Consistent Adaptive Remeshing of Multiple Disk-Like Surfaces

Mauricio Cele López Belón
Huddle Group S.A. Buenos Aires, Argentina.
mauricio.lopez@huddle.com.ar
WWW home page: http://www.huddle.com.ar

Abstract. We present a novel method for computing the consistent adaptive remeshing of multiple disk-like surfaces with hard constraints. The quality of the remeshing is assured by minimizing the curvature error metric on all surfaces simultaneously. In order to generate meshes of relatively low triangle count, we adaptively optimize several red-green subdivision surfaces. The proposed method is very simple and easier to implement than most consistent remeshing methods. It is also useful for many applications such as mesh registration, compression and morphing, multiresolution rendering and texture transfer. We demonstrate the mesh morphing application between disk-like meshes with feature-point correspondences specified by the user.

Keywords: Computer Graphics, Computational Geometry and Object Modeling

1 Introduction

Remeshing is a fundamental step for efficient mesh processing. Rendering, texture mapping, morphing, animation, deformation, among others, are improved by remeshing. Remeshing is about modifying the sampling and connectivity of geometry to generate a new mesh which approximates the original mesh. The state of the art algorithms of remeshing involves computing a parameterization of the mesh. Parameterize a mesh is about computing a bijective linear mapping $F$ between the surface in $\mathbb{R}^3$ and the parameter surface in $\mathbb{R}^2$ space. The parameterized surface can then be redefined, to give them other connectivity and point density in parameter space, then the new parametric surface is resampled using the mapping $F^{-1}$ to create a new approximated surface in $\mathbb{R}^3$.

Given a number of different surfaces in $\mathbb{R}^3$, we can apply the remeshing procedure to all of them to give them the same connectivity and point density. We call this procedure Consistent Remeshing. Of course, when more than one surface is remeshed the user can define correspondence points or feature-points among surfaces, for instance, in a morphing sequence, an animator may want that a given point on the nose of a dog must correspond to a given point on the nose of a cat.
We propose a novel method to compute the consistent adaptive remeshing of multiple meshes with hard constraints. Our method allows the user to choose several feature points on each surface. These feature points are used to compute a common coarse base mesh to all the input surfaces in parameter domain. The common base mesh is then adaptively refined to minimize the curvature error on all surfaces. The feature points are respected by the common mesh, so that the same vertex in the shared connectivity is consistently mapped to all remeshed shapes. For example, the nose of a bunch of input head models can be selected, so that the same vertex in the shared connectivity is consistently mapped to the noses of all shapes. To reach this goal we define a proper warp function in parametric space that enforce the constraints.

In order to generate meshes of relatively low triangle count, we optimize several red-green subdivision surfaces in a novel way. Red-green subdivision allows adaptive or uniform subdivision of meshes without introducing T-junctions, also produces meshes with regular shaped triangles, therefore computations are much more stable on them.

2 Related Work

2.1 Surface Parameterizations

Mapping a surface to a suitable planar domain is usually called surface parameterization. Surface parameterization itself has been extensively studied, and has been playing an important role in modern graphics, modeling and geometric processing pipeline. A thorough review of surface parameterization is beyond the scope of this work. We refer the reader to [8], [19], [6] and [11] in order to survey the state of the art techniques in this field.

2.2 Surface Mapping

Consistent remeshing of multiple surfaces can be seen as a Surface Mapping problem. Surface Mapping is a fundamental problem in the computer graphics/modeling field. It can be defined as the computation of a continuous one-to-one function $F$ from one surface $M_i$ to the second surface $M_j$. A thorough survey is beyond the scope of this work. We are briefly reviewing the most related work only.

Computing inter-surface maps between star-shaped surfaces without feature points correspondences was addressed by Kent et al. [13]. Later, Kanai et al. [12] developed a method to map all genus-zero surfaces easily but it only allows one feature point specified by users. The work of Zöckler et al. [20] allows multiple feature points specified by users. Feature correspondences are enforced in parametric domain applying a foldover-free warping. Alexa [1] proposed to match multiple feature points between genus-0 surfaces but no bijectivity is guaranteed and hard constraints may not be fully enforced. More recently, Asirvatham et al. [2] used constrained spherical parameterization to map genus-zero surfaces
onto the sphere, the progressive mesh was used to get a simple base mesh and
to enforce constraints at certain positions on the sphere. This method allows
multiple hard constraint points between genus-0 surfaces.

For higher genus meshes, most techniques construct piece-wise consistent
parameterizations. Meshes are first consistently segmented into several disk-like
patches. A simplicial base mesh is constructed from such patches. Then each
patch is cross parameterized. Global parameterization is obtained by composing
the patches together and optimizing them. See for instance Lee et al. [16], Praun
et al. [17], Kraevoy and Sheffer [15] and Schreiner et al. [18].

2.3 Consistent Remeshing

There are two main approaches for generating meshes with consistent connec-
tivity:

- **Base-Mesh Subdivision**: Several methods (for instance [17] and [16]) are
  based in semi-regular refinement (one-to-four subdivision pattern), introducing
  as many levels of subdivision as necessary to capture the geometry of both mod-
  els. The advantage of the method is simplicity. It’s drawback is the dependence
  on the shape of base mesh triangles. The method also tends to require large
  triangle count to achieve acceptable accuracy (roughly factor 10 compared to
  input mesh sizes).

  Although our method is based on refinement, we are, at best of our knowl-
  edge, the first to use the red-green refinement (i.e adaptive refinement) to pro-
  duce consistent meshes. As we show later, our method does not produce meshes
  with large triangle count to achieve acceptable accuracy.

- **Overlay**: Another approach for generating common connectivity (for instan-
  ce [1], [18], [13], [12], [20] and [2]) is to intersect the two input meshes in the
  parameter domain, combining all their vertices and generating new vertices at
  edge-edge intersections. The method preserves exactly the input geometries but
  is not very robust and it is difficult to implement. Moreover, it increases the
  triangle count by roughly a factor of 10.(see [11] for more information)

2.4 Red-Green Refinement

Red-Green refinement was introduced by Bank et al. [3], for refinement of triangle
surfaces. Bank et al. method combines regular one-to-four subdivision pattern
(red), with irregular subdivision pattern (green). Regular refinement produces
four children triangles by connecting the edge mid-points of the parent. Irregular
refinement bisects or trisects a parent triangle according to the number of edge
mid-points introduced.

In order to meet stability conditions, only regular (red) triangles can be
refined. Using regular refinement only leads to uniform subdivision. Adaptive
subdivision is possible by applying irregular refinement to red triangles that lies
on the boundary of two consecutive resolutions, i.e. in order to maintain the mesh
conformity, creating the so called green closure.
Bank et al. method holds three properties: Conformity, Nestedness and Stability. Conformity states that not T-junctions are produced by the method. Nestedness derives from the fact that refinement is computed through subdivision, so each triangle in the base mesh is the root of a tree-like hierarchy of sub-triangles, therefore sub-triangles may share vertices with its parent(s). Stability states that the method should generate stable triangulations. There are several metrics to measure triangulation stability. The most popular technique is based on the measurement of triangle angles. Stability assures that triangles have regular shapes, i.e. angles are close to $60^\circ$. Regular shaped triangles are very important to assure the numerical stability for PDE’s solvers. As a consequence, applications such as fluid simulation or mesh deformation will have stable solutions. Mesh stability is also important for further Geometry Processing treatment and even for rendering, because smooth meshes produces better attribute interpolation than scattered ones.

In this paper, the adaptive refinement is conducted by a numerical solver, i.e. the minimization of some error (energy) function. Triangles that are to be refined are selected according to some minimization criteria. Such triangles are marked for refinement, using a method called \texttt{MarkTriangle}. After all required triangles are marked, we perform the adaptive refinement using a method called \texttt{RedGreenRefinement}.

3 Consistent Adaptive Remeshing

3.1 Algorithm Stages

Given a number of disk-like manifold meshes $\langle M_0, M_1, ..., M_{n-1}, M_n \rangle$, our strategy for computing consistent meshes $\langle M'_0, M'_1, ..., M'_{n-1}, M'_n \rangle$ is as follows:

- Compute an initial parameterization $U_i$ for each mesh $M_i$ and define the bijective map $F_i(M_i) = U_i$.
- Based on corner and feature points $\langle C_i, P_i \rangle$, compute a set of consistent base meshes $B_i$ in parameter domain. Each base mesh $B_i$ is used to create a red-green mesh $S_i$, one for each mesh $M_i$.
- Apply consistent adaptive refinement to each mesh $S_i$ in order to minimize the error function $E_N$ of all meshes.
- For each red-green mesh $S_i$, compute the consistent mesh $M'_i$ using the inverse map $F^{-1}_i$, such that $M'_i = F^{-1}_i(S_i)$. Finally, optimize $M'_i$ to improve the mesh quality.

3.2 Computing the Initial Embedding

A parameterization $U$ can be defined by assigning parameters $(s, t) \in U$ to each vertex of the surface and interpolating them linearly within the triangles. This defines a linear mapping from points in the triangulation ($R^3$ space) to points in the parametric domain ($R^2$ space). To construct a homeomorphism, the parameters $(s, t)$ have to be chosen in such a way that the mapped edges do
not intersect each other in parameter domain; otherwise the mapping would not be bijective.

In this stage, our goal is to compute a bijective linear mapping \( F_i(M_i) = U_i \), and its inverse mapping \( F_i^{-1}(U_i) = M_i \).

Several existing methods can be used to reach this goal (e.g. [8], [19]). Boundary fixed methods, such as Barycentric mapping, allows us to fix the boundary parameterization to be a convex polygon in parameter domain. These techniques are convenient for us because we can easily match boundary points of different surfaces in parameter space. Please note that the proposed technique does not depend on the parameterization method. In this paper we choose to fix the boundary to the unit square (again just for the sake of simplicity but without loss of generality).

The user must pick just four corner points per input mesh \( M_i \). We map these corner points to corners of the unit square in parameter domain. Then we parameterize the mesh boundary using the so-called chord length parameterization and the interior vertices using the Mean Value Parameterization (Floater et al. [7]).

Given the mesh \( M_i \) and its parameterization \( U_i \), define a piece-wise linear mapping \( F_T \) between them, such that for each point \( P \) in triangle \( \langle P_i, P_j, P_k \rangle \in M_i \) with barycentric coordinates \( \langle \alpha, \beta, \gamma \rangle \), the function \( F_T \) maps \( P \) to a point \( p \) in triangle \( \langle p_i, p_j, p_k \rangle \in U_i \) as follows:

\[
F_T : P \mapsto p = \alpha p_i + \beta p_j + \gamma p_k
\]

In a similar way, define a piece-wise linear mapping \( F_T^{-1} \), so that for each point \( p \) in triangle \( \langle p_i, p_j, p_k \rangle \in U_i \) with barycentric coordinates \( \langle \alpha, \beta, \gamma \rangle \), the function \( F_T^{-1} \) maps \( p \) to a point \( P \) in triangle \( \langle P_i, P_j, P_k \rangle \in M_i \) as follows:

\[
F_T^{-1} : p \mapsto P = \alpha P_i + \beta P_j + \gamma P_k
\]

Finally, the set of piece-wise mappings \( F_T \) defined over the triangulation compose the globally continuous mapping \( F \). The same holds for \( F_T^{-1} \) and \( F^{-1} \).

### 3.3 Computing the Consistent Base Meshes

For each input mesh \( M_i \), the user must define four corner-points in the boundary, denoted by \( \langle C_{i0}, C_{i1}, C_{i2}, C_{i3} \rangle \), and optionally, the user may define any number of feature-points on the interior of each mesh, denoted by \( \langle P_{i0}^i, P_{i1}^i, ..., P_{m-1}^i, P_{m}^i \rangle \).

Feature points must comply the following constraints:

- The same number of feature-points must be defined for all meshes.
- Inter-surface matches of feature-points must not produce fold-overs in the common base mesh (for example, if a user matches the left eye of one model to the right eye of another model, the common base-mesh will be invalid).
We define the unified feature-points as the linear combination of matched feature points of all input meshes. We start by computing the set of unified feature-points in the following way:

For each input mesh $M_i$, use the mapping $F_i$ to map the four corner-points and the feature-points into the parametric domain. Compute the set of unified feature-points $\langle \bar{p}_0, \bar{p}_1, ..., \bar{p}_{m-1}, \bar{p}_m \rangle$ by calculating a weighted average of matched feature points from all meshes, in parameter domain.

$$\bar{p}_j = \frac{1}{\sum_{i=0}^{n} \mu^i} \sum_{i=0}^{n} \mu^i p^j_i, \quad j = 1, ..., m$$

where $\mu^i$ is a weight defined by each mesh $M_i$. This weight is used to fine tune the contribution that feature-points of each $M_i$ would give to the unified feature-points. In the general case we can roughly set $\mu^i = 1$.

Making the Common Base-Mesh

We define a Common Base Mesh $B_c \subset U_i$ as a planar triangular mesh in parameter domain. For simplicity, the boundary of $B_c$ is the unit square. The user defined corner-points maps to corner points of the unit square and the unified feature-points, computed in the previous stage, maps to interior vertices. The mesh connectivity is given by the Delaunay Triangulation (DT) (Fortune [9]) of these points.

The common base-mesh $B_c$ is like a template that contains the shared connectivity of all base meshes, i.e. all base meshes are just a warped version of $B_c$.

Usually the simple Delaunay Triangulation of the unified feature-points does not produce good quality triangles. The common base mesh must have good quality triangles in order to produce good quality refined triangles. In this stage, our goal is to produce regular triangles in base mesh. By regular we understand triangles without sharp angles.

Since we are not able to modify the initial set of vertices $(\bar{p}, \bar{c})$ (i.e. optimize them), because they maps to user-defined feature points, we need to improve the triangulation quality by adding Steiner vertices. We use Delaunay Refinement techniques to improve the mesh quality (Constrained Delaunay Triangulation or CDT). We constrain the border of the unit square and then compute the CDT of the set of vertices. This algorithm inserts Steiner vertices where necessary until certain triangle quality measures are achieved (see Fig. 3 left).

Making Consistent Base-Meshes

In this stage we develop a set of base meshes $\langle B_0, B_1, ..., B_{n-1}, B_n \rangle$ in parameter domain. Each base-mesh correspond to one input mesh $\langle M_0, M_1, ..., M_{n-1}, M_n \rangle$. All base meshes will share the connectivity of the common base mesh $B_c$. We start the computation as follows:

For each set of feature points $p^i \in U_i$, trivially warp the common base-mesh $B_c$ to get the base-mesh $B_i$ in the following way: Replace all the unified feature-points $\bar{p}_k \in B_c$ with the feature points $p^i_k$.

In case the common base-mesh $B_c$ have Steiner vertices, we proceed with the following steps:
- Recompute the DT of the *unified* feature-points. The result is the temporary mesh $B'_c$.
- Compute the barycentric coordinates to Steiner vertices of $B_c$ w.r.t the triangles of $B'_c$.
- Replace the *unified* feature-points $\bar{p}_k \in B'_c$ with $p^i_k \in B_i$. The result is the temporary mesh $B''_c$. So $B''_c$ keeps the connectivity of $B'_c$ but has the feature-points of $B_i$.
- Use the computed barycentric coordinates to place the Steiner vertices in $B''_c$.
- Replace all the *unified* feature-points $\bar{p}_k \in B_c$ with $p^i_k \in B''_c$ and Steiner vertices in $B_c$ with Steiner vertices in $B''_c$. The result is the final mesh $B_i$. So $B_i$ keeps the connectivity of $B_c$ but has the points of $B''_c$ (Fig. 1 left: top and bottom, shows examples of final warping).

This procedure may lead to fold-overs in $B_i$ under certain circumstances where the matched feature points lead to different feature distribution on each mesh. To illustrate this scenario, consider for example a user selecting three feature points on two meshes, so that connecting them, they would form a triangle pointing upwards in one face and downwards in the other. This kind of distribution may lead to fold-overs.

![Fig. 1. Left: two consistent base meshes $B_0$ (top) and $B_1$ (bottom) warped from $B_c$ and optimized. Middle: final optimized $S_0$ and $S_1$ after consistent red-green subdivision. Right: resampled and optimized meshes $M'_0$ and $M'_1$.](image)

### 3.4 Consistent Remeshing

In this stage we apply a consistent remeshing to all meshes $\langle M_0, ..., M_n \rangle$, obtaining the new consistent meshes $\langle M'_0, ..., M'_n \rangle$. 
Definitions First, we define a Red-Green mesh $S_i$ as the 3-tuple $(B_i, \Sigma_i, \gamma)$, where $B_i$ is the base mesh, $\Sigma_i$ is the set of red-green refinements applied to $B_i$ and $\gamma$ is the set of rules that conduct the refinement. The set of modifications $\Sigma_i = \langle \Sigma_{i,0}, \Sigma_{i,1}, \ldots, \Sigma_{i,m-1}, \Sigma_{i,m} \rangle$ are arranged as a forest of modifications, i.e. each leaf triangle $t_k \in B_i$ is the root of a complete tree $\Sigma_{i,k}$ of red-green refinements.

We define a modification $\Sigma_{i,k}$ as a hierarchy of refinement operations arranged as a tree. The root of the tree is a leaf triangle $t_k$ of the base mesh $B_i$. A refinement operation consists of adding mid-points to triangle edges and then creating children triangles by connecting the mid-points using the red-green rule.

Then, we define the Warp function $W_{i,j} : S_i \mapsto S_j$. This function takes a red-green mesh $S_i$ as input and produces the warped red-green mesh $S_j$ that shares the connectivity of $S_i$. Firstly we must warp the base mesh $B_i$ to obtain $B_j$. In this step we simply replace the points of $B_i$ with points of $B_j$. Then we must hierarchically recompute all the mid-points of the red-green mesh $S_i$, starting from mid-points introduced in the base mesh $B_i$. We use the hierarchy of modifications $\Sigma_i$ to conduct the mid-points regeneration. The algorithm of Table 1 summarizes the idea.

![Algorithm 1. Warp function $W_{i,j} : S_i \mapsto S_j$](image)

This Warp function is very easy to implement and also is efficient. In fact, as you can see in the proposed algorithm, we just need to provide the base mesh of the target mesh. So the implementation is memory efficient, and therefore suitable to process large meshes. In addition, the warp function $W$ can be also used to warp the mesh to its original form $W_{j,i} : S_j \mapsto S_i$. 

---

**Table 1.** Warp function $W_{i,j} : S_i \mapsto S_j$
Finally, we define the error function $E$. For a given point $p$ in the planar red-green mesh $S$, find the 1-ring neighbors $p_j$, $j = 0, \ldots, N_v$. Use the inverse mapping function $F^{-1}$ to map the points $\{p_j \cup p\}$ into the surface $M$, getting the points $\{P_j \cup P\}$. Find the least-squares best-fitting plane $Q$ for the points $\{P_j \cup P\}$. Define the error function $E$ as the minimum euclidean distance between the plane $Q$ and the point $P$. The normalized error function is given by:

$$E_N = \frac{\text{dist}(Q, P)}{\|\text{max} - \text{min}\|}$$

where $\|\text{max} - \text{min}\|$ is the diagonal length of the bounding box of the mesh $M$.

**Refinement** We create a set of red-green meshes $\langle S_0, S_1, \ldots, S_{n-1}, S_n \rangle$. Each red-green mesh $S_i$ is initialized with the corresponding base mesh $B_i$, calculated in the previous stage, and with an empty set of modifications $\Sigma_i$.

We define the curvature-guided refinement as follows: for each point $p$ in $S$ compute the error $E_N$. If $E_N(p) > \epsilon$ refine all the triangles incident to $p$. The refinement of those triangles should reduce the error. So by repeating the described process a number of times, it eventually converges. The algorithm of Table 2 details the process.

```
Algorithm RefineMesh( S )
    repeat
        for each vertex $p_k \in S$
            compute $E_N(p_k)$
            if $E_N(p_k) > \epsilon$
                for each triangle $t_w$ incident to vertex $p_k$
                    MarkTriangle( $t_w$ )
                end
            end
        end
    end
    until $\forall E_N(p_k) < \epsilon$

Table 2. Minimizing the error $E_N$ of $S$
```

We apply the RefineMesh algorithm on each red-green mesh $S_i$ until curvature error function $E_N$ is minimized for all $S_i$. See the algorithm of Table 3.

As you can see in the proposed algorithm, we have in fact refined the same instance of the mesh every time. We just warp the mesh $S$ before each refinement step. The warping procedure assures that we minimize the curvature error metric $E_N$ for each red-green mesh $S_i$. In this way, every red-green mesh $S_i$ approximates $M_i$ independently according to the $E_N$ metric, but maintaining the consistency with all the other meshes.
Algorithm ConsistentRefinement( $S_i$ )

RefineMesh( $S_i$ )

if $i < n$

$S_{i+1} = \text{WarpMesh}( S_i, B_{i+1} )$

ConsistentRefinement( $S_{i+1}$ )

end

end

Table 3. Consistent refinement of $S_i$, keeping $E_N < \epsilon$

**Resampling and Smoothing** After applying the *ConsistentRefinement* algorithm, we have several consistent-refined red-green meshes $\langle S_0, ..., S_n \rangle$ condensed in the mesh $S_n$. In this step we will extract the meshes $\langle M'_0, ..., M'_n \rangle$ from the meshes in $S_n$.

To produce the final meshes, we iteratively apply the warp operator $W_{i,j}$ to $S_n$ in order to produce the red-green meshes $\langle S_n, S_{n-1}, ..., S_1, S_0 \rangle$. For each point $p \in S_i$ we use the corresponding inverse map $F_i^{-1}$ to map each point $p$ back to $P' \in M'_i$. Then we apply a constrained relaxing operator to each interior vertex $p \in S_i$ in order to smooth the final mesh $M'_i$. Our goal is to minimize the triangle distortion in $M'_i$, i.e. get approximately the same size of each triangle $T \in M'_i$ incident to $P'$.

We apply one or two iterations of the umbrella operator $U$ of Kobbelt et al. [14]. In practice this relaxation step works well with the curvature error metric $E_N$. The umbrella operator minimizes the membrane energy of the mesh, i.e. smooth the triangulation improving triangle quality. The update rule for a vertex $p_i \in S_k$ is:

$$U : p_i \mapsto \bar{p}_i = (1 - \alpha)p_i + \frac{\alpha}{\sum_{j=0}^{N_{p_j}} \lambda_{i,j}} \sum_{j=0}^{N_{p_j}} \lambda_{i,j} p_j$$

where $p_j$ are the 1-ring neighbors of $p_i$, $N_{p_i}$ is the degree of $p_i$ and $\lambda_{i,j}$ is the weight of the edge $\{i, j\}$. To obtain triangles near the same size in $R^3$ space, we choose to use the density weights $\lambda_{i,j}$:

$$\lambda_{i,j} = \sum_{T \in T_{p_j}} \text{Area}(T)$$

where $T_{p_j}$ is the set of triangles in $M' \subset R^3$ incident to $F_i^{-1}(p_j)$. So in other words, density weight $\lambda_{i,j}$ for the edge $\{i, j\}$ is the sum of areas of triangles in $M' \subset R^3$ incident to vertex $p_j$ mapped back to $R^3$ using the inverse mapping $F^{-1}$.

We apply the umbrella operator $U$ to each vertex $p_i \in S_k$ one by one, constraining the feature points in place. We repeat the relaxation procedure a few times. In each step, before moving a vertex to its optimized position, we check if any triangle folds over. If so, we do not move the vertex.
As we optimize the points in $R^2$, the size of triangles in $R^3$ change as well. So we need to recompute the areas of triangles, and therefore, the weights $\lambda_{i,j}$, after each application of $U$. The result is showed in Fig. 1 middle.

After the relaxation is finished, for each $S_i$ we apply the inverse map $F_i^{-1}$ to it again, and obtain the optimized mesh $M'_i$. The result is showed in Fig. 1 right.

### 3.5 Example

Figure 3 shows the steps of our consistent remeshing. The feature correspondences between two models, the common base mesh $B_c$ as well as two warps $B_0$ and $B_1$, the final red-green mesh $S_1$ refined and optimized and the resampled mesh $M'_1$.

### 4 Results

We have implemented the consistent remeshing of multiple disk-like surfaces as described above. The application was written in C++ using standard computational geometry data structures. We use the Parametrization module of the SINTEF GoTools library to compute the initial embedding and the CGAL library [4] for Constrained Delaunay Triangulation.

The remeshing results has been evaluated by Metro tool V.4.07 [5]. The Metro tool compares two triangular meshes which describe the same surface. Error can be measured in terms of the symmetric mean distance, the symmetric RMS distance and the Hausdorff distance. After getting the symmetric RMS distance, we followed the evaluate function in [10]. Accuracy was measured as 

$$\text{PSNR} = 20 \log_{10} \left( \frac{\text{peak}}{d} \right)$$

where $\text{peak}$ is the diagonal length of the bounding box and $d$ is the symmetric RMS distance between the original mesh and the remesh. Roughly speaking, the PSNR about 70dB is considered to be a nice approximation. Table 4 shows statistics about input datasets used in our experiments. Table 5 shows statistics of resulting consistent meshes with $\varepsilon_N < 0.00225$, plus total calculation time, which was performed on an AMD Phenom II X2 555 CPU 3.2GHz and 2.0GB of RAM. Table 6 shows the result of Metro tool comparisons between input and output meshes, the PSNR, the symmetric Hausdorff and Mean distances normalized w.r.t bounding box diagonal.

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Faces</th>
<th>Feature Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armadillo</td>
<td>16245</td>
<td>32287</td>
</tr>
<tr>
<td>Bunny</td>
<td>8941</td>
<td>17756</td>
</tr>
<tr>
<td>Horse</td>
<td>4900</td>
<td>9721</td>
</tr>
<tr>
<td>Noh-Mask</td>
<td>1945</td>
<td>3088</td>
</tr>
</tbody>
</table>

Table 4. Input datasets statistics
Table 5. Output datasets statistics and time

<table>
<thead>
<tr>
<th></th>
<th>Vertices</th>
<th>Faces</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armadillo/Bunny</td>
<td>11381</td>
<td>22651</td>
<td>2.73</td>
</tr>
<tr>
<td>Horse/Armadillo</td>
<td>10384</td>
<td>20662</td>
<td>2.60</td>
</tr>
<tr>
<td>Bunny/Noh-Mask</td>
<td>6373</td>
<td>12655</td>
<td>2.24</td>
</tr>
</tbody>
</table>

Table 6. Result of Metro tool comparisons between input and output meshes

<table>
<thead>
<tr>
<th></th>
<th>Hausdorff %</th>
<th>PSNR (dB)</th>
<th>Mean %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armadillo</td>
<td>0.48</td>
<td>68.98</td>
<td>0.03</td>
</tr>
<tr>
<td>Bunny</td>
<td>0.37</td>
<td>70.86</td>
<td>0.02</td>
</tr>
<tr>
<td>Horse</td>
<td>0.78</td>
<td>70.57</td>
<td>0.02</td>
</tr>
<tr>
<td>Armadillo</td>
<td>0.41</td>
<td>68.74</td>
<td>0.03</td>
</tr>
<tr>
<td>Bunny</td>
<td>0.38</td>
<td>69.35</td>
<td>0.02</td>
</tr>
<tr>
<td>Noh-Mask</td>
<td>0.38</td>
<td>68.26</td>
<td>0.03</td>
</tr>
</tbody>
</table>

4.1 Morphing

As the produced meshes \( M'_i \) shares their connectivity, we can easily morph them by simply linearly interpolate their vertices. In our examples we morph two meshes \( M'_i \) and \( M'_j \) by computing \( p_{\text{morphed}}^k = (1 - \lambda)p_i^k + \lambda p_j^k \). Feature point correspondences are crucial to the quality of the morphing since they allow plausible transitions. Figure 2 shows the spatial morphing sequences for various models.

5 Conclusions And Future Work

We have proposed a novel method to compute the consistent adaptive remeshing for multiple disk-like manifold meshes. Our technique minimizes the curvature error metric of all surfaces, produces meshes with low triangle count and are easy to implement. Moreover, our algorithm enforces the alignment of feature-point correspondences in the output meshes using a straightforward warping algorithm. We have demonstrated the mesh morphing application between disk-like meshes with feature-point correspondences specified by the user.

One field of future work is to handle closed meshes. A common practice is to partition the closed meshes into topological disks first and later map the patches individually. One option is to segment each mesh in two topological disks. Then we can apply our method to each patch. Patch boundaries can be matched later. To assure perfect matching of patch boundaries we can add steiner vertices at the boundary where necessary and re-triangulate it using our red-green subdivision meshes. Other field of future work is to avoid fold-overs completely. Since the feature points don’t constrain the initial parameterization, they are mapped to different \((s, t)\) coordinates for each input mesh. As we have explained, this may
Mauricio Cele López Belón

lead to folds under certain configurations. To avoid this pathological behavior, constrained parameterization techniques may be used. However a known drawback of constrained parameterization is the high distortion that arises even with few constraints.

References


Fig. 2. Morphing sequences generated through linear interpolation of vertices.

Fig. 3. Left: initial $B_c$ with boundary Steiner vertices and two (refined) warps $B_0$ (armadillo) and $B_1$ (bunny). Right: final red-green mesh $S_1$ and the generated mesh $M'_1$ (bunny)