surface remeshing in arbitrary codimensions

Surface Remeshing in Arbitrary Codimensions

Guillermo D. Cañas
Harvard University

Steven J. Gortler
Harvard University

Abstract

We present a method for remeshing surfaces that is both general and efficient. Existing efficient methods are restrictive in the type of remeshings they produce, while methods that are able to produce general types of remeshings are generally based on iteration, which prevents them from producing remeshes at interactive rates. In our method, the input surface is directly mapped to an arbitrary (possibly high-dimensional) range space, and uniformly remeshed in this space. Because the mesh is uniform in the range space, all the quantities encoded in the mapping are bounded, resulting in a mesh that is simultaneously adapted to all criteria encoded in the map, and thus we can obtain remeshings of arbitrary characteristics. Because the core operation is a uniform remeshing of a surface embedded in range space, and this operation is direct and local, this remeshing is efficient and can run at interactive rates.

1 Introduction

In this paper we describe a novel remeshing algorithm that produces flexible remeshes of reasonable approximation quality at interactive rates. Due to its speed, it may can be applied to dynamically changing meshes during simulation and rendering.

The algorithm uses the following simple two-stage structure. In the first stage, the mesh M is mapped using *some chosen function* $\mathbf{f}$ to some range space $\mathbb{R}^n$. In the second stage, a uniform grid is placed in the range space, and $\mathbf{f}(M)$ is uniformly *contoured* by intersecting it against this grid. In the resulting uniform remesh, each face will lie in a single grid-cell in $\mathbb{R}^n$ and so the variation of its range-space coordinates will be bounded by the grid size. Finally, this remesh is then mapped back to $\mathbb{R}^3$ and output by the algorithm.

Clearly the choice of $\mathbf{f}$ greatly affects the properties of the output remesh. If $\mathbf{f}$ is simply the identity mapping to $\mathbb{R}^3$, then each face of the remesh will be guaranteed to have bounded area. If, in contrast, we map each surface point p to $\mathbb{R}^3$ by using its normal $p \rightarrow (n_x, n_y, n_z)$, then each face of the remesh will be bounded in its normal variation. As shown later, when the Gauss map is used, then in the limit each face will have aspect ratio roughly proportional to the ratio of the surface's principle curvatures.

Mappings to $\mathbb{R}^6$ of the form $p \rightarrow (p_x, p_y, p_z, n_x, n_y, n_z)$ will guarantee bounds on both spatial and normal variation. Other mappings $\mathbf{f}$ can also be chosen to include data fields such as color to bound that type of variation in each output face. The mapping can be also chosen in a view-dependent fashion allowing for spatial variation to be measured in pixel coordinates.

Our algorithm has the following desired properties.

- It is fast, running in real time for moderate-sized meshes. The mapping and contouring stages each only require a single pass over the mesh. No iterative processing is needed. The mapping can be evaluated lazily as needed, resulting in sub-linear time behavior.
- It is quite flexible, one simply chooses the mapping $\mathbf{f}$ to encapsulate the kinds of remesh properties they wish to bound.
- By construction the algorithm outputs faces with L^∞ bounds on the variation of the mesh under $\mathbf{f}$.
- If $\mathbf{f}$ is chosen appropriately, the output remesh faces will be anisotropically elongated as needed. For example if $\mathbf{f}$ maps each mesh point to $\mathbb{R}^3$ using its normal (n_x, n_y, n_z) then faces will have aspect ratio proportional to the ratio of the surface's principle curvatures.
- Because of the regular structure of the intersection grid in $\mathbb{R}^n$, the output mesh is guaranteed to have all vertices with valence 4. If this mesh is dualized, the resulting remesh has all quad faces.

Our algorithm does have the following limitations

- Our remeshes are not optimal, and better results can be achieved using off-line methods. The algorithm can *over-tessellate* in some local regions. This also may result in some over long and skinny faces.
- When we dualize the remesh to obtain a mesh with all quad faces, we have no guarantees to the regularity

of the valences of the vertices. Both of these problems can be ameliorated to some degree using some iterative post-process as we describe.

- Similarly to many implicit surface contouring methods, we can not guarantee that the output mesh will have topology that matches the input surface. If the surface has a small feature that misses all grid-edges, this feature may not be sampled. (For example, an input barbell shape may get meshed as two disconnected spherical shapes.) Moreover, since our algorithm is a "seeded traversal", it will not directly find all of the disconnected components.

Overall our paper has the following contributions

- We introduce a general remeshing strategy based on mappings to $\mathbb{R}^n$ followed by uniform contouring in $\mathbb{R}^n$.
- We describe a novel but simple algorithm to trace out the intersection of a surface and uniform grid in $\mathbb{R}^n$ to produce the uniform tessellation. Traditional methods used to contour implicit surfaces in $\mathbb{R}^3$ are not appropriate for our remeshing since when $n > 3$, the surface is not codimension 1.
- We explore a number of interesting mappings $\mathbf{f}$ and show their resulting remeshes.

2 Previous Work

Approaches for surface remeshing that are based on iteration perform a series of local operations in order to obtain a mesh that is more optimal with respect to some metric [15, 13, 25]. These algorithms do not require any special pre-processing of the input, and allow great flexibility in the way they operate. Because they generally can accept any metric to evaluate the quality of a mesh, the output they produce can be adapted to any number of criteria. They are, however, inherently iterative, preventing them from producing complete remeshes at interactive rates.

Other approaches reduce the dimensionality of the problem by posing it in a parametrized domain of the surface. The remesh can be computed by uniformly sampling the parametrization in a direct way [11, 14, 21]. These techniques produce highly regular remeshes, but often at the expense of the resulting approximation quality. The domain can be non-uniformly point-sampled and a mesh built that connects these samples [3, 2], producing locally isotropic remeshes at interactive rates, but only *after* an expensive parametrization step. [1] uses curvature lines traced on the parametrized domain to induce a mesh. This method is specifically designed to produce one type of remesh.

The method of [8] poses the remeshing problem as a variational optimization problem, by decomposing the surface into regions that are balanced with respect to a predefined criteria using a k-means clustering approach. The regions are then approximated by polygons of an arbitrary number of sides, which can be adapted to the curvature of the surface. Dong et al. compute quadrilateral meshes with semi-regular connectivity, but their method is not interactive and produces nearly uniform elemnts [10].

The type of contouring that our approach uses is similar to methods for reconstructing implicit surfaces that are based on spatial decompositions [19, 18, 17, 23]. These methods are efficient when reconstructing implicit volumes, but unfortunately cannot be applied to input meshes without first converting them to implicit form (i.e. by computing their distance field). Extensions of implicit methods exist for n-dimensions [6, 24].

Other methods exist that reconstruct or repair a surface by finding its intersection with a grid or an octree [16, 4, 22, 20]. They are typically separated in two stages: a *scan-conversion* stage, and a contouring stage that may be based on a primal [4, 22, 20] or a dual grid [16]. In these methods, the scan conversion uses no coherence between neighboring triangles and can typically take seconds of processing time. The contouring process is limited to the codimension-1 case of recovering surfaces in $\mathbb{R}^3$.

Our approach is perhaps closest in spirit to that of [5], where an implicit surface is adaptively triangulated in a three-stage process that first warps $\mathbb{R}^3$, then applies a uniform contouring algorithm, and finally transforms the resulting mesh through the inverse of the warping function. Since this is a method for contouring implicit surfaces, it does not extend to arbitrary dimensions, and the warping is restricted to be invertible. By allowing both non-invertible mappings and mappings into arbitrary-dimensional spaces, we can create a variety of useful adaptive remeshes in a direct way and at fast rates.

3 Uniform Contouring in $\mathbb{R}^3$

We begin our exposition by describing a simple an efficient algorithm that is capable of uniformly contouring surfaces that are embedded in $\mathbb{R}^3$, and later show how this can be extended to surfaces (triangulated 2-manifold) in $\mathbb{R}^n$. This is a different problem than iso-surface extraction, which finds an $n-1$ dimensional manifold in $\mathbb{R}^n$. Although this arbitrary co-dimension method is quite simple and natural, we are not aware of its previous use in the literature. In section 8 we will show how various mappings $\mathbf{f} : M \to \mathbb{R}^n$, can be applied to a mesh to obtain useful remeshes.

We uniformly approximate a surface M sitting in $\mathbb{R}^3$ by considering its intersection with a uniform grid. This is shown in figure 2.a-c. The intersection between the surface and the uniform grid (figure 2.a) is a decomposition of the surface, with vertices of valence four everywhere. (Therefore by taking its dual (figure 2.b), we will later be able to obtain a uniform remesh with quads only). What follows assumes that the input surface is a triangle mesh representing a closed 2-manifold, and we describe latter in section 7 how this can be extended to manifolds with boundary. We will use the term *surface-triangle* to refer to a triangle in this input surface.

A uniform grid in $\mathbb{R}^3$ is a cellular decomposition of space, comprising what we term *grid-cells*, *grid-faces*, and *grid-edges*. Cells are unit cubes that tile space, while faces are the two-dimensional sets shared by adjacent cells. Edges are one-dimensional sets shared by adjacent faces. By intersecting the surface against this grid, we create a decomposition of M. This decomposition is comprised of *intersection-vertices*, *intersection-arcs*, and *intersection-regions*, which are the result of intersecting the surface M with the grid-edges, grid-faces, and grid-cells, respectively.

More specifically, the intersection between a surface and a grid has the following structure:

- A grid-edge is a one-dimensional segment, and will transversely intersect the surface at one (or more) discrete point(s) on the surface.

- A two-dimensional grid-face will intersect the surface forming one (or more) curve(s) on M. There are two possible cases for each such curve. In the *good case*, the curve is an *intersection-arc* connecting two intersection-vertices. In the *bad case*, the curve is a closed loop.

- The intersection of a grid-cell with M results in one (or more) disjoint intersection-regions of finite area on M. There are two possible cases for these regions. In the *good case*, regions have the topology of a disk. In the *bad case*, a region can have more complicated topology.

Associated with these intersections we define the following remeshed surface:

- For each surface with grid-edge intersection, we will output an intersection-vertex.

- For each intersection-arc of the good kind we will output an intersection-edge. Intersections of the bad kind, however, are ignored.

- For each simple cycle of intersection-arcs that is associated with one grid-cell, we will output one intersection-face.

First we note that if all of the intersections between the surface and the grid-cells are of the good kind, then this remesh will be homeomorphic to the input surface. Currently we do not have any conditions that guarantee all intersections will be of the "good kind", nor have we yet explored methods to test for this and adapt appropriately.

We also note that the remesh described above retains the same incidence relations of the grid, because it is derived from it. For this reason, the graph formed by intersection-vertices and intersection-arcs can be properly embedded (as a polyline) in the surface without any crossings.

Additionally because every grid-edge is incident to four grid-faces, every intersection-vertex will be incident to four intersection-edges. That is, all intersection-vertices have valence four. However, intersection-faces may be bounded by an arbitrary number of edges.

The above decomposition can be computed efficiently using a simple tracing algorithm, which we discuss next. Unlike implicit surface methods, this algorithm will generalize to arbitrary codimensions in a straight-forward way.

3.1 Algorithm

We compute the above decomposition of the surface, obtained by intersecting it with a uniform grid, by tracing the graph formed by the decomposition.

The tracing algorithm uses the following key data types:
point*:* $\mathbb{R}^3$ *coordinates*
intersectionVertex*:*
  {*point, surfaceTriangle, gridEdge, 4 gridFaceProcessedBits*}
intersectionEdge*:*{*2 intersectionVertices*}

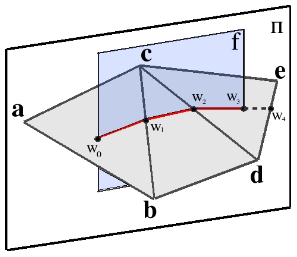
The *intersection-vertex* data type represents the point of intersection of a grid-edge with the surface. Each grid-edge is incident to four grid-faces. Each of these faces will need to be processed by tracing out the intersection arc between that face and the surface. As such we need to store 4 processed-bits at an intersection-vertex.

Starting from an intersection-vertex, we progress on the surface along all its incident arcs, which arrive at other intersection-vertices and in turn spawn more arcs. Eventually, arcs reach intersection-vertices that were already computed, closing the graph. The following pseudo-code summarizes this algorithm, which uses a seed intersection v_0 that is a chosen surface point that is simply translated to make it pass through a grid edge.

```
 $Q \leftarrow$ empty queue of intersectionVertices.
Enqueue $v_0$ in $Q$.
While $Q$ is not empty do
    Dequeue $v$ from $Q$.
    While $v$ has an unprocessed gridFace $f$ do
    Mark $f$ as processed in $v$.
    (point $\bar{p}$, surfaceTriangle t, gridEdge e) $\leftarrow$
        ArcTraverse(v.point, v.surfaceTriangle, $f$)
    if ($\bar{p}$, t, e) has yet to be encountered
        $\bar{v} \leftarrow$ new intersectionVertex($\bar{p}$, t, e)
        enqueue $\bar{v}$
    create intersectionEdge connecting $(v, \bar{v})$
    mark $f$ as processed in $\bar{v}$
```

The intersection-vertices and intersection-edges can be used to form the connectivity of the output remesh. The faces of the remesh can be easily computed by finding the simple cycles in the graph of intersection-vertices and intersection-edges. The various data fields in the remesh are obtained by sampling the surface point at the intersection-vertex.

3.1.1 Computing The Intersection-Arcs



The key operation, and the one that consumes the majority of the time in the entire algorithm, is the arc traversal along the surface. Because the surface is piecewise linear, its intersection with a grid-face f is a polyline, with one linear segment for every surface-triangle that the grid-face intersects (in the side figure, the intersection between the surface and a face f is the polyline $w_0w_1w_2w_3$). To trace an intersection-arc, represented as a polyline, we find the intersection between the supporting plane of the grid-face and every input surface-triangle along the polyline. We then test whether each of these intersections are completely inside the grid-face or whether it we have reached a grid-edge indicating, whether we should continue tracing the polyline or an intersection-vertex has been found.

Consider the example depicted in the side figure. A simple polyline from w_0 to w_3 represents the intersection between a grid-face f and the surface. The grid-face is a squared portion of its supporting plane Π. Imagine that we start tracing the polyline from w_0, knowing that we must progress in the direction towards w_1, which lies in Π. We first test whether w_1 is inside f. Because $w_1 \in f$, we can continue tracing to the next triangle $\triangle bcd$. In order to process triangle $\triangle bcd$, starting from w_1, we compute the intersection between $\triangle bcd$ and Π, resulting in segment (w_1, w_2). Again, because $w_2 \in f$, we have not yet reached a grid-edge and so we proceed to triangle $\triangle cde$. We compute the intersection between this triangle and f, and find that this time the intersection (segment (w_2, w_4)) exits f at point w_3. In general, because the input surface may be very finely tessellated, we may be forced to traverse many triangles along the arc until we reach a grid-edge on the boundary of f. Once the grid-edge is reached, the arc is completely traced. We can then report the found intersection. The following pseudo-code summarizes this procedure for tracing an arc.

```
**ArcTraverse(point** p, **surfaceTriangle** t, **gridFace** f)
  lineSegment $(p, \bar{p}) \leftarrow$ intersection between $f$ and t.
  **While** $\bar{p}$ does not lie on a gridEdge **do**
      t $\leftarrow$ neighbor surfaceTriangle of t that contains $\bar{p}$.
      $p \leftarrow \bar{p}$.
      lineSegment $(p, \bar{p}) \leftarrow$ intersection between $f$ and t.
  e $\leftarrow$ gridEdge containing $\bar{p}$
  return($\bar{p}$, t, e)
```

In practice, we have found that the cost, in terms of floating point operations, of computing the arc segments is low enough to be small even when compared with the cost of simply reading the sequence of triangles from memory.

We note here that this algorithm does not require the mapping **f** of the surface to $\mathbb{R}^3$ to be an invertible embedding. The arcTraverse routine always keeps track of the current surface-triangle and so it can unambiguously trace the intersection of the mapped surface and a grid-face. The main algorithm uses the surface-triangle id and grid-edge id to uniquely identify an intersection-vertex, as well as to test if we have previously encountered it. (A surface-triangle and grid-edge can intersect at most once). As the arcs are traversed, and new intersection-vertices are encountered, we sample the appropriate data from the surface-point (unmapped geometric position, as well as normal, and color). This data is stored in the intersection-vertex data structure. Because we directly sample unmapped position data from the surface, we never need to explicitly apply $\mathbf{f^{-1}}$.

We note that in our algorithm, the representation of the grid is implicit in the logic of the algorithm, and requires no explicit storage. (From a grid-edge id, we can directly compute neighboring grid-face ids, and vice-versa. And from a grid-face id, we can directly compute the geometry of its supporting plane, π.) We also note that the algorithm never touches surface-triangles that lie completely in a grid-cell and that do not touch any grid-face. Thus, for highly tessellated input surface, the remeshing algorithm will run in *sublinear* time.

Degenerate Intersections Certain degenerate intersections between the surface and the grid may invalidate the structure described in section 3: surface vertices lying on grid-faces or grid-edges, surface triangles lying entirely on grid-faces, or grid-edges intersecting a surface-triangle at one of its edges. We detect these events by using exact large integer arithmetic and predicates that involve only multiplications and additions. Degenerate cases are symbolically perturbed to break ties in a consistent manner, similar to [12]. This frees us from considering degeneracies in the logic while ensuring consistency for all inputs.

4 Uniform Reconstruction in $\mathbb{R}^n$

We now generalize the algorithm of the previous section to higher-dimensions. Given a continuous, but not necessarily invertible map $\mathbf{f} : M \rightarrow \mathbb{R}^n$ that embeds a surface M in $\mathbb{R}^n$ (with $n \geq 3$), we uniformly triangulate $\mathbf{f}(M)$ directly in $\mathbb{R}^n$.

We first transform the input surface M, which is given as a triangle mesh, by applying the mapping $\mathbf{f}$ to all its vertices. Points in the interior of each surface-triangle map to $\mathbb{R}^n$ using their barycentric coordinates.

We must now extend all the operations of the algorithm described in section 3 to work with a grid in $\mathbb{R}^n$ and a surface $\mathbf{f}(M) \subset \mathbb{R}^n$. We use the same pseudo-code that was used before, but we must generalize the operations of finding a seed and computing arcs. In order to understand how these operations can be extended to $\mathbb{R}^n$, we first describe the structure of a uniform grid in $\mathbb{R}^n$, against which the surface will be intersected.

4.1 Uniform Grid in $\mathbb{R}^n$

A uniform grid in $\mathbb{R}^n$ is composed of elements of every dimensionality, from 0 to n. However, we are only concerned with those of dimension n, $n-1$, and $n-2$, that is, of codimension 0, 1, and 2. We again refer to these as grid-cells, grid-faces, and grid-edges, respectively. Note that for $n = 3$ this results in exactly the same notation as used above.

This generalization of a uniform grid in $\mathbb{R}^3$ to n-dimensions has the advantage that many of its properties are intact. For instance, the intersection of the surface $\mathbf{f}(M)$ with a grid-cell, grid-face, and a grid-edge is, respectively, an intersection-region, an intersection-arc, and an intersection-vertex on the surface. Intuitively, because a grid-cell has codimension 0, its is simply a portion of space with volume, and its intersection with the transformed surface is a two dimensional region of it. Analogously, a grid-face is the result of applying to a cell *one* linear constraint, it is therefore of codimension 1 and its (generic) intersection with the surface is a set of dimension one less than the dimensionality of the surface. As a result, the intersection of the surface with a grid-face is an intersection-arc of dimension 1. Predictably, intersecting a grid-edge (codimension 2) with the surface results in a single point: an intersection-vertex. Grid elements of higher codimension than 2 do not need to be considered since, for a surface in general position, its transversal intersection with such an element is null.

An important property that is carried over to n-dimensions is the valence of intersection-vertices, which once again is always four. Analogously as in $\mathbb{R}^3$, grid-faces are contained in a (codimension-1) axis-aligned hyperplane (one linear constraint). A grid-edge in turn, is contained in a (codimension-2) axis-aligned set obtained by intersecting two hyperplanes (two linear constraints). Similarly as in $\mathbb{R}^3$, in $\mathbb{R}^n$, grid-edges have two incident grid-faces for each of the two hyperplanes that they are contained in, therefore grid-edges have exactly four incident grid-faces. Consequently, every intersection-vertex has valence four.

We can now confront the task of extending all the steps in the reconstruction algorithm to n-dimensions. Finding a seed has a simple extension: we choose a point on the surface $\mathbf{f}(M)$ and translate the surface so that a grid-edge will pass through the chosen point. This point can now serve as a seed. Computing the intersection-arcs, however, requires further considerations.

4.2 Computing Arcs in $\mathbb{R}^n$

Tracing arcs in $\mathbb{R}^n$ follows exactly the same pseudo-code as in section 3.1.1. An intersection-arc is still the intersection between a grid-face and the surface. As before, we trace the arc from surface-triangle to surface-triangle until we reach a grid-edge, and we place an intersection-vertex at that point.

In order to find the intersection between a surface-triangle in $\mathbb{R}^n$ and a grid-face, we first compute the intersection between the (two-dimensional) supporting plane of the surface-triangle and the (codimension one) supporting hyperplane of the grid-face, resulting in a (one-dimensional) line. This is an algebraic operation (which is especially simple when our grid is axis aligned).

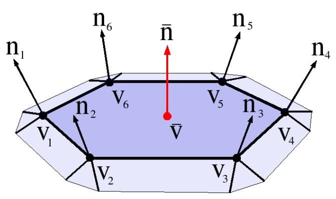
The actual intersection between the triangle and the face must be contained as a line segment contained in this computed line. This segment is computed by *clipping* the line against the surface-triangle and the grid-face. (Note, however, that the cost of clipping a line against a grid-face is linear in the dimension n).

The above generalization of the arc traversal is expressive enough that we, in fact, do not use separate implementations for $\mathbb{R}^3$ and $\mathbb{R}^n$, but instead a single one with the dimension n being a compile-time parameter. Additionally, the speed of the algorithm in practice does not vary with the dimension n, as the operations that are sensitive to the dimension have a very small constant, and, as explained above, the n dimensional grid is never explicitly constructed or stored.

5 Computing the Dual

If we want our algorithm to output a mesh with all quads, we can achieve this by creating a dualized remesh. In this remesh, the connectivity is the topological dual of the primal remesh. Because all vertices in the primal have valence four, all faces in the dual are quads.

The result is always a manifold This is in contrast with dual contouring methods for implicit surfaces where two or more sheets of the surface can touch at a vertex. This cannot happen in our method. Because the traversal is done *on the surface*, not by traversing space. If a surface passes by a grid-cell cube several times, it will generate a new set of vertices and quads for that cell each time it passes through the cell.



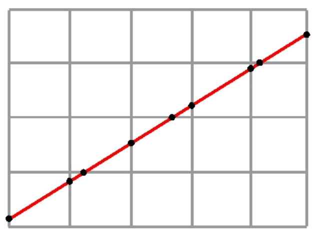
We can simply set the geometry and normal for a dual vertex $(\bar{\mathbf{v}}, \bar{\mathbf{n}})$, whose primal is a face f, by simple averaging as $(\mathbf{v} = \sum \mathbf{v}_i/m$, $\mathbf{n} = \sum \mathbf{n}_i/\|\sum \mathbf{n}_i\|)$, where $(\mathbf{v}_i, \mathbf{n}_i)$ are the primal vertices incident to f.

To obtain a higher quality dualization, we set the position $\bar{\mathbf{v}}$ to minimize the Quadric Error Metric based on the distance to the tangent planes defined by $(\mathbf{v}_i, \mathbf{n}_i)$.

6 Iterative Improvement

Although globally the density and sizes of elements of our algorithm adjust to all the quantities measured by the mapping, they may not be well distributed locally.

Consider as an example the situation depicted in the side figure, where a straight line is remeshed by intersecting it with a uniform grid on the plane. Although the average distribution of sampling points is proportional to the length of the curve, locally, their density is irregular: higher where



the line passes close to a grid-vertex, and lower away from them. This situation is aggravated as the dimensionality of the mapping increases. A (0-dimensional) vertex of a uniform grid in $\mathbb{R}^n$ will have $n \cdot (n-1)$ (codimension-2) grid-edges incident to it. As the number of incident grid-edges increases, the local non-uniformity of the sampling equally increases. For very-high-dimensional mappings, intersection-vertices can cluster together in groups. This can degrade the quality of the remesh.

Relaxation A natural solution to the above problem is to somehow relax the vertices' positions until they are locally uniformly distributed. The relaxation step could take the form of simple averaging of neighboring vertices, but this produces significant shrinking and smoothing. Other methods exist that do not shrink the surface, but produce smoothing [26, 27]. The type of relaxation that we desire locally slides the vertices *without* smoothing the surface. One possible solution would be the vertex repositioning method of [25]. Here we present a very simple alternative.

To produce better vertex distributions, we apply a series of relaxation iterations to each dual vertex $(\bar{\mathbf{v}}, \bar{\mathbf{n}})$ - each iteration consisting on two steps: a smoothing step, and a projection step. A smoothing step is an instance of simple Gaussian smoothing, where each dual vertex's position and normal are substituted by the average of those of its neighbors, obtaining a new *smoothed vertex* $(\bar{\mathbf{v}}', \bar{\mathbf{n}}')$. This first step locally relaxes vertices over the surface, but also introduces smoothing and shrinking.

We follow every smoothing step by a projection step, intended to eliminate the shrinking and smoothing, while retaining the local relaxation. To this end, we consider a modification of the Quadric Error Metric method. Because each dual vertex has associated with it a primal face with incident vertices $(\mathbf{v}_i, \mathbf{n}_i)$, we can consider the Quadric Error Metric defined by the distances to the tangent planes defined at those points. However, if we were to apply the QEM method directly at this point, the resulting vertex would be always the same and independent of the position of the smoothed dual vertex $(\bar{\mathbf{v}}', \bar{\mathbf{n}}')$. Instead, we restrict the QEM solution to lie on the line $\bar{\mathbf{v}}' + \lambda\bar{\mathbf{n}}'$. The consequence of this decision is two-fold. On the one hand, because the solution lies in $\bar{\mathbf{v}}' + \lambda\bar{\mathbf{n}}'$, the effect of the combined smoothing/projection iteration is that vertices are effectively locally *slid* over the surface without smoothing or shrinking. On the other hand, while the solution to the original QEM problem reduces to solving a linear system, or more generally an SVD decomposition [18, 17] to avoid stability problems, the solution of a QEM problem restricted to a line reduces to simply solving a linear equation (a considerably faster operation).

Furthermore, the projection step can be made unconditionally non-singular by introducing an extra *QEM constraint* plane at $(\bar{\mathbf{v}}', \bar{\mathbf{n}}')$ weighted by a very small ϵ, which, if $\bar{\mathbf{n}}'$ is unitary, leads to the following expression:

$$\lambda = \frac{\sum_{i=1}^{m} < \mathbf{v}_i - \bar{\mathbf{v}}', \mathbf{n}_i > \cdot < \bar{\mathbf{n}}', \mathbf{n}_i >}{\sum_{i=1}^{m} < \bar{\mathbf{n}}', \mathbf{n}_i >^2 + \epsilon}$$

As shown in figure 1 (lower left and right), by applying the described relaxation step, vertices in the final remeshing are locally more evenly distributed while retaining their global density distribution over the surface, and no shrinking or smoothing is introduced. The speed impact of this step is quite small, generally around 5 to 8

Quad Collapses The local non-uniformity of the primal remesh described above affects both the distribution of dual

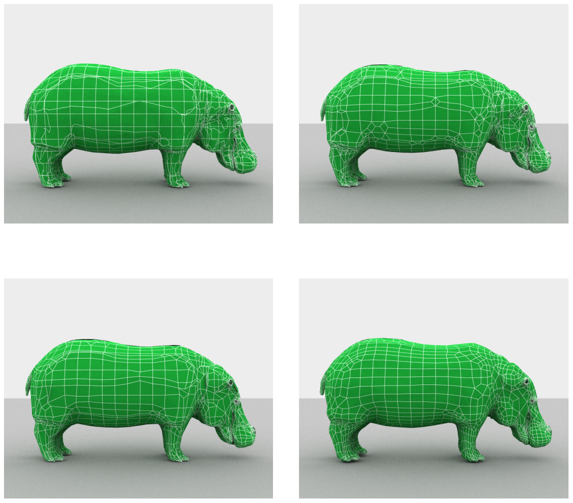


Figure 1. The hippo remeshed in $\mathbb{R}^6$. The primal intersection (upper left) is dualized into a quad-remesh (upper right). A quad-collapsing pass improves the vertex valence distribution (lower left). Vertex relaxation locally redistribute the vertices' positions (lower right).

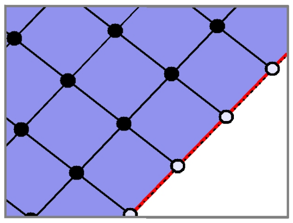
vertices on the surface as well as their valence. We use vertex relaxation to improve the first, while the second can greatly benefit from local optimization operations similar in spirit to those of [3, 25]. In their work, a series of edges are considered and they are flipped only when this improves the valence regularity of the mesh.

Because our dual remesh is composed of quads only, edge flipping cannot be applied in general without generating non-quads. However, we may pick any quad in the dual and collapse it if the valence variance of its incident vertices is lower after the collapse. To implement a quad collapse, consider a quad with ordered vertices (a, b, c, d). We collapse opposing vertices, say (a, c). It is easy to see that the resulting mesh is still all quads (assuming their respective neighborhoods are unrelated). This operation eliminates one quad and collapses two vertices into one.

is determined by simple averaging from those of the collapsed vertices, while its associated QEM is obtained by merging those of the collapsed vertices. The vertex is then projected back to the surface using the merged QEM information, by applying the procedure described above.

Note that this collapsing step can modify both the positions of certain vertices as well as their connectivity. For this reason, we apply it just after the dualization step, and before the local vertex relaxation. Figure 2 shows this complete process, where a primal remesh is first dualized, a quad-collapsing stage locally improves the valence regularity of the remesh, and finally vertex relaxation improves the local uniformity of the vertices.

7 Reconstructing Manifolds with Boundary



If we allow the input manifold to have boundaries, then as we trace intersection-arcs we may reach the boundary of the surface. Shown in the side figure is a diagram of an intersection graph near a boundary (solid circles are intersection-vertices and black segments connecting them are intersection arcs; the boundary is shown in red). At the point of intersection between the arc and the boundary we may not continue tracing. However, we use this intersection point in order to reconstruct the boundary itself by placing a new boundary-intersection-vertex there (hollow circle). Because the boundary intersection-vertices shown in the figure have valence one (they are only connected along the arc that generated them), they do not form regions. We must therefore connect them using *boundary edges* to the two boundary intersection-vertices that are contiguous along the boundary (dashed lines).

During the dualization process, we do not create dual edges for each primal boundary edge. We note that on boundaries, there will be vertices with only valence 2.

To prevent the new dual boundary edges from "shrinking" away from the primal boundary, we only use primal boundary vertices when computing the initial dual boundary vertex position. During the smoothing stage, we only average dual boundary vertices with other dual boundary vertices. During the QEM stage, we only use data from the primal boundary vertices.

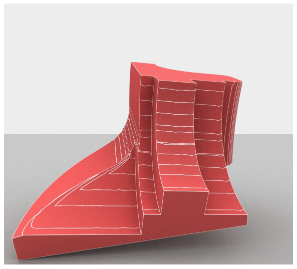
8 Mappings for Remeshing

The following applications demonstrate that our contouring algorithm is able to produce a wide variety of remeshes by simply varying the chosen map $\mathbf{f}$.

Choosing the identity map and reconstructing the surface in $\mathbb{R}^3$, we efficiently obtain a uniform remesh with elements of bounded size. This is shown in figure 2.a-c. The graph that we trace in section 3 is shown in figure 2.a, whose elements (arcs and vertices) are, respectively, the intersection of the surface with the edges and faces of the uniform grid. Its dual (figure 2.b) is an all-quads uniform remesh. By including a scaling factor in the mapping we can control the resolution of the uniform remesh (figure 2.c shows the effect of increasing this factor.)

Different types of anisotropic meshes can be obtained by incorporating normal variation in the mapping. We may wish to directly map the surface through its Gauss map - mapping every point to its normal in $\mathbb{S}^2$. We show in the appendix that, for surfaces, if a uniform tesselation of the Gauss map is used as range, then in the limit each intersection-region will have aspect ratio roughly proportional to the ratio of the surface's principle curvatures, which has been shown to be the optimal aspect ratio with respect to derivative error [9, 8, 7]. In order to avoid dealing with a codimension-0 mapping, and to ensure that the elements in the range space are approximately uniform (of approximate the same size and unit aspect ratio), we simply embed the Gauss map in $\mathbb{R}^3$ and reconstruct the transformed surface in $\mathbb{R}^3$ directly, rescaling the normals so they map to the unit cube and not the unit sphere. This introduces a small amount of bounded distortion in the mapping.

The results of using the Gauss map are shown in figures 2.d and 2.g, where 2.d is the intersection structure (visualized in the unmapped space) and 2.g is its all-quads dual. Although using the Gauss mapping will produce anisotropic remeshes at interactive rates, certain problems can occasionally arise. As seen in Figure 2.d, the image of the intersection-arcs on the mesh can be quite curved. If these arcs are to be represented as straight intersection-edges, significant approximation errors will occur.



If we are using a Gauss map, we are required to have a C^0-continuous normal-field over the input surface. Thus, in order to represent a sharp feature in the input surface, we must insert a degenerate triangle of null area between any two input triangles that straddle a feature edge (resp. an additional triangle with no area at a sharp vertex). This ensures, without changing the surface's geometry, that normals can have differing values at both sides of a feature. When the Gauss map is applied to such a model, the sharp features will exhibit significant normal variation. As such, the uniform tessellation in this space will automatically place intersection arcs along these sharp features. This behavior is shown in the side figure.

This mesh could be topologically dualized. In this case, the resulting quads may have more than one vertex/edge coincident on the original surface, and so may be degenerate geometrically. Because of the large bending of intersection-arcs, this remesh is not geometrically suitable to be directly dualized and, similarly to [8], it would instead need to be approximated by a polygonization of the primal regions.

To ensure stability when computing anisotropic remeshings we therefore combine *both* position and normal variation in the mapping. Position and normal components are each embedded in $\mathbb{R}^3$, and thus the mapping is six-dimensional. Both can have a different scale factor associated with them, providing control over the amount of anisotropy of the reconstruction. Because this is a high-dimensional mapping, the reconstruction tends to have a less regular valence distribution. We follow the reconstruction by one pass over the remesh intended to improve the vertex valence variance. Results of this algorithm are shown in figures 2.e, 2.f, 2.h, and 2.i. Figure 2.e shows the intersection between the six-dimensional grid and the transformed surface. The dual remesh is shown in figure 2.f. We show the effect of varying the spatial and normal scaling factors in figures 2.f, 2.h, and 2.i. By decreasing the scaling factor of the normal component of the mapping we obtain remeshes that range from highly anisotropic (2.f), to uniform (2.i).

Figures 2.j-l, show an example where the input surface has been mapped to a nine-dimensional space using position, normal, and color-gradient, with three components each: $p \rightarrow (p_x, p_y, p_z, n_x, n_y, n_z, \|\nabla r\|, \|\nabla g\|, \|\nabla b\|)$. The contribution of the color variation part of the mapping is best seen at the sharp color transitions between the white and red areas, in the hat and near the hands. The primal decomposition is shown in figure 2.k, representing the intersection between the surface embedded in $\mathbb{R}^9$ and a uniform grid in that space. The dual all-quads remesh is shown in figure 2.l. Figure 2.j shows the fraction of the model that is actually transformed through the mapping (the colored part). Because it is evaluated lazily as needed, the mapping only has to transform part of the surface, resulting in sub-linear remeshing cost when applied to high resolution input meshes.

Timings are presented in table 1. Speed was measured on a Pentium 4, 3Ghz. Error was measured using the METRO tool. For comparison, we have run QEM [13] on the hippo model, which was approximated with an equal number of edges, and an RMS error of $7.04e{-}5$ (runtime 850 ms.), as opposed to $7.82e{-}4$ for our remeshing of figure 1 (runtime 51 ms.) The speed depends on the size of the input and output. Most models can run at interactive speeds, with perhaps the exception of the adaptive David model, which produces as output a quad-mesh of size comparable to the input.

9 Conclusion and Future Work

In this paper we have described a simple and effective remeshing algorithm based on mapping a surface to a range space, and uniformly tessellating the surface in that space. The result is a fast algorithm that is flexible and provides absolute bounds on data variation over the faces in the remesh and is guaranteed to have all quad faces.

Model	n	input size	output	time (ms)	RMS
David	3	100,454	23,600	53	5.62e-4
David	6	100,454	69,174	223	4.08e-4
Elephant	3	311,376	14,208	52	3.40e-4
Hippo	6	46,202	8,375	51	2.07e-3
Fish	3	129,664	17,287	61	8.42e-4
Santa	9	151,558	18,578	83	5.55e-4

Table 1. Timings for remeshings. *n* is the dimensionality of the mapping. *Input* and *output* sizes are, respectively, in number of triangles and quads. RMS error is measured after scaling models to a unit bounding box diagonal.

In future work, we plan to more carefully study various mappings and their approximation theoretic properties. It would be beneficial to find mappings that can *guarantee* the topological correctness of the tessellation.

References

[1] P. Alliez, D. Cohen-Steiner, D. Devillers, B. Lévy, and M. Desbrun. Anisotropic polygonal remeshing. In *SIGGRAPH 03*, pages 485–493, 2003.

[2] P. Alliez, E. C. de Verdière, O. Devillers, and M. Isenburg. Isotropic surface remeshing. In *Proceedings SMI '03*, pages 49–58, 2003.

[3] P. Alliez, M. Meyer, and M. Desbrun. Interactive geometry remeshing. In *SIGGRAPH 02*, pages 347–354, 2002.

[4] C. Andujar, P. Brunet, and D. Ayala. Topology-reducing surface simplification using a discrete solid representation. *ACM Trans. Graph.*, 21(2):88–105, 2002.

[5] L. Balmelli, C. J. Morris, G. Taubin, and F. Bernardini. Volume warping for adaptive isosurface extraction. In *IEEE VIS '02*, pages 467–474, 2002.

[6] P. Bhaniramka, R. Wenger, and R. Crawfis. Isosurfacing in higher dimensions. In *IEEE Visualization*, Washington, DC, USA, 2000.

[7] G. D. Canas and S. J. Gortler. On asymptotically optimal meshes by coordinate transformation. In *Proceedings of 15th International Meshing Roundtable*, 2006.

[8] D. Cohen-Steiner, P. Alliez, and D. Desbrun. Variational shape approximation. *SIGGRAPH '04*, pages 905–914, 2004.

[9] E. D'Azevedo and B. Simpson. On optimal triangular meshes for minimizing the gradient error. In *Numer. Math. 59, 321–348*, 1991.

[10] S. Dong, P.-T. Bremer, M. Garland, V. Pascucci, and J. C. Hart. Spectral surface quadrangulation. In *ACM Transactions on Graphics, Proceedings of SIGGRAPH*, 2006.

[11] M. Eck, T. D. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH 95*, pages 173–182, Aug. 1995.

[12] H. Edelsbrunner and E. P. Mucke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. In *SCG '88*, pages 118–133. ACM Press, 1988.

[13] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. *SIGGRAPH '97*, pages 209–216, 1997.

[14] X. Gu, S. J. Gortler, and H. Hoppe. Geometry images. In *SIGGRAPH '02*, pages 355–361, 2002.

[15] H. Hoppe. Progressive meshes. In *SIGGRAPH '96*, pages 99–108, 1996.

[16] T. Ju. Robust repair of polygonal models. *SIGGRAPH '04*, pages 888–895, 2004.

[17] T. Ju, F. Losasso, S. Schaefer, and W. Warren. Dual contouring of Hermite data. *SIGGRAPH '02*, pages 339–346, 2002.

[18] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature sensitive surface extraction from volume data. In *SIGGRAPH '01*, pages 57–66, 2001.

[19] W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. In *SIGGRAPH '87)*, pages 163–170, 1987.

[20] F. S. Nooruddin and G. Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE TVCG*, 9(2):191–205, 2003.

[21] E. Praun and H. Hoppe. Spherical parametrization and remeshing. *SIGGRAPH '03*, pages 340–349, 2003.

[22] C. Rocchini, P. Cignoni, F. Ganovelli, C. Montani, P. Pingi, and R. Scopigno. Marching intersections: An efficient resampling algorithm for surface management. In *SMI '01*, pages 296–305, 2001.

[23] S. Schaefer and J. Warren. Dual marching cubes: Primal contouring of dual grids. In *Pacific Conference on Computer Graphics and Applications*, pages 70–76, 2004.

[24] J.-K. Seong, G. Elber, and M.-S. Kim. Contouring 1- and 2-manifolds in arbitrary dimensions. In *Shape Modeling and Applications*, 2005.

[25] V. Surazhsky and C. Gotsman. Explicit surface remeshing. In *Symposium on Geometry Processing*, pages 17–28, 2003.

[26] G. Taubin. Curve and surface smoothing without shrinkage. In *International Conference on Computer Vision*, pages 852–857, 1995.

[27] G. Taubin. Dual mesh resampling. *Graphical Models*, 64(2):94–113, 2002.

Appendix

We briefly argue that, by mapping every point of a surface to its normal, and reconstructing the surface uniformly in normal space, the elements will, in the limit, have aspect ratio approaching the ratio of principal curvatures, and will be elongated along the direction of minimum curvature.

For a surface being approximated very finely we can approximate the surface near a face f by its second order Taylor expansion which, in a reference frame formed by the normal $(\hat{z})$ and the principal curvature directions $(\hat{x}, \hat{y})$, can be expressed as $z = -\frac{1}{2}(\kappa_1 x^2 + \kappa_2 y^2)$. The normal field in the vicinity of the center of the expansion is $\hat{n} = \frac{(\kappa_1 x, \kappa_2 y, 1)}{1+\kappa_1^2 x^2+\kappa_2^2 y^2}$.

Because the face f is uniform in normal space, we can consider it circumscribed by a very small circle around the normal $(0, 0, 1)$, and see how this circle transforms back on the surface to analyze the shape of the face. A small circle around normal $(0, 0, 1)$ is the set of normals such that $< \hat{n}, [0, 0, 1] >= c$, with $c < 1$ very close to 1. This circle can be transformed back to the surface as the set of points such that

$$< \frac{[\kappa_1 x, \kappa_2 y, 1]}{1 + \kappa_1^2 x^2 + \kappa_2^2 y^2}, [0, 0, 1] >= \frac{1}{1 + \kappa_1^2 x^2 + \kappa_2^2 y^2} = c \quad (1)$$

which can be written as

$$\kappa_1^2 x^2 + \kappa_2^2 y^2 = \epsilon \quad (2)$$

where $\epsilon = \frac{1}{c} - 1$, which, because c is close to 1, means that $\epsilon > 0$ is very small. Equation 2 represents an ellipse aligned with the principal curvature directions, of aspect ratio equal to the ratio of the principal curvatures, and elongated along the direction of minimum curvature.

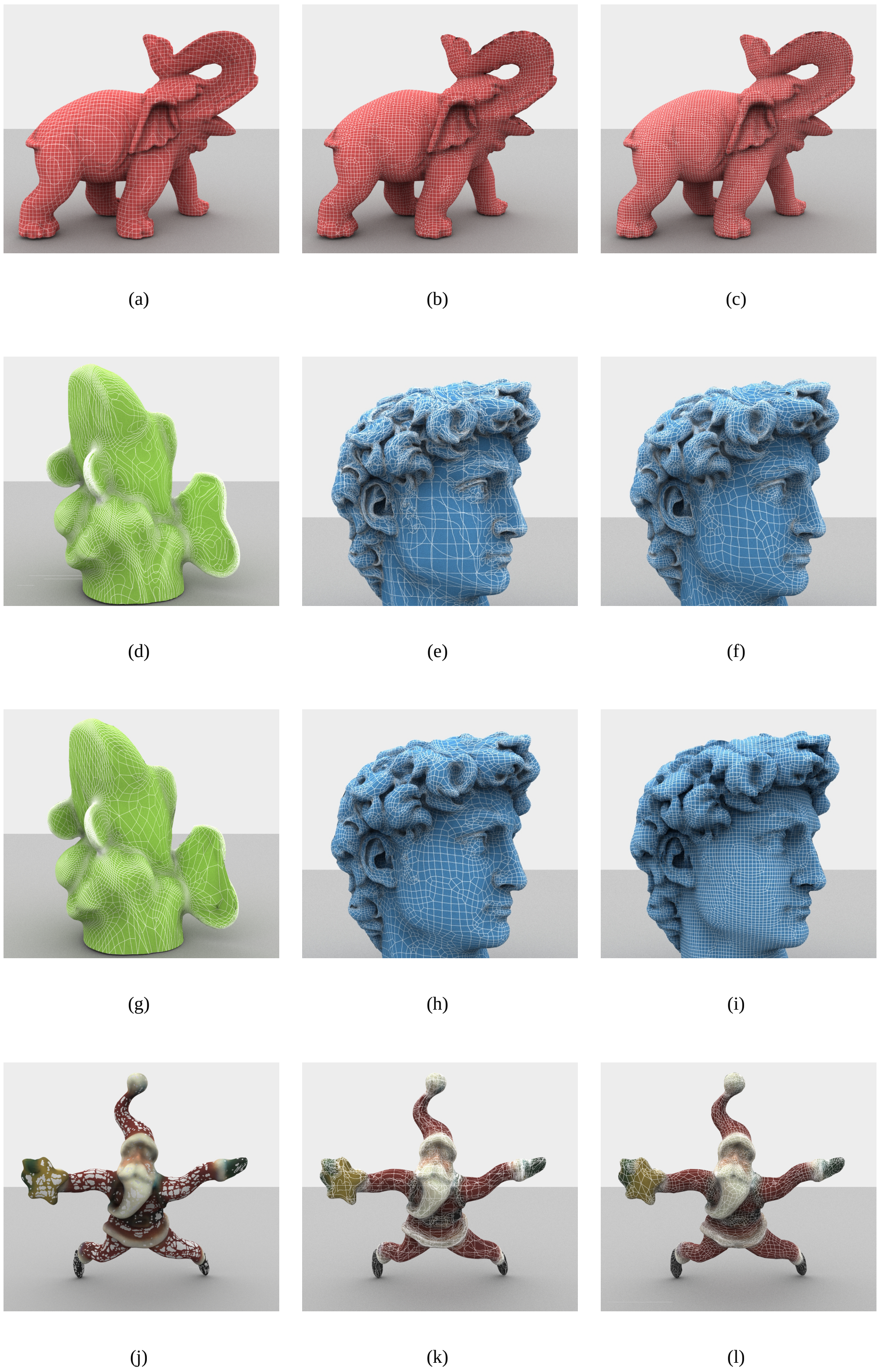


Figure 2. Remeshings obtained using our algorithm by applying different mappings to input surfaces. The first row shows the effect of remeshing under the identity map. Figures d and g show how surfaces are remeshed when mapped through the Gauss map. We combine position and normal in figures e, f, h, and i. Finally, figures j through l describe the process of remeshing a surface undergoing a nine-dimensional mapping.