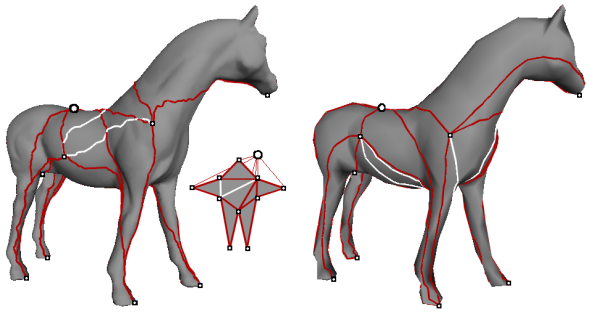


Figure 7: *Patch boundary curves from the star connectivity mapped onto the horse. The star has one vertex on its back connecting to all other vertices. On the left the naïve algorithm. Note how several curves intersect. The white curves which are supposed to be on the belly end up using the shortest path across the back of the horse. On the right the result of our algorithm, which properly traces the white curves across the chest of the horse.*

we can run our algorithm again with $\mathcal{B}$ as base domain and compute consistent parameterizations and remeshes for the $\mathcal{M}^m$ meshes.

2.3 Examples

Figure 7 shows the layout of a very simple star-like patch network onto the horse. On the left is the result of the naïve curve tracing algorithm. Several curves intersect and others pass on the wrong side of the horse. On the right is the result of the modified algorithm. This example also shows that we can transfer patch layouts between very dissimilar objects.

Figure 8 shows a more realistic example with a more detailed patch layout which is shown on the top left. It is transferred to a human, a horse, and a cow.

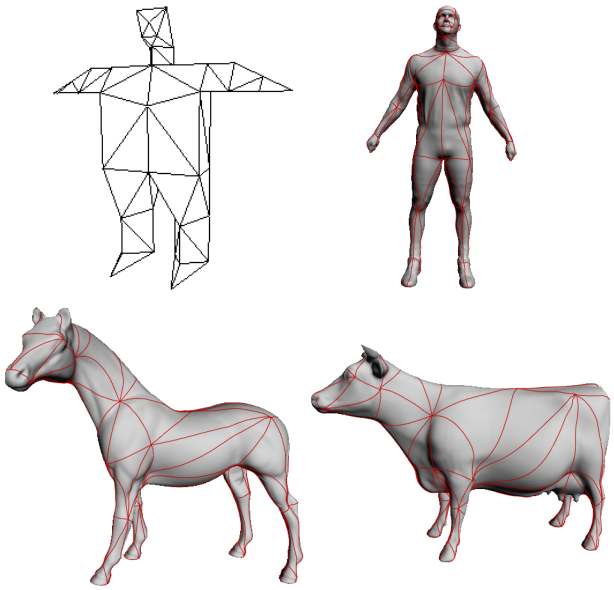


Figure 8: *Another mapping of a patch layout (top left) applied onto the human figure, the horse, and the cow models.*

All the examples presented here take about the same time to process. On a 900Mhz Pentium III machine, tracing the curve net on the coarse mesh (1000 triangles) takes around 5 seconds, transferring to the original mesh (100K triangles) takes 2 seconds, smoothing 1.5 minutes, and remeshing to 43K triangles 6 minutes. Selecting 54 feature vertices on the human figure, for example, takes around 10 minutes of user time.

3 Applications

Once a consistent parameterization has been established for several meshes, we can use the resulting correspondences for many different DGP tasks. Here we sketch a few exemplary applications to indicate the variety of possible algorithms.

Principal Mesh Components This application is motivated by the use of principal component analysis for images. For example, given a number of images of faces that are aligned and illumination-equalized one can compute "eigenfaces," i.e., a set of orthogonal images which are eigenvectors of the variance/covariance matrix of the set of face images [12]. We can perform a similar computation with meshes. Figure 1 shows a number of head meshes that are all aligned through rigid body motion and scaled to equal volume. The rightmost mesh represents the mean. Figure 9 shows the three main eigenheads, each visualized in a sum with the mean. Each mesh in the original set can be characterized

Figure 9: *The first three principal component meshes for the heads in Figure 1. The first eigenhead seems to indicate hair on the front of the skull, the second hair on the back of the skull, and the third whether the face carries a smile.*

by its eigendecomposition. Such feature vectors can be useful in compression, recognition or database search. Figure 10 shows how eigenheads can be used in filtering applications. The middle image shows the result of tripling the third eigenmode of the leftmost head. The right image is the result of doubling all but the mean eigenmode.

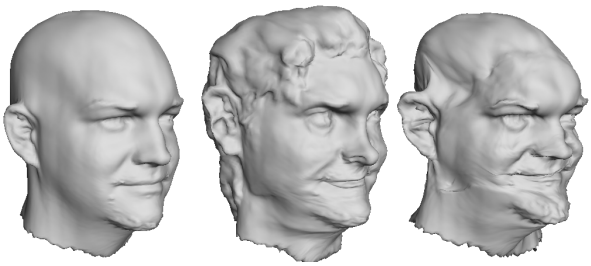


Figure 10: *Left: original head. Middle: the third eigenmode was tripled. Right: all the eigenmodes (except the mean) were doubled.*

Transfer of Textures Given the vertex-wise correspondence between two meshes it is trivial to transfer attributes from one model to another through direct parametric mapping. Figure 11 shows a simple example. The texture of one human body scan is applied to another human body scan and, for fun, to the horse model.

Transfer of Wavelet Details A more interesting transfer of attribute data can be performed in the wavelet domain. For example, one can compute a wavelet transform of two models with consistent parameterizations and then perform scale dependent blending. To illustrate this idea we combine the base domain (coarsest level scale coefficients or lowest frequencies) of one model with the wavelet coefficients (higher frequencies) from another model. For this procedure Loop-based wavelets as described by [6] were used. Figure 12 shows the result of applying the cow wavelet coefficients to

Figure 11: *Texture transfer from one model to others.*

the horse coarsest level control mesh and vice versa. Such mappings can also be the basis for morphs with scheduled transitions in the scale domain (similar to scheduled Fourier morphing [5]).

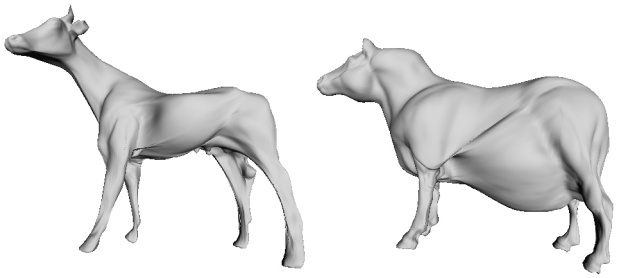


Figure 12: *Application of details from one model to the coarsest control mesh of another (cow onto horse and vice versa).*

Shape Blending Establishing a mapping between different shapes, which may be quite dissimilar, is the first step in any morphing application. Our algorithm can establish the necessary mapping even when there are multiple models to be blended. Consequently n-way blends of free-form models are greatly facilitated. Figure 13 shows an whimsical example of various affine combinations of a cow, a horse, and a human figure.

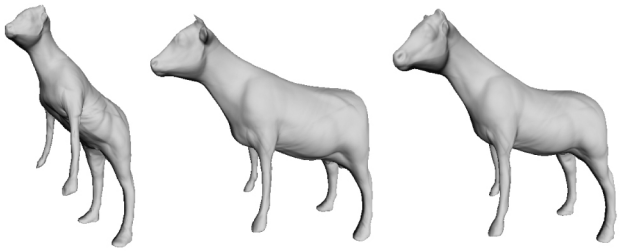


Figure 13: *Affine combinations of the human figure, cow, and horse: a) 50% human, 25% horse, 25% cow; b) 50% cow, 25% human, 25% horse; c) 50% horse, 25% human, 25% cow.*

4 Conclusion and Future Work

We have presented an algorithm for the simultaneous, consistent parameterization of multiple shapes. The algorithm is guaranteed to lay out patch boundaries in a topologically equivalent way and we have developed a suite of heuristics to ensure that the layout is fair. Such a parameterization forms the basis of consistent semi-regular samplings of the input shapes which in turn form the foundation for a wide variety of Digital Geometry Processing algorithms. A few examples of such algorithms were presented.

As digital geometry becomes more widespread the need for a geometry processing toolbox of efficient algorithms with a well developed mathematical apparatus continues to increase. Construction of "good" parameterizations is the first step to enable a whole host of DGP algorithms. Interesting areas for future study include:

- extending the method to handle higher-genus models, boundaries, missing feature point specifications, and additional constraints;
- applications in animation and transfer of animation controls from one shape to another;
- computing principal components for a large database of models and using them for search and recognition;
- code design for efficient compression of multiple shapes;
- consistent watermarking and authentication.

Acknowledgments The research reported here was supported in part by NSF (DMS-9874082, DMS-9872890, ACI-9982273), Alias|Wavefront, Microsoft, Intel, Lucent Technologies, and the Packard Foundation. Special thanks to Matthieu Ferrant and Silvain Jaume for discussions motivating this research, the TigGraph reviewers for feedback, Adam Finkelstein for support, Tony DeRose for insightful comments, Igor Guskov for his remeshing code, and Sooha Lee for application discussions. Models are courtesy Cyberware, Stanford University, Max Planck Institut für Computer Graphik, and the University of Washington.

References

[1] AKLEMAN, E., AND CHEN, J. Guaranteeing 2-Manifold Property for Meshes. In *Proceedings of the Shape Modeling International*, 18–25, 1999.

[2] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. *Proceedings of SIGGRAPH 95* (1995), 173–182.

[3] FLOATER, M. S. Parameterization and Smooth Approximation of Surface Triangulations. *Computer Aided Geometric Design 14* (1997), 231–250.

[4] GUSKOV, I., VIDIMČE, K., SWELDENS, W., AND SCHRÖDER, P. Normal Meshes. *Proceedings of SIGGRAPH 2000* (2000), 95–102.

[5] HUGHES, J. F. Scheduled Fourier Volume Morphing. *Computer Graphics (Proceedings of SIGGRAPH 92) 26*, 2 (1992), 43–46.

[6] KHODAKOVSKY, A., SCHRÖDER, P., AND SWELDENS, W. Progressive Geometry Compression. *Proceedings of SIGGRAPH 2000* (2000), 271–278.

[7] KIMMEL, R., AND SETHIAN, J. Fast Marching Method on Triangulated Domains. In *Proceedings of the National Academy of Science*, vol. 95, 8341–8435, 1998.

[8] KRISHNAMURTHY, V., AND LEVOY, M. Fitting Smooth Surfaces to Dense Polygon Meshes. *Proceedings of SIGGRAPH 96* (1996), 313–324.

[9] LEE, A., DOBKIN, D., SWELDENS, W., AND SCHRÖDER, P. Multiresolution Mesh Morphing. *Proceedings of SIGGRAPH 99* (1999), 343–350.

[10] LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. MAPS: Multiresolution Adaptive Parameterization of Surfaces. *Proceedings of SIGGRAPH 98* (1998), 95–104.

[11] MARSCHNER, S., GUENTER, B., AND RAGHUPATHY, S. Modeling and Rendering for Realistic Facial Animation. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering* (2000), 231–242.

[12] TURK, M., AND PENTLAND, A. Eigenfaces for Recognition. *Journal of Cognitive Neuroscience 3*, 1 (1991), 71–86.