Fast Hierarchical Discretization of Parametric Boundary Representations

Mikhail Frank and Horea Ilies

November 4, 2009

Abstract
Discretization of parametric surfaces not only expedites their rendering, but also facilitates other applications such as motion planning, collision detection, and engineering analysis. Currently, discretization methods such as Delaunay Triangulation and Advancing Front techniques, which sample and mesh simultaneously and can guarantee well shaped triangles, are popular despite their high computational cost.

In this paper we present a novel two-part sampling and meshing algorithm, which produces topologically correct meshes on arbitrary parametric boundary representations, without additional topological information such as an initial coarse mesh. The algorithm offers a performance increase of approximately two orders of magnitude over Delaunay based methods and at least one order of magnitude over advancing front methods. Based on spatial partitioning, our rule-based approach allows the user to employ arbitrary subdivision criteria to obtain meshes with differing geometric properties. In practice we afford very good control over the properties of the mesh, however the performance increase we offer comes at the expense of analytical guarantees concerning triangle shape. The hierarchical nature of our surface decomposition results in multi-resolution meshes well suited to rendering, collision detection, and path planning, where hierarchical pruning is heavily employed. Moreover, this hierarchy provides an intuitive control structure, which facilitates a future multithreaded implementation.

1 Introduction

The parametric surface has become the preferred representation in geometric modeling, and parametric modeling engines drive the major Computer Aided Design (CAD) packages such as Unigraphics NX and CATIA. Parametric representations are continuous, differentiable, and they offer the conveniences that surfaces can be defined easily and surface points can be enumerated very quickly. Other surface representations, such as implicit surfaces, are also continuous and differentiable, but it is the ease and flexibility with which parametric surfaces can be defined that makes them so popular in modeling.

1.1 Motivation

Despite their utility in the context of modeling, parametric surfaces can be quite cumbersome. Intersections, distance computations, and Point Membership Classification (PMC) testing are all expensive operations where parametric surfaces are concerned, and this makes rendering as well as motion planning and collision detection very time consuming if parametric representations are used directly. Consequently, parametric surfaces are typically discretized prior to use in such applications.

Unfortunately obtaining quality discrete representations of parametric surfaces also takes a relatively long time. It is not uncommon, even with state of the art hardware and software, to wait minutes to mesh a relatively simple model from CAD. This is an acceptable state of affairs if the intention is to export the geometry to a discrete format for later use and reuse. Such is the case for example in gaming and interactive simulation applications as well as in engineering analysis. However, if we were able to speed up this conversion process, such that we could afford to build a high-quality discrete representation of a CAD model on the fly, then we could drastically improve the CAD interface. Improving the state of the art in fast discretization of
parametric Boundary Representations (B-Reps) would not only improve rendering in the modeler interface, but could also facilitate tactile assembly of parts, immediate ability to move linkages without having to carefully specify constraints first, and faster more robust automated planning of tool paths. Such features would create a richer, more intuitive, and more powerful CAD interface.

1.2 Problem Statement

The problem of discretizing parametric surfaces can be thought of in terms of two major deliverables, a sample set, and the associated topology. For some applications, the samples alone are sufficient, however often it is necessary to establish connectivity between the points in order to specify their collective topology. An ideal general discretization algorithm would be able to sample quickly, without the additional overhead of establishing connectivity, but should also be capable of connecting the samples to tessellate the surface with well shaped triangles. We therefore elect to divide the task of discretizing surfaces into two steps, sampling and polygonization.

To sample parametric surfaces, it would clearly be advantageous to exploit the ease with which we can enumerate surface points by choosing their parameter space coordinates. The primary problem is that the geometry of the parameter space is distorted by the mapping to \( \mathbb{R}^3 \), and consequently we do not have direct control over the distribution of points on the surface. Therefore, in order to create ‘uniform’ sample distributions on arbitrary parametric surfaces, we must solve a 2D distribution problem under uncertainty, namely the mapping.

In the second step, polygonization, we must connect the points such that they form a triangular mesh. The sample distribution is already determined, so the problem becomes finding the connectivity, with the help of some quality metric(s) such as triangle shape and distance from the surface. A second challenge during the meshing phase is coping with trimmed edges of the surfaces in a complex, such that we can join adjacent surface meshes according to valid mesh topologies.

1.3 Previous Work

At the time of their inception, the relative expense of intersections and distance computations involving parametric surfaces made their visualization a challenging proposition. Fast discretization methods were absolutely necessary to visualize parametric surfaces at all, and spatial partitioning, particularly using quaternary subdivision and quad-tree data structures \([24, 23]\) provided a very effective way of accomplishing this. By recursively subdividing the surface according to some predefined policy such as ‘cut at the centroid,’ spatial partitioning algorithms can enumerate many sample points very quickly. This kind of recursive process facilitates regular as well as adaptive sampling, and much if not all of the topology of the sample points is implied by the decomposition. On the other hand, the rigidity of the sampling sequence makes it difficult for partitioning algorithms to adapt smoothly to changes in curvature and create alignments with edges of non-rectangular domains. Presumably for these reasons, most of the sampling research has gone away from spatial partitioning as a generator of sampling sequences, however the elegant data structures it creates are still quite popular for accelerating computations wherever hierarchical pruning can be exploited, such as in collision detection \([13, 16, 18, 3, 12, 9]\), rendering \([8, 11]\), and motion planning \([15]\).

In contrast to the hierarchical behavior of spatial partitioning, advancing front algorithms, \([20, 1, 10]\) guess at the parameter space coordinates of a neighbor for some points already in the sample set. The quality of this choice is then evaluated according to some metric, and the guess is iteratively refined until a suitable point is found. As points are added to the sample set in this way, the set grows until it spans the entire surface. Such methods offer the benefit that they can guarantee the quality of meshes they produce, however they can be very slow due to the iterative searching. Furthermore, they cannot be terminated early, as that would result in an incomplete sample set. Lastly, the meshes produced by advancing front techniques contain no hierarchy, which makes them rather undesirable for applications where hierarchical pruning could be exploited. In practice advancing front approaches are quite popular in the engineering community, where powerful computers are available, and people are willing to devote significant resources to obtaining a high quality mesh to facilitate Finite Element Analysis (FEA) and other techniques, which rely primarily on marching around the mesh.
A family of algorithms, which can incorporate hierarchy, is based on the Delaunay triangulation. Such algorithms [2, 7] begin with some kind of simple base mesh, then subdivide the elements of that mesh according to Chew’s furthest point strategy [4], usually maintaining a priority queue such that the worst triangle in the set is always subdivided, until all triangles satisfy some quality criteria. This approach avoids searching, however there is still significant overhead associated with maintaining priority queues and constantly updating the mesh by flipping edges and other procedures. Consequently, Delaunay based meshing approaches are also quite expensive. The hierarchical nature of Delaunay triangulations means that pruning can be applied in order to facilitate fast rendering, as demonstrated by Chhugani and Kumar [5, 6], and the algorithms can be terminated early while still returning a mesh. Delaunay triangulations also offer some guarantees about triangle shape and size, although they are somewhat more relaxed than the guarantees made by the advancing front approaches.

An algorithm proposed by Vehlo, Figueiredo, and Gomez [22, 21] subdivides patches by searching for the subdivision that results in child patches, which approximate the surface best. The algorithm is designed to minimize the number of triangles necessary to approximate a surface to some given accuracy, and in light of this, it does not control triangle shape at all. Additionally, it requires the user to supply an initial mesh to specify the topology of the surface, and though they do not report any execution times, the authors admit the algorithm is 'slow'.

An enumerative sampling algorithm proposed by Quinn, Langbein, Martin, and Elber [19] is designed to produce statistically optimal sets of samples on arbitrary parametric surfaces. It reduces the 2D distribution problem to a 1D problem by mapping a space filling curve onto the surface. The space filling curve is generated in the parameter space and recursively refined according to its geometry in $\mathbb{R}^3$. Then the algorithm walks along the curve, distributing sample points in the parameter space according to well known low discrepancy distributions [17]. The algorithm delivers a well distributed set of samples, however connecting them would require extensive searching.

### 1.4 Approach

All of the methods previously discussed offer viable solutions to problems in sampling and meshing parametric surfaces. In this work, we seek to combine the speed and elegance of spatial partitioning with the ability of the advancing front and Delaunay based methods to produce high quality regular and adaptive tessellations of well shaped triangles. Our algorithm consists of five steps:

1. Sample each edge in the complex using spatial partitioning, and store the result in a binary tree.
2. Sample each surface in the complex using spatial partitioning, and store the result in a quad tree.
3. Discretize the parameter space of each edge and surface using the properties of the corresponding tree.
4. Deform the lattice of samples on each surface such that it conforms to trimmed edges.
5. Mesh the sample set on each surface using a combination of marching and recursive bisection in the discrete parameter space.

- We define a new five point patch primitive as the basis of our 2D spatial partitioning decomposition. This facilitates the creation of well shaped triangles, as well as the 'sewing' approach we use to close cracks in the mesh and attach it to the surface edges. (Section 2.1)

- We combine binary and quaternary subdivision of patches, which allows our decomposition to adapt to changes in curvature better than purely quaternary algorithms while remaining significantly faster and simpler to implement than purely binary ones. (Section 2.2)

- Prior to meshing we deform our lattice of sample points on each surface such that it conforms to any trimming curves that are present. This allows us to automatically conform to the topology of the surface, without requiring user input. (Section 3.2)

- Finally, we begin meshing by using the topological information from our tree structures, and we fill in the missing information by local search in a discrete parameter space, which facilitates extremely rapid mesh generation. (Sections 3.1, 3.3.1, 3.3.2)
2 Sampling Parametric Surfaces by Spatial Partitioning

In this section, we assume familiarity with binary and quaternary spatial partitioning for surfaces. For an in-depth description of these techniques, we refer the reader to [24].

2.1 Motivation for a Five Point Surface Patch Primitive

Consider a planar parametric surface, which we intend to triangulate using spatial partitioning. We use binary and/or quaternary subdivision to tessellate the surface with right quadrilaterals, and we use the lattice formed by the edges of these quadrilaterals as the basis of our mesh (figure 1 (a)). Finally we complete the mesh by connecting one diagonal on each quadrilateral to form two right triangles.

Now consider the same tessellation, but instead of using the edges of the quadrilaterals as the basis of our mesh, we connect the four corners of each quadrilateral to its centroid (figure 1 (b)). This arrangement allows us to shear the lattice by changing the aspect ratio of the constituent quadrilateral patches. In fact for this planar case, choosing patches with aspect ratio \( \sqrt{3} \) results in a quadrilateral lattice with angles 60 and 120 degrees. By connecting the correct diagonals (flipping some patch edges), we can construct a mesh with equilateral triangles everywhere except the edges.

This is the primary reason for the five point surface patch primitive pictured in figure 2, however it also offers the following benefits over the traditional four point approach:

- The amount of information in each patch is increased. We are therefore able to make better decisions about how to subdivide early in the decomposition (when our surface approximation is poor) as well as how to subdivide degenerate surface patches. We also need fewer recursions to generate a particular number of samples.

- The fifth point facilitates the ‘sewing’ procedure we use to close cracks and attach the meshes to the edges of their respective surfaces.

- The additional point also allows us to easily define a bounding ball for the patch without any additional computation. This is quite useful for hierarchical pruning applications such as collision detection and path planning.

![a) 4-Point Patch Primitive  b) 5-Point Patch Primitive](image)

Figure 1: Base meshes (thick lines) for 4-point and 5-point patch primitives. The 4-point primitive produces an orthogonal base mesh regardless of the aspect ratio of the patch, whereas the 5-point primitive allows the base mesh to be sheared by changing the patch aspect ratio.

2.2 Motivation for a Binary-Quaternary Subdivision Scheme

The other defining characteristic of our spatial partitioning algorithm is that we use both binary and quaternary subdivision. Quaternary subdivision and the associated quad-tree data structure offer superior speed and compactness, in that the branching of the decomposition is well optimized for 2D space, however it offers poor control over the local sample distribution because it cannot alter the aspect ratio of the surface patches. Binary subdivision on the other hand can and in fact must alter the aspect ratio of the patches and therefore offers improved control over the sample distribution, however due to its lower branching rate, it is significantly slower than the quaternary approach. Additionally, it can be difficult to determine in which
Area: sum of the areas of triangles $abm$, $acm$, $cdm$, and $bdm$

Aspect Ratio: \(\frac{ab+cd}{ac+bd}\) or its reciprocal, whichever is $\geq 1$

Curvature: The cosine of the largest angle between any two surface normals

Warp: The cosine of the smallest angle of $abd$, $bdc$, $dca$, and $cab$

Figure 2: The definition of a 'surface patch'

direction degenerate patches should be subdivided. This causes problems near singular points such as the poles of a sphere. Consequently, we have developed a hybrid binary-quaternary sampling decomposition, where we use quaternary subdivision unless it is clear that binary subdivision would improve the aspect ratio of the surface patch. This results in an algorithm that is nearly as fast as pure quaternary algorithm while it produces much better sample distributions, especially where the parametric mapping causes extreme distortion between parameter space and $\mathbb{R}^3$.

2.3 The Sampling Algorithm

Our spatial partitioning algorithm begins by sampling each edge in the complex, using recursive binary subdivision, until each line segment in the piecewise linear approximation of the edge curve is shorter than our target length, which we define to be the approximate length of the sides of the smallest surface patches we would like to see in our decomposition. In practice, the target length must be smaller than the characteristic length of the smallest features we hope to represent. Once this is done, we store the samples in an array, so that later, we can easily march along the edge.

Once the edges have been sampled, we move on to the faces. We first find the edges, which are associated with a particular face and map each sample point on each edge from $\mathbb{R}^3$ back into the parameter space of the face in question. In this way, we develop a set of piecewise linear curves, which represent the edges of the face in parameter space.

Then we begin sampling the face using a combination of binary and quaternary subdivision. Because the domains of the surfaces in the complex may not be rectangular, we do a PMC test for each sample point in the decomposition with respect to the face on which it lies, and points that happen to fall on edges are considered 'out'. We then examine the surface patch in parameter space to determine whether or not it intersects with any of the edge curves. We refer to this as the Patch-PMC (PPMC) test. All patches in the decomposition can therefore be classified with respect to the face as follows:

- 'ON' - PPMC=true (the patch is on an edge)
- 'IN' - PPMC=false, PMC='in' for all points (the patch is inside the face and covers no holes)
- 'OUT' - PPMC=false, PMC='out' for all points (the patch is completely outside the face)

Patches that are 'OUT' are immediately discarded, and 'ON' patches are subdivided as long as subdivision brings the patch area closer to the square of the target length, such that the sampling density near the edges of the face is similar to the sampling density on the edges. Patches that are 'IN' may be subdivided further or not, depending on what kind of subdivision rule is employed.

Algorithm 1 shows a practical curvature adaptive subdivision rule, which is intended to generate (nearly) equilateral triangles on arbitrarily curved parametric surfaces. The rule uses binary subdivision to bias the patch aspect depending on the warp (figure 2). Above a threshold, binary subdivision is used to make the patch more square, and below the threshold it is used to bias the aspect ratio toward $\sqrt{3}$. 

Algorithm 1
We justify the selection of this particular rule as follows: Clearly, on a planar surface with no distortion between the parameter space and \( R^3 \), patches of aspect ratio \( \sqrt{3} \) result in a perfect hexagonal packing (figure 1 (b)). However when surfaces have appreciable warp (figure 2), the patch edges are no longer orthogonal, and experimentation has shown that we produce better triangles by driving the patch aspect ratio toward 1.

We use this subdivision rule with the threshold set to 0.2 to generate the meshes pictured in figure 9, however the user is free to replace this rule with any function of the surface patch in figure 2 that returns two booleans, which control subdivision in the \( u \) and \( v \) directions in parameter space. By using different subdivision rules, we have been able to generate meshes with a variety of geometric properties.

<table>
<thead>
<tr>
<th>input</th>
<th>parent_node, target_curvature, target_area</th>
</tr>
</thead>
<tbody>
<tr>
<td>output:</td>
<td>divU, divV</td>
</tr>
</tbody>
</table>

```plaintext
SUBDIVIDE(parent_node) begin
  divU = false, divV = false;
  if parent_node.curvature < target_curvature OR parent_PPMC then
    if \( [\text{parent_node.warp} < \text{warp\_threshold} \ \text{AND} \ \text{parent_node.aspect\_ratio} < \sqrt{2} \ \text{OR} \ \text{parent_node.aspect\_ratio} > \sqrt{2} \ \text{OR} \ \text{parent_node.aspect\_ratio} > 4 \frac{\sqrt{3}}{3} ] \ \text{OR} \ \text{parent_node.aspect\_ratio} > \sqrt{2} \) then
      if \( (\text{parent_node.area} - \text{target\_area})^2 > (\text{parent_node.area}/2 - \text{target\_area})^2 \) then
        DO BINARY SUBDIVISION;
        divU ← true OR divV ← true;
      end
    end
  elseif (parent_node.area - target_area)^2 > (parent_node.area/4 - target_area)^2 then
    DO QUATERNARY SUBDIVISION;
    divU ← true AND divV ← true;
  end
end
```

Algorithm 1: SUBDIVIDE(parent_node)

### 2.4 Termination Criteria

In order to capture the topology of the B-rep, we must clearly capture the topology of each surface. This requires that we have adequate sampling density in the neighborhood of all edges. The criteria by which we can ensure adequate sampling density in these areas are as follows:

- No surface patch should intersect more than one edge unless it contains the vertex between the edges and has at least one sample point with PMC='in'. This is because we do not consider triangles with all three points on edges (see the discussion in section 3.3.1).
- Each vertex of the B-rep must have a unique nearest point in the sample set with PMC='out'. This ensures that the deformation step (section 3.2) includes both vertices of each edge, and during the 'sewing' step (section 3.3.1) every edge point is utilized.
- Each loop of one or more edges that surrounds an 'out' region must contain at least one entire patch with PPMC=false, ensuring that we cannot define triangles that traverse holes.

### 3 Meshing the Complex

Upon completion of the sampling phase, we have a binary tree of samples for each edge and a quad-tree of samples for each surface in the B-rep, and much of the topology of these points is implied by the hierarchical surface decomposition. In this section we describe the methods by which we complete the meshes on each face and attach them to the relevant edges.
3.1 Discretization of Parameter Space

Because subdivision occurs at the centroid of the patch in parameter space, it results in a lattice such that all points can be grouped into rows and columns of constant parameter (figure 1). By inspecting the properties of this lattice, we can define a matrix, which functions as a discrete container for our piecewise linear surface and is tremendously useful for finding the remaining topology we need to complete the mesh, namely closing cracks and determining which edges to flip.

Consider a hybrid binary-quaternary tree, such that binary branching nodes represent division of a surface patch in the $u$ or $v$ direction, whereas quaternary branching nodes represent a division in both directions. If during the construction of the tree, we count the number of divisions in $u$ and $v$ that lead to each leaf node, then we know that in the worst case, we have divided the root node of the tree $m$ times in $u$ and $n$ times in $v$. Therefore, the tree can contain no more than $2^{m+n}$ patches and $(m+1)(n+1) + mn + (m-1)(n-1)$ unique parameter space points, which are arranged in $i = 2^{m+1} + 1$ rows and $j = 2^{n+1} + 1$ columns. We use this information to discretize the parameter space, creating an $i$ by $j$ matrix shown graphically as the checkerboard in figure 3 (b).

In order to populate the matrix, we traverse the tree and plot each sample point according to its parameter space coordinates. At the same time, we update the tree structure to refer to the appropriate matrix element. Redundant points overwrite their twins, and thus is redundancy in parameter space eliminated from the data set. The tree is no longer directly linked to the point structures (figure 3 (a)), but rather to an element of the parameter space matrix, which is in turn linked to exactly one point structure (figure 3 (b)). Clearly, the points can still be accessed through the original tree, but now they can be accessed directly through the matrix as well, facilitating the use of marching and recursive bisection methods in the parameter space. With these methods we can efficiently close cracks and decide which edges to flip.

3.2 Deformation of Lattices

For each face (trimmed surface) in the B-rep there will be surfaces patches, which lie on the edges. These have PPMC=true, and as such, have one or more points which are exactly on the edge or outside the face (PMC='out'). By moving these 'out' points to the nearest edge point, we can deform the edges of our (slightly too large) surface mesh to conform to the trimmed edges of the surface (figure ?? (a) and (b)). We do this in two steps.

First we look at each vertex (as described in section 2.4) and find the nearest 'out' point among the samples on the face. We then replace the pointer in our discrete representation of parameter space (the matrix in figure 3) with a pointer to the vertex in the appropriate edge array. Each vertex must have a unique 'out' point from the surface mesh, otherwise the deformed surface mesh will be incomplete. In other
words, the lattice must have enough points to conform to the geometry and topology of the underlying surface.

In the second step, we look at each ‘out’ point in the discrete parameter space and replace it with a pointer to the nearest edge point. Here we can reuse the same edge point repeatedly without problems. Once this is complete, the mesh has been deformed in \( \mathbb{R}^3 \), and it conforms to the geometry of its trimming curves. However, the structure of the lattice in parameter space has not been affected, as points are not moved from one location to another in the parameter space matrix. Therefore, we can carry out marching and recursive bisection procedures (section 3.3) in the parameter space as though we had not deformed the mesh at all.

### 3.3 Connecting the Mesh

In order to connect the mesh, we visit each leaf node in the decomposition tree (figure 3 (a)), and inspect the relevant points in the parameter space matrix (figure 3 (b)). We first check for any patch edges which have both points on an edge of the surface due to the aforementioned lattice deformation. These patch edges, which are also surface edges, are handled as described in section 3.3.1. The remaining edges, which are on the face, are checked for cracks as described in section 3.3.2, and finally edges that are on the face and not involved in cracks may either be preserved or flipped as described in section 3.3.3.

#### 3.3.1 ‘Sewing’ the Mesh to Surface Edges

Since we have arbitrarily curved edges, triangles with all three points on the edge may be ‘in’ (figure 4 (b) triangle \( abc \)) or ‘out’ (figure 4 (b) triangle \( cde \)) with respect to the face. In the former case, the triangle is topologically invalid, and in the latter case it is geometrically invalid. Therefore, we simply adopt the policy that we consider only triangles that have two points on the edge, and one point on the face (PMC=’in’). This leads us to throw away triangles \( abc \) and \( cde \), resulting in the mesh shown in figure 4 (c).

![Figure 4: Mesh Deformation and Triangle Selection](image)

In the previous example, all of the edge points were accounted for after deformation, however consider that the two end points of a patch edge, \( p \) and \( q \), are deformed to non-adjacent points on a trimmed edge. In this case, we must connect the patch to every edge point that lies between \( p \) and \( q \), such that each surface mesh in the complex is attached to every sample point on every edge that belongs to that surface. In this way we can guarantee that our mesh will be valid and will reflect the topology of the B-rep as a whole. We simply use points \( p \) and \( q \), which are pointers to an edge array, to define an interval on the edge. We march from \( p \) to \( q \) defining triangles using each element of the piecewise linear edge approximation and point \( m \) (figure 2). If point \( m \) is itself on the edge, we must look for a corner point \((a,b,c,\) or \(d)\) that is on the face, and use it in place of point \( m \).

#### 3.3.2 Closing Cracks

The tree structures produced by spatial partitioning (figure 3 (a)) are generally not of a uniform depth. Consequently, there exist surface patches in the decomposition, which have more than one neighbor on a side. This situation results in an ambiguous edge, which is commonly referred to as a ‘crack’. Where a surface patch has exactly two neighbors to one side, as in figure 5, the topology of the crack is triangular. Therefore, one solution to the cracking problem is to enforce that patches never have more than two neighbors to a
side, and to simply include the crack region(s) in the triangulation. While this solution is valid (in fact it is the solution employed by Von Herzen in [24], which he calls the 'Restricted Quad-Tree'), it generally produces triangles, which have high aspect ratios, and normals that are not well aligned with those of the surrounding faces. Additionally, the policy introduces parallel dependency into the tree, such that the recursive decomposition is no longer easily parallelizable. Finally, the restricted quad tree does not allow the structure shown in figure 6, but rather it would force oversampling and the creation of high aspect patches near the annulus of the torus.

Figure 5: A crack in a section of a conic

Figure 6: A crack in a section of a torus

In an effort to support the structure shown in figure 6, as well as to produce better triangles with better normals than the restricted quad-tree and avoid parallel dependency, we close the cracks as shown in figure 7. We accomplish this by using a recursive algorithm, which operates in the discretized parameter space. Consider edge \( cd \) in figures 5 and 6. Before adding triangle \( cdm \) to the mesh, we use a recursive function to look for a bisector \( p \) on the edge \( cd \) in the parameter space matrix. If there is a bisector \( p \), we recursively look for a bisector on \( cp \) and \( pd \). We recurse until we no longer find a bisector, and finally define a triangle using the two adjacent points and point \( m \), as shown in figure 7.

Figure 7: Closed cracks

3.3.3 Flipping Patch Edges

In the previous section we described how patch edges are checked for cracks. In the event that an edge is not involved in a crack, it becomes possible to flip the edge as shown in figure 8. Consider an edge \( ab \), which separates two patches having midpoints \( m_1 \) and \( m_2 \). If edge \( ab \) is not involved in a crack, we can choose to define either triangles \( abm_1 \) and \( abm_2 \), preserving the patch edge, or \( m_1m_2a \) and \( m_1m_2b \), flipping the patch edge. When we visit a patch during meshing, we clearly have access to points \( a, b, \) and \( m_1 \), and in order to
facilitate the edge flipping we must only find the point \( m_2 \). We accomplish this by marching (in the \( \pm u \) or \( v \) direction) in the parameter space matrix, and once we find an \( m_2 \), we must only check that the patches are indeed adjacent, by looking for the common points \( a \) and \( b \) in the patch that contains \( m_2 \). If we find the patches are adjacent, we can decide whether or not to flip the edge according to any relevant quality metric. For our purposes, we choose the pair of triangles according to their shape.

\[ \begin{align*}
\text{Figure 8: Two Alternatives (gray lines) for Connecting Patches}
\end{align*} \]

4 Experimental Results

In lieu of a standard set of benchmarking tests, we have chosen to present experiments on models which are similar to those in other recent work (figure 9 elbow bracket, triangle bracket), as well as a model with a deliberately undesirable parameterization (figure 9 rotated elbow), a real engineering part (figure 9 caliper), and several Non-Uniform Rational B-Spline (NURBS) surfaces which exhibit a great deal of distortion between parameter space and \( \mathbb{R}^3 \) (figure 9 bottle, chair) and many local changes in curvature (figure 9 face).

The meshes in figure 9 were generated on a PC with a single 3.2 GHz processor and 4 Gb of memory. In terms of performance, we compare our run times to those of the recent Delaunay based algorithm by Dey and Levine [7] as well as to the advancing front algorithm from Guan, Shan, Zheng, and Gu [10], both of which report execution times for generating similar numbers of triangles on comparable hardware.

Figure 10 shows that we can generate on the order of tens of thousands of triangles per second for all of the models. Dey and Levine generate hundreds of triangles per second, and Guan et al. generate just over one thousand, however they base this claim on one model, which consists of flat and cylindrical surfaces, in fact surfaces most conducive to the advancing front approach. Therefore we claim that our approach is approximately two orders of magnitude faster than Delaunay based methods and at least one order of magnitude faster than the advancing front. Moreover, unlike Delaunay and Advancing approaches, ours is completely conducive to multi-threaded implementations. One final comment on run time is that we have thus far made no effort to optimize our searches for points during the mesh deformation step. In fact we do an exhaustive search of all the edge points to ensure robustness under all parameterizations. This is a procedure which takes an appreciable amount of CPU time, and it would be an interesting topic for further research to develop strategies to improve the search.

5 Discussion and Conclusions

In this work we have introduced a new two-part sampling and meshing algorithm based on spatial partitioning. Our algorithm is capable of generating point clouds on arbitrary parametric B-reps without the added overhead of meshing. Additionally, it can triangulate the point clouds to form topologically correct meshes without the need for a user-defined base mesh or any other kind of topological information not intrinsic to the model at hand.

Our algorithm is much faster than other popular approaches such as advancing front and Delaunay based techniques, while still affording the user rule based control of the sample distribution. The hierarchical nature of the sample set and associated mesh can be exploited wherever hierarchical pruning is an appealing search strategy, and our meshes are therefore conducive to rendering, collision detection, and path planning.
Figure 9: From top-left to bottom-right: Elbow Bracket, Rotated Elbow, Triangular Bracket, Caliper, Bottle, Chair, Face

Figure 10: Experimental Data: Shape refers to the triangle shape metric defined by Knupp in [14], and time is reported in seconds.
Finally, in contrast to Delaunay and advancing front techniques, our algorithm lends itself to multithreaded implementations.

The primary drawback of our approach is that we cannot guarantee triangle shape, and in practice we do generate some poorly shaped triangles. These are primarily located adjacent to the edges of the B-rep, and the mechanism that causes this is clearly visible in figure 4 (b). A secondary source of poorly shaped triangles is the closing of large cracks in the mesh (when a patch has four or more neighbors to one side). This is visible in the exploded view of the ‘rotated elbow’ model in figure 9. These poor triangles are however relatively few, and their causes are clear. We are therefore interested in pursuing strategies to improve the worst triangles in our meshes.

References


