Parameterization and Remeshing Over Dynamically Changing Domains

SMI 2006 Submission - Paper ID 59

Abstract

We present a method for incrementally maintaining a global, bijective mapping between an arbitrary triangular input mesh and a dynamically changing domain mesh by use of a meta-mesh. The domain mesh is derived from the input mesh through an arbitrary sequence of local mesh connectivity operations. The mapping can be maintained independent of the choice of metrics for driving the simplification process to derive the domain mesh. In addition, due to the dynamic nature of the mapping, user post-processing can be applied to adjust the final domains. We illustrate the usefulness of this parameterization in the context of several applications, including texture mapping and remeshing.

Keywords: Mesh parameterization, dynamic mapping, attribute transfer, remeshing and resampling.

1 Introduction

Progressive meshes [6] are one of the most widely used representations with applicability across the geometry processing pipeline, from model creation to rendering. The progressive mesh representation can be viewed as a family of meshes generated from a given input mesh through repeated application of a single operation: the edge collapse. Other applications, such as fitting, remeshing, and resampling may benefit from additional connectivity operations leading to improved meshes according to criteria such as approximation quality and regularity of connectivity [13, 18].

In this paper we focus on establishing a global parameterization between two meshes one of which is derived through connectivity operations from the other. We refer to the given input mesh as the fine mesh and to the coarser changing mesh derived from it as the domain mesh. In general, establishing a bijective parameteric mapping between the two is not straightforward. We focus on this problem as we consider that such mappings are very useful in many applications. For example, if the input mesh has certain properties (texture, color, or other attributes), a mapping is needed to transfer them to the domain mesh. Conversely, if certain computations are more readily performed on the simpler domain mesh, then a mapping to the input mesh is necessary in order to transfer the results from the domain mesh to it. This is the case, for example, in texture mapping applications in which it is easier to paste given images onto a coarse domain or in model decomposition scenarios [10] in which it is typically more convenient to segment model parts on a simpler mesh and then transfer the partition to the detailed mesh. The availability of a mapping between a coarse and a fine mesh is also at the basis of robust remeshing and resampling methods as it circumvents error-prone strategies such as those based on normal shooting [13].

1.1 Our contribution

We present a method for generating a continuous bijective mapping between an input mesh and a mesh derived from it through connectivity modifications and for dynamically updating this mapping as new operations are performed.

We introduce a novel framework for computing an atlas of charts corresponding to regions of the input mesh parameterized over faces of the domain mesh. As the domain mesh changes as a result of local operations, the affected charts are updated on-the-fly without the need for re-computing the entire mapping. The method guarantees a global continuous bijective mapping between the input mesh and the domain mesh. We do not impose any restrictions on the topology of the input mesh. The parameterization takes into account model features and the resulting maps ensure preservation of tagged features.

In addition to the parameterization itself, we describe a novel combination of metrics to drive the connectivity operations for the purpose of remeshing with semi-regular connectivity. It includes a parameterized shape criterion that aims for optimal triangle shapes vis-a-vis various remeshing goals (i.e., triangular or quadrilateral). The criterion favors the generation of a graded tessellation, for high-quality transitions between regions of high and low curvature.

We also allow the option of combining the automatic selection of connectivity changes to be with user intervention to adjust the results. The ability to maintain the parameterization after user intervention is one of the main strengths.
of our technique. It is well known that good stopping criteria for automatic progressive mesh generation are difficult to tune. They are mainly based on the number of operations or on thresholds for acceptable error values. Having the option to touch-up an automatically-generated domain while maintaining a mapping to the input model is, to the best of our knowledge, a novel capability that our method provides.

1.2 Related Work

Methods for establishing parameterizations between arbitrary meshes have recently emerged ([12, 17]). However, they are not readily applicable in the context of dynamically changing connectivity as they typically involve heavy computations which cannot be easily updated as local mesh operations are performed. Unlike these methods which focus on establishing a mapping between unrelated meshes, we aim to find a mapping between two related meshes. In our case we have the benefit of an existing high-quality initial parameterization (the identity map) and a relationship between meshes derived from one another through local connectivity changes.

The usefulness of mappings between related meshes has been recognized by many authors. Lee et al. [14] were among the first to describe a method for parameterizing meshes obtained by decimation from a given mesh - MAPS - and illustrated the resulting parameterization in the context of a remeshing application. The more recent parameterization work of Khodakovsky et al. [11] builds upon the MAPS framework. As in MAPS, we generate the parameterization during the process of deriving a new mesh. There are several important novelties in our technique with respect to MAPS: (a) we allow for various types of operations besides simplification / decimation, thus increasing the applicability of the method, (b) instead of successively computing the mapping of input vertices onto the domain mesh, we introduce a novel decomposition of the input mesh into charts; the charts are updated dynamically as connectivity changes are applied and once they are computed they allow for very efficient point location - a frequently performed operation in applications that involve resampling of the original mesh (e.g., for remeshing, transfer of properties), (c) we support derivation of polygonal meshes with quad faces (this is useful for applications such as editing and analysis that prefer quad elements for their tensor-product nature).

Sander et al. [16] describe the parameterization of progressive meshes with applications to texture transfer. In this case, the input mesh is first decomposed into charts, the simplification process follows taking into account the initial charts and the corresponding parameterizations, and lastly the chart parameterizations are optimized in another pass. The resulting parameterization is computed once and cannot be dynamically updated if additional edge collapses are performed. In fact, the edge collapse sequence is prescribed by the process of optimizing texture deviation).

There are multiple examples in recent literature that rely on progressive meshes for various applications. Besides the ubiquitous progressive mesh representation [6], such representations are useful in remeshing [14, 13, 18] (the latter offers an example in which a more complex set of operations is involved besides edge collapses), resampling [13], adaptive refinement [19], and finite element analysis.

2 Dynamically Changing Domains

We define a set of atomic mesh connectivity operations such that the parameterization is dynamically updated following each applied operation. The sequence of operations can be chosen independently of the parameterization, and the mapping will be valid after each atomic operation. Connectivity operations over the changing domain mesh can be driven automatically by an energy-driven simplification method similar to QEM[3], through user interaction, or a combination of the two. The dynamic nature of the domain mesh allows the user to customize the layout of the parametric patches corresponding to the domain mesh faces in order to fit the needs of their application.

2.1 Definitions

Our method builds and maintains three types of meshes (Fig 1): the fine input mesh $M^F$, the coarse domain mesh $M^D$, and for each triangular face $t^D \in M^D$ a chart mapping submesh $M^F_t$ that maps to and from a set of clipped triangles of $M^F$ corresponding to the image of $t^D$ over $M^F$.

We represent these three mesh types using an extended half-edge mesh topological structure that is a variant of the quad-edge mesh [5]. The half-edge mesh contains vertices, faces, half-edges, and wedges. An edge is made up of two opposite pointing half-edges. Each half-edge $he$ is associ-
An edge tag of either a feature tagged mesh \[8\], an edge \(e\) of the mesh has an edge tag of either smooth or sharp. Interior edges default to having a smooth tag, while edges on the boundary of an open mesh and additional edges specified by the user have sharp tags. For a point in the middle of a sharp edge, the local parameterization is split into two distinct half-discs, one for each side of the edge. While the local parameterization of a point on a smooth edge is a full-disc. A vertex \(v^F\) derives its vertex tag based on the number of sharp tagged incident edges.

- A smooth \(v^F\) has zero sharp edges.
- A dart \(v^F\) has one sharp edge.
- A crease \(v^F\) has two sharp edges.
- A corner \(v^F\) has three or more sharp edges.

A wedge [7] is a fan of faces incident to a vertex separated by only smooth edges and terminating at sharp edges, if present. The number of wedges around a vertex is a function of the number of sharp edges incident on the vertex. A smooth vertex has a single wedge that has a full disc local parameterization. A crease vertex has one or two wedges that each have a half-disc local parameterization. The local parameterization of each wedge around a corner or dart vertex is independently treated as a sector with angle \(0 < \theta < \pi\).

The fine mesh \(M^F\) is a static arbitrarily-connected, \(2\)-manifold triangle mesh with tagged features.

The domain mesh \(M^D\) is a triangle mesh that is initialized as a copy of \(M^F\) and is dynamically altered through a series of mesh connectivity operations. In order to maintain the chart mapping meshes, each vertex \(v^D \in M^D\) must correspond to a unique vertex \(v^F \in M^F\). Each \(t^D \in M^D\) has an associated chart mapping submesh \(M^C_t\).

A chart mesh \(M^C_t\) has the connectivity of a clipped submesh of \(M^F\) corresponding to the overlap mapping of the domain triangle face \(t^D \in M^D\). A region \(r^C \in M^C_t\) is a convex face with between 3 and 6 sides corresponding to a subarea of a fine triangle \(t^F \in M^F\) (Fig 5). It is possible to have multiple \(r^C\’s\) within a single chart that correspond to distinct subregions of the same \(t^D\). The parametric nature of \(M^C\) as well as the clipped edges require a different vocabulary of vertex and edge tags as those of \(M^F\) and \(M^D\).

A chart vertex \(v^C \in M^C_t\) corresponds to a unique fine vertex \(v^F\), except in the case of a clipped vertex where no such fine vertex exists. Each \(v^C\) contains parametric coordinates \((u, v)\) that are barycentric coordinates within a single \(t^D\), but take on arbitrary values when multiple \(M^C_t\’s\) are affinely mapped to a common parametric plane. Chart vertices are classified into five types (Fig 1(b)):

- An interior (1N) \(v^C\) is the image of a smooth \(v^F\) inside \(t^D\).
- A corner (CO) \(v^C\) is the image of a pair \(v^F\) and \(v^D\) on the intersection of 2 or more \(t^D\’s\).
- A boundary (BO) \(v^C\) is the image of a smooth \(v^F\) on \(t^D\).
A chart edge \( e^C \) corresponds to subsegment of a fine edge \( e^F \) and/or a domain edge \( e^D \). Each half-edge \( he^C \) maintains a parameter value \( s \in [0, 1] \) measured from the origin fine vertex of its corresponding \( he^F \). A CL edge \( e^C \) has parametric length \( |\Delta s| = (1 - s_a) - s_b \). The value of \( s \) is updated during edge clipping and merge steps. Each half-edge is one of the following four types (Fig 1(b)):

- An interior (IN) \( he^C \) is the image of a smooth \( he^F \), interior to \( t^D \), and incident to an IN, CO, BO, or CR \( v^C \).
- A clipped (CL) \( he^C \) is the image of a smooth \( he^F \), interior to \( t^D \), and incident to a CL \( v^C \).
- A boundary (BO) \( he^C \) is the image of an overlapping pair of \( he^F \) and \( he^D \), on the boundary of \( t^D \), and incident to a CO, BO, or CR \( v^C \).
- A domain (DO) \( he^C \) is the image of a \( he^D \), on the boundary of \( t^D \), and incident to a CO, BO, or CL \( v^C \).

There are two important properties of chart meshes that allow them to maintain the bijective mapping while \( M^D \) is undergoing arbitrary connectivity changes. First, the parameterization and connectivity of two \( M^C \)'s corresponding to two neighboring \( t^D \)'s along their shared \( e^D \) stays fixed outside of atomic mesh operations involving that pair of \( t^D \)'s. This ensures \( C^0 \) parametric continuity across patch boundaries. The second property is that all regions \( r^C \) are convex and non-overlapping.

2.2 Method Overview

The following is an outline of the processing necessary for each atomic operation to maintain the bijective mapping between the meshes \( M^F \) and \( M^D \). In a preprocessing step over the fine input mesh \( M^F \), edge splits are applied to all smooth edges connecting two non-smooth vertices to ensure that all atomic operations can be performed without reducing any \( t^F \) to zero area in the parameterization. The parameterization is then initialized by copying \( M^F \) to \( M^D \) and constructing an identity chart mesh \( M^C \) for each triangle \( t^D \). We consider six types of atomic mesh operations that can be applied to alter the domain mesh and the parametric mapping: edge collapses, edge splits, edge flips, vertex jumps, wedge smoothing, and edge smoothing (Fig 3). The transition from Fig 2(a) to Fig 2(h) shows a domain edge collapse in the mesh \( M^D \). For each atomic operation, we first identify the neighborhood of affected \( t^D \)'s (Fig 2(a)) and their corresponding chart meshes (Fig 2(b)). We compute a common chart by affinely mapping all affected charts into a common 2D parameter plane (Fig 2(c)) and merging the meshes together (Fig 2(d)). The parametric positions of the chart vertices are relaxed to smooth the parameterization and to avoid overlapping images of fine triangles (Fig 2(e)).

This new common chart provides a local parameterization in which the connectivity modification of the atomic operation is performed. The resulting \( e^D \)'s are introduced as clipping chords across the common chart (Fig 2(f)). Lastly, the new chart meshes are copied and pruned out of the common chart (Fig 2(g)) and transformed and assigned to the resulting \( t^D \)'s (Fig 2(h)).

2.3 Atomic Connectivity Operations

After each atomic operation on \( M^D \), the local faces are processed and the corresponding charts are updated to conform to the newly generated faces.

The edge collapse operation is useful for removing unnecessary faces from the tessellation for the purpose of mesh simplification. One endpoint is collapsed onto the other as shown in Fig 3(a)-(c).

The edge split operation allows the mesh to be refined in places where additional vertices are needed. A vertex \( v^D \) is inserted along the edge and is connected to the opposite vertices of the adjacent faces as shown in Fig 3(d), and it is associated with a vertex \( v^F \) from the common chart.

The edge flip operation swaps the diagonal of a quadrilateral consisting of two adjacent triangles as in Fig 3(e). This is typically done to improve mesh quality (for valence and face aspect ratio optimization).

Vertex jump allows derived mesh vertices to move through local relaxation. This operation is useful not only to smooth the parameterization, but also to optimize the shape of adjacent derived mesh faces according to shape measures discussed in section 4. The vertex \( v^D \) is snapped to a fine vertex \( v^F \) whose image is present in the common chart.
Flattening quad patch on top of the two triangle patches. The neighborhood depends on the type of operation: for edge processing, we accommodate a postprocessing mains by pairing adjacent triangles [1]. To support such operations we use a hinge map (i.e., we rotate one of the triangles around the common edge until it becomes coplanar with the adjacent triangle). If the resulting planar mesh is convex, we use this mapping. Otherwise, we simply compute for the vertices of the 1-ring to the unit circle, such that arcs subtended by edges opposite are proportional to the edge lengths . We require the convexity condition in order to avoid flips after connectivity operations are performed (Fig 3 (c)) and to ensure that there are no fine triangle foldovers after relaxation.

To flatten a 1-ring of faces sharing an edge we use a hinge map (Fig 3(f)) and for which the combination of the quadric error and shape error metrics is minimized. After the vertex is repositioned in parameter space, the charts are updated with respect to the new position.

Wedge smoothing performs parameter smoothing in the neighborhood of a domain mesh vertex . As with the vertex jump, the common chart vertices are relaxed in parameter space and may migrate within the wedge across edges, but reassociates with the same afterward.

Edge smoothing performs smoothing across a domain mesh edge. This is the same as an edge flip without changing the connectivity of .

After each edge collapse, edge split, and edge flip, vertex jump operations are applied first to all the domain mesh vertices in the neighborhood of the edge that was modified. If a jump changes the location of the vertex, it is followed by wedge smoothing operations in its local neighborhood. The goal of wedge smoothing is to improve the parameterization through local smoothing (Fig 4). If the wedge cannot be flattened into a convex region (see section 2.4), then wedge smoothing is not performed. Edge smoothing between pairs of faces in the wedge are performed instead.

The operations above preserve the triangular nature of the tessellation (i.e., the domain mesh is a triangle mesh). For certain applications it is useful to form quadrilateral domains by pairing adjacent triangles [1]. To support such processing, we accommodate a postprocessing edge removal operation that pairs two ’s and builds a logical quad patch on top of the two triangle patches.

2.4 Common Chart Computation

Flattening Each atomic mesh connectivity modification is accompanied by an update of charts. All updates require the computation of a local common chart. The affected neighborhood depends on the type of operation: for edge collapse, vertex jump, and wedge smooth it consists of the 1-ring of the central domain vertex, whereas for edge split, edge flip, and edge smooth it contains only the two faces adjacent to the edge.

To flatten a 1-ring of faces around a given domain vertex, we aim to reduce distortion by finding a planar mapping that compromises between conformality and isometry, while guaranteeing a valid bijection after the operation is performed. To achieve this, we compute a mapping of the vertices (Fig 2(c)) and corresponding chart entities are in the parameter plane (Fig 3(b)). If the resulting polygon formed by is convex we use the mapping induced by as our parameterization. Otherwise, we simply compute for the vertices of the 1-ring to the unit circle, such that arcs subtended by edges opposite are proportional to the edge lengths . We require the convexity condition in order to avoid flips after connectivity operations are performed (Fig 3 (c)) and to ensure that there are no fine triangle foldovers after relaxation.

Merging Once a mapping is available, the corresponding chart meshes are flattened onto a common plane (Fig 2(c)). Note that the chart edges and vertices along the shared domain edges match. Next the chart meshes are merged into a single common chart (Fig 2(d)). Matching are removed along with the two DO ’s, and the two are merged into a single segment with the parameters updated. Matching are tagged as IN, and matching are tagged as IN and their references to the are removed. Matching are full 1-ring, i.e., in (Fig 2(e)), are also tagged as IN. After
merging, all references to \( e_D \) internal to the common chart have been removed, so these edges can be safely altered by the connectivity operation.

**Relaxation** The initial positions of chart mesh vertices in the plane is computed based on their barycentric coordinates within each domain mesh face using the mapping \( q \). An iterative relaxation procedure is then used to optimize these positions (keeping CO, BO, and CL vertices fixed) using the Floater mean value weights \([2]\) (Fig 2(e)). The weights are computed over the clipped 1-ring of the IN \( v_C \) using the clipped edges incident to it and adjusting the length of the \( e_F \)'s accordingly. CR \( v_C \)'s derive their weights from their two neighbors along the crease boundary edge only. Since the boundary of the parameterization is convex, the iterative relaxation is guaranteed to converge to a valid configuration (no foldovers).

**Clipping** The connectivity modification is performed on the domain mesh, and the resulting \( e_D \)'s generate clipping chords across the common parameter mesh (Fig 2(f)). Each domain chord \( e_D \) is added to the mesh by shooting a planar ray from an origin \( CO \) to \( v_C \) to the opposite endpoint. Colinear IN \( v_C \)'s are tagged as BO, and colinear \( e_D \)'s are tagged as BO and are updated to refer to the domain edge \( e_D \). An intersection with an IN or CL \( e_C \) splits the edge segment into two CL \( e_C \)'s, generates a CL \( e_C \) at the point of intersection, and generates two DO \( e_C \)'s which split the two \( r_C \) convex regions adding two additional convex regions. Along with calculating the position of the intersection \( e_C \), the fine edge \( s \) parameters on the CL \( e_C \)'s are updated.

**Pruning** Next the common chart is partitioned and pruned into the charts associated with the resulting \( t_D \)'s. For a \( t^D \) with three edges \( e_D^1, e_D^2, e_D^3 \), the BO and DO \( e_C \)'s that refer to these domain edges form the perimeter of the chart mesh \( M_C \), and a flood filling algorithm respecting these \( e_C \)'s as boundaries is used to prune the chart. Note that the chart vertices and edges along the chord edges will be compatible on neighboring \( t_D \)'s and that no chart edges or vertices on the boundary of the common chart are affected by this process. Lastly, each pruned chart is affinely mapped to barycentric coordinates in its new \( t^D \) (Fig 2(g)).

**Regions** Parametric mappings of arbitrary topology objects can create bend fine triangles over a collection of domain faces including fine triangles that map to multiple regions on one domain triangle. However, a single region is a convex polygon with between three and six sides. Fig 5 shows the six possible region types that can be created by a fine triangle intersecting with a convex domain. A single region will intersect with at most three edges of the common chart’s convex domain, hence a single region is the result of a triangle-triangle intersection.

Regions that intersect multiple domain edges (Figs 5(d)-(f)) split the common domain into convex subdomains at edges like \( e_D^1 \) or chords like \( e_F^1 \) in Fig 5(e). Due to the convex hull property of our choice of relaxation weights, vertices topologically connected to a subdomain will converge to a location in that subdomain. Because these subdomains are convex, the relaxation process will avoid fine triangle foldovers \([2]\).

### 3 Applications

We describe several applications that benefit from the availability of a parameterization as described in section 2.

**Level-of-detail (LOD) generation** Our dynamic parameterization method provides the means for establishing a mapping between the input mesh and an intermediate mesh in the LOD hierarchy computation. MAPS-type approaches \([14]\) rely on a hierarchical structure to maintain parameterization information and require traversal of the hierarchy (along with expensive point location queries) every time an evaluation operation is performed. In contrast, we provide an atlas of charts that describe a direct mapping which can be evaluated very efficiently at arbitrary locations in the parameter domain. Fig 6 illustrates the use of this parameterization to transfer attributes from the input mesh to a coarser mesh. Note that we used a mixture of edge connectivity modifications not just edge collapses to derive the new mesh.
applying refinement operations in reverse order of the coarsen-...mation onto the input model is typically accomplished by pasted onto the domain mesh obtained by simplification, the applications [10]. For texture mapping, once the texture ispler domain or in model decomposition and skeletonization scenarios in which a set of textures are first applied to a sim-...in order to perform dyadic resampling [14, 9, 13]. The parameterization method described in this paper supports this process. First, the mapping between the base domain and the input mesh is readily available (no need for normal shooting as in [13]) and it is efficient to compute (no hierarchy traversal and expensive point location computations as in[14]). Second, due to its dynamic nature, the mapping can be adjusted as needed, for example if the domain obtained automatically needs to be corrected through user intervention. In Fig 11 the user modifies the base domain by interactively adjusting the mesh. The parameterization is dynamically updated and then used for remeshing.

4 Domain Mesh Generation

In this section, we take a closer look at the process of generating a domain mesh through local connectivity operations. While we recognize that heuristics and metrics to drive this process are as varied as the applications themselves, we wish to explore metrics that focus on the requirements of semi-regular remeshing, as they are quite demanding and can benefit a number of other applications. Thus, we seek to generate domain meshes such that:

- They fit the input mesh within a given tolerance
- Have well-shaped faces (ideally equilateral in the case of triangles, and rectangular in the case of quads)
- Have good connectivity (ideal interior vertex valence is six for triangles and four for quads)

We combine several metrics to quantify these properties.

The Quadric Error Metric (QEM) [3, 4] quantifies the fit to the input mesh. To allow portions of the input mesh separated by features to be processed without artificially affecting each other, we compute the quadric errors per wedge and along feature curves. For features we compute the fit to the original feature curves.

The Shape Error Metric quantifies element shape with respect to ideal shape. For triangle output, we use the following shape metric based on the Frobenius norm [15]:

$$\alpha = 4\sqrt{3\frac{\text{area}(T)}{||a||^2}}$$

where area(T) and ||a||^2 are the area and the sum of square edge lengths of the triangle. We have $\alpha \in [0, 1]$ and $\alpha = 1$ for equilateral triangles. Fig 7 shows the deviation from ideal when two vertices of a triangle are kept fixed and the third moves in the plane. The nice symmetric pattern of this metric allows us to select meaningful thresholds so as to avoid poorly shaped elements.

For quad-dominant remeshing, we have investigated shape metrics leading to triangles such that when they are later paired with neighbors lead to ideal quads. For this we consider pairs formed by a given triangle with each of its neighbors and we compute a quad shape error based on the four sorted $\alpha$ values (with $\alpha_1$ being the largest) for the four corner triangles of a quad:

$$\beta = \frac{\alpha_3\alpha_4}{\alpha_1\alpha_2}$$

where $\beta \in [0, 1]$ with $\beta = 1$ for rectangular shapes. We combine this metric with a planarity constraint that rejects
Fig 7. Plot of shape error metric representing deviation from ideal equilateral triangle as the base of the triangle is fixed along x-axis and the opposite vertex moves in the plane.

The Valence Error Metric quantifies deviation from ideal vertex valences. To accommodate features, we measure it per wedge as the square difference between the actual and ideal wedge valences. The table below lists ideal valences for various types of wedges.

<table>
<thead>
<tr>
<th>Wedge type</th>
<th>Smooth or dart</th>
<th>Crease</th>
<th>Corner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tri</td>
<td>6</td>
<td>3</td>
<td>1.2</td>
</tr>
<tr>
<td>Quad</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

To combine the above metrics we transform them so they take values in $[1, \infty)$ with a value of 1 corresponding to the ideal case. We multiply the values together to obtain a cost per edge and we prioritize edges accordingly. We drive the coarsening process using a combination of edge connectivity modifications. Similar to [18], we distinguish between edge flips that result in a local valence improvement and flips that improve the valence of a given vertex at the expense of one of its neighbors, thus effectively causing the valence problem to drift over the mesh (possibly towards a location where it leads to local improvement).
Repeat
  If valid valence-improving edge flip exists
  perform edge flip
  Else If valid edge collapse exists
  perform edge collapse
  Else If valid drifting edge flip exists
  perform edge flip
  Locally update edge costs
Until (no valid operations exist)

Figs 10 and 13 show a comparison of results obtained with our technique versus QEM simplification [3]. The color encodes the quality of the triangles (blue corresponds to ideal shape, gradation to red measures deviation from it). A high-quality parameterization is built on-the-fly and is subsequently used for remeshing and texture mapping.

**5 Results**

**Parameterization evaluation.** The main benefit of our parameterization is the ability to compute correspondences between the input and the domain mesh as connectivity modifications are applied to it. The input mesh can be trivially evaluated over the domain mesh using the chart meshes and barycentric coordinates. The inverse mapping from an arbitrary position inside a domain mesh triangle $t_D$ with barycentric coordinates $(u, v)$ requires a local point location within the corresponding chart mesh. We use mean value coordinates [2] to identify the exact location of the point being evaluated inside a fine (possibly clipped) triangle (Fig 1(b)).

**Domain mesh generation.** The strengths of our parameterization become evident in conjunction with progressive mesh generation. We have selected to focus on a process that drives the combination of various types of connectivity modifications giving special attention to fit, element shape, and vertex valence. We emphasize that this is only one of many such processes that can benefit from parameterizations that can be updated on-the-fly. With respect to our choice, we discuss several implementation details.

To preserve the topology of the input mesh we only allow connectivity operations that do not alter topology of the input mesh. Before every edge operation, we verify that a modification along that edge would not lead to a change in topology by checking that no two vertices are connected by more than one edge if the operation would be performed. In addition, we separately test and prevent tetrahedron configurations from being collapsed. To ensure consistency of tagged features, we split smooth edges linking tagged vertices in a preprocessing step, we constrain edge collapses involving endpoints of tagged edges to be performed only along the feature curve, and we prevent tagged edges from being flipped.

For the automatic domain mesh generation we use a stopping criterion based on thresholding errors for quadric error, shape, and valence metrics. When there are no edges left in the mesh that do not violate these thresholds, the process stops. Of course, we allow the user to interactively perform additional adjustments, as needed.

**Parameterization quality.** We measure the quality of the parameterization using the distortion metric of [16]. The results obtained are tabulated in Table 1. In addition to the $L^2$ stretch metric, we are also reporting the Peak Signal to Noise Ratio (PSNR) for the remeshed models with 0, 2, and 4 subdivision levels, respectively.

**Performance.** We measure the performance of our method in terms of the average time to update an existing
parameterization after a connectivity operation (Table 2). This time includes the identification of affected charts, their merging and flattening onto a common plane, the local smoothing of the parameterization, performing the operation, and updating the charts. The total time for generating a domain mesh depends on the number of operations that are performed to arrive to that mesh. In the case of automatic generation, the time is dominated by priority queue updates in the early stages of the process, and by chart mesh merges and updates as the charts grow larger (assuming derived meshes get coarser).

Figs 11, 12, and 13 illustrate our technique and its applications. Fig 11 shows the mapping computed in the presence of sharp features. The results of the automatic mesh generation process in (b) are adjusted through user interaction in (c). The mapping is updated accordingly on-the-fly. Domains derived using triangle and quad shape metrics are computed and used for resampling in (d) and (e). Fig 12 illustrates applications to texture mapping using the inverse mapping from the derived domain to the input mesh. Fig 13 shows comparatively the shape distributions obtained with QEM alone (b) versus using our combined metrics (c). Remeshing results over the domain computed with our method are shown in (d).

<table>
<thead>
<tr>
<th>Model</th>
<th>Input (derived) size (#v)</th>
<th>Distortion</th>
<th>Remesh PSNR (# levels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cow</td>
<td>2911 (234)</td>
<td>1.046</td>
<td>42.93 56.33 71.57</td>
</tr>
<tr>
<td>torus</td>
<td>5884 (148)</td>
<td>1.010</td>
<td>41.16 61.92 79.57</td>
</tr>
<tr>
<td>fandisk</td>
<td>6500 (229)</td>
<td>1.004</td>
<td>53.61 75.39 93.28</td>
</tr>
<tr>
<td>cat</td>
<td>7289 (201)</td>
<td>1.010</td>
<td>43.25 59.24 78.77</td>
</tr>
<tr>
<td>david</td>
<td>20000 (356)</td>
<td>1.041</td>
<td>43.79 53.79 70.35</td>
</tr>
</tbody>
</table>

Table 1. Parameterization quality.

<table>
<thead>
<tr>
<th>Model</th>
<th>cow</th>
<th>torus</th>
<th>fandisk</th>
<th>cat</th>
<th>david</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg op time (s)</td>
<td>0.055</td>
<td>0.067</td>
<td>0.057</td>
<td>0.099</td>
<td>0.142</td>
</tr>
</tbody>
</table>

Table 2. Performance.

6 Conclusions

We have described a method for establishing a bijective parametric mapping between a given input mesh and a dynamically changing domain mesh derived from it through an enhanced set of local connectivity modifications. Once computed, the map can be efficiently evaluated at arbitrary locations and it can be updated to account for new connectivity modifications. These can be performed automatically or can be induced by the user. The mapping takes into account input model features. We illustrated our approach in the context of several applications. An interesting venue for future exploration is the extension of such dynamic parameterizations to more general mesh editing operations, including not only connectivity modifications, but also geometry changes.

References

Figure 11. (a) Input model with tagged features. (b) Automatically derived mesh. (c) Mesh in (b) after user has cleaned up faces (e.g., inside the circled areas). (d) Semi-regular triangle remeshing over the domain in (c) generated using triangle shape metric. (e) Quad semi-regular remeshing over a domain generated using quad shape metric.

Figure 12. (a) Input model with tagged features. (b) Derived coarse mesh with chart structure superimposed (gray lines indicate the projections of the input mesh faces onto the coarse mesh). (c) Texture mapping onto coarse mesh. Transfer of texture from coarse to input mesh using the parameterization.

Figure 13. Comparison of domain mesh generation metrics. (a) Input model. (b) Coarse mesh derived through simplification using QEM alone. (c) Coarse mesh using the mesh generation process described in section 4. (d) Semi-regular remeshing over the domain in (c).