FAST HIERARCHICAL DISCRETIZATION OF PARAMETRIC BOUNDARY REPRESENTATIONS

MIKHAIL FRANK

Dalle Molle Institute for Artificial Intelligence
University of Lugano & SUPSI
Manno-Lugano, Switzerland
kailfrank@gmail.com

HOREA T. ILIEŞ

Computational Design Laboratory
University of Connecticut
Storrs, USA
ilies@engr.uconn.edu

Prevalent discretization methods based on Delaunay triangulations and advancing fronts, which sample and mesh simultaneously, can guarantee well shaped triangles but at a fairly high computational cost. In this paper we present a novel and flexible two-part sampling and meshing algorithm, which produces topologically correct meshes on arbitrary boundary representations whose faces are represented parametrically, without requiring an initial coarse mesh. Our method is based on a hybrid spatial partitioning scheme driven by user-designed subdivision rules that combines the power of quadtree decomposition with the flexibility of the binary decompositions to produce meshes that favor prescribed geometric properties. Importantly, 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. At the same time, our algorithm is practically as fast as the computationally optimal algorithm based on a pure quadtree decomposition, but with a markedly better distribution in the regions with parametric distortion. The hierarchical nature of our surface decomposition is well suited to interactive applications and multithreaded implementation.

Keywords: Discretization; Parametric Surfaces; Sampling; Triangulation; Boundary Representations.

1. Introduction

Parametric representations have become a *de facto* standard in geometric and solid modeling due to a number of favorable differential and computational properties. For example, curves and surfaces represented parametrically offer the significant benefit that points belonging to these entities can be enumerated very quickly.

1.1. *Motivation*

Despite their utility in the context of modeling, parametric surfaces can be quite cumbersome. Intersections, distance computations, and Point Membership Classifi-

cation (PMC) are all expensive operations where parametric surfaces are concerned, which is why parametric representations are not used as native representations in time-sensitive applications such as rendering, motion planning and collision detection. 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 solid model. 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 facilitate efficient computations in downstream applications.

One can view the problem of parametric surface discretization in its most general form as an optimization problem, consisting of a chosen objective function as well as the set of all possible problem instances, or surfaces, over which the objective must be optimized.

The size and complexity of the set of all parametric surfaces is problematic for discretization algorithms, because they must cope with a plethora different geometries and topologies. This makes it very likely that the optimization of even simple objective functions becomes NP-hard, and in practice we have to settle for approximate solutions. Therefore, a surface discretization algorithm's capacity to adapt to challenging instances and return a reasonably good result is critically important.

1.2. *Problem Statement*

The problem of discretizing parametric surfaces can be thought of in terms of two major deliverables: (1) a sample point set, and (2) the associated topology. For some applications, the samples alone are sufficient. However it is often necessary to establish connectivity between the sampled points in order to specify their collective topology.

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 challenge 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 in the Euclidean space in which the parametric surfaces are embedded, we must solve a $2D$ distribution problem under the uncertainty imposed by the uncountably many mappings between the parametric and Euclidean spaces encountered in practice.

In the second step, polygonization, we must connect the points such that they

form a triangular mesh. The sample distribution is already determined[a], 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 the boundary representation (B-rep), such that we can join adjacent face meshes according to valid mesh topologies.

We aim to construct a highly efficient procedure capable of outputting a high quality regular and adaptive tessellation of well shaped triangles for B-reps whose faces are represented parametrically. Clearly, such a procedure requires the ability to sample each face as well as the capability to connect the meshes of two adjacent faces across the edges.

As it is usually the case, the computational cost and quality of the resulting tessellation are competing objectives. We argue that the hybrid hierarchical subdivision that we propose in this paper retains the speed and elegance of spatial partitioning approaches with the ability of search based methods to produce surface tessellations with prescribed geometric properties. Importantly, the quality of the resulting mesh improves the prescribed quality metric as the subdivision progresses.

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. Discretization methods were absolutely necessary to visualize parametric surfaces at all, and spatial partitioning – particularly using quaternary subdivision and quad-tree data structures [25,24] provided a very effective way of accomplishing this. By recursively subdividing the surface according to some predefined policy, spatial partitioning algorithms can enumerate many sample points very quickly. This kind of recursive process facilitates regular as well as adaptive sampling. Importantly, 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. Furthermore, the algorithms based on quaternary subdivision imply the mesh topology, but are driven by a single subdivision policy (i.e., a single and rigid objective function) with limited control over the resulting tessellation. 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 implied by these decompositions 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].

[a]As explained in section 1.3, one can treat the sampling and polygonization as either independent or coupled steps of the meshing process.

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 point 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, but they can be very slow due to the *required* 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 those applications in which a high quality mesh is more important than the speed with which they can be computed, such as Finite Element Analysis.

A family of algorithms, which can incorporate hierarchy, is based on the Delaunay triangulation. The idea behind all these algorithms is fairly straightforward: start with a set of points on the surface, build some kind of triangulation to obtain a base mesh, and insert additional points while updating the connectivity to improve certain quality metrics [2,7]. These algorithms can be terminated early during the point insertion process while still returning a mesh. Furthermore, pruning can be applied in order to facilitate fast rendering by augmenting the Delaunay triangulation with a hierarchical point sample set, as demonstrated by Chhugani and Kumar [4,5]. However, the computational cost of inserting the additional points and of updating the mesh is significant [6] and guaranteeing termination of the insertion procedure remains a challenge.

An algorithm proposed by Vehlo, Figueiredo, and Gomez [23,22] 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 (see section 2.3):

(1) Sample each edge of the B-rep using spatial partitioning, and store the result in a binary tree.
(2) Sample each face in the B-rep cell complex using a hybrid spatial partitioning driven by a user-designed subdivision, and store the result in a quad tree.
(3) Discretize the parameter space of each edge and face using the properties of the corresponding tree.
(4) Deform the lattice of samples near the edges of each face such that it conforms to trimmed edges.
(5) Mesh the sample set on each face using a combination of marching and recursive bisection in the discrete parameter space.

1.5. *Main Features of This Approach*

- We define a new five point patch primitive as the basis of our $2D$ spatial partitioning decomposition (section 2.1). This facilitates the creation of well shaped triangles, as well as supports the 'sewing' we use to close cracks in the mesh and attach it to the face edges (section 3.3).
- We combine binary and quaternary subdivision of patches in a new subdivision scheme driven by user-designed subdivision rules. Our scheme combines the the power of quaternary decomposition with the flexibility of the binary subdivision. In practical terms, our decomposition adapts to changes in the quality metric 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 face such that it conforms to any trimming curves that are present. This allows us to automatically conform to the topology of the face, 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 to obtain meshes that favor prescribed geometric properties. (sections 3.1, 3.3.1)

2. Sampling Parametric Faces 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 [25].

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, choosing patches with aspect ratio $\sqrt{3}$ for this planar case 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 (figure 1 (c)).

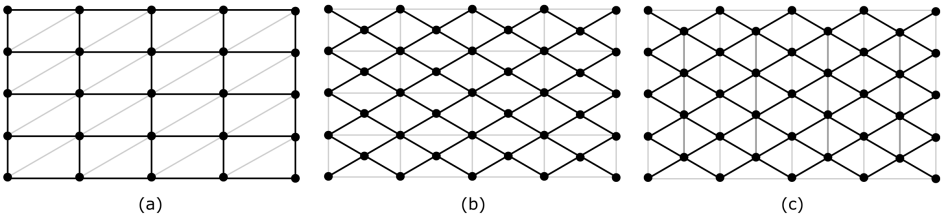
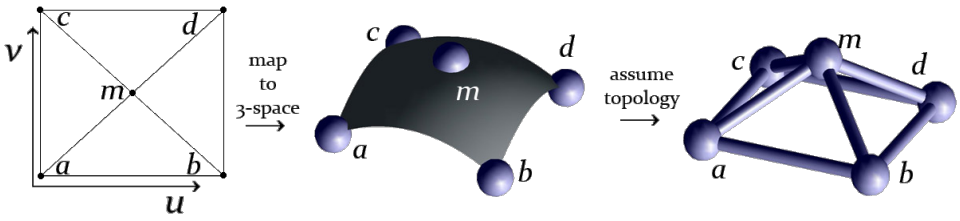


Fig. 1. Base meshes (thick lines) for 4-point and 5-point patch primitives. The 4-point primitive (a) produces an orthogonal base mesh regardless of the aspect ratio of the patch, whereas the 5-point primitive (b) allows the base mesh to be sheared by changing the patch aspect ratio (c).

This is the primary reason for the five point surface patch primitive pictured in figure 2. At the same time, this patch 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 construct a bounding ball for the patch that can handle four coplanar vertices of the patch.



Area: sum of the areas of triangles *abm*, *acm*, *cdm*, and *bdm*
Aspect Ratio: $\frac{(ab+cd)}{(ac+bd)}$ or its reciprocal, whichever is ≥ 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*

Fig. 2. The definition of a 'surface patch'

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 quaternary subdivision 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. On the other hand, due to its lower branching rate, the binary partitioning is significantly slower than the quaternary approach. Additionally, it can be difficult to determine in which 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 the mesh with respect to the chosen subdivision metric. 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 B-rep, using recursive binary subdivision, until each line segment in the piecewise linear approximation of the edge curve is shorter than our target length[b], 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

[b]Observe that a more effective implementation would vary the edge subdivision strategy along with the face sampling strategy discussed in this section.

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 parametric domains of the faces in the B-rep are typically not 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 until the local sampling density on the face resembles that on the edge. 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 favors the generation of 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 $\sqrt{3}$. To justify the selection of this particular rule, observe that on a planar surface with no distortion between the parameter space and $\mathbb{R}^3$, 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.

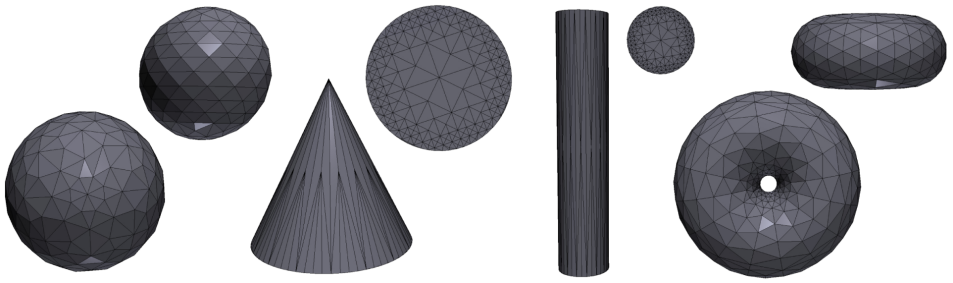
We use this subdivision rule with the threshold set to 0.2 to generate the meshes pictured in figure 10. Note that this subdivision rule can be replaced with any function of the surface patch in figure 2 that returns two Booleans that control subdivision in the u and v directions in parameter space. By using different subdivision rules, we have been able to generate meshes with a variety of geometric properties.

To illustrate the influence of the subdivision rule on the resulting tessellation, we implemented a second subdivision function (algorithm 2) that only takes into

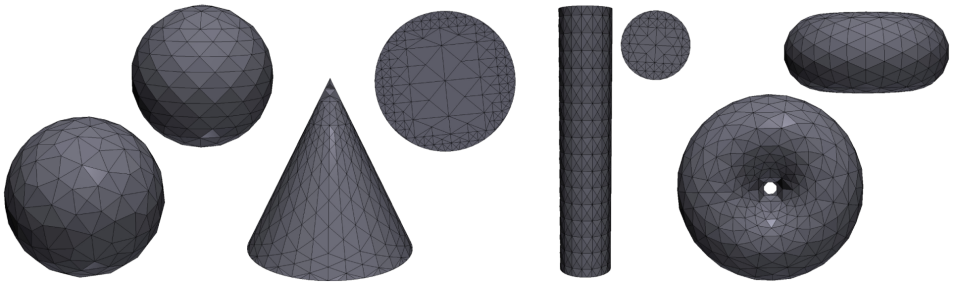
```
input  : parent_node, target_curvature
output: divU,divV

SUBDIVIDE(parent_node) begin
    divU = false, divV = false ;
    if parent_node.curvature > target_curvature OR parent_PPMC then
        if [parent_node.warp < warp_threshold AND
        (parent_node.aspect_ratio < √2 OR parent_node.aspect_ratio > 4√3/3)]
        OR parent_node.aspect_ratio > √2 then
            DO BINARY SUBDIVISION;
            divU ← true OR divV ← true ;
        end
        else
            DO QUATERNARY SUBDIVISION;
            divU ← true ;
            divV ← true ;
        end
    end
end
```

Algorithm 1: CURV_AR_SUBDIVIDE(parent_node) performs curvature adaptive subdivision while controlling the aspect ratio of patches in the decomposition

account the curvature of the surface. Figure 3(a) shows the top and side view tessellations of a sphere, cone, cylinder and torus that result from the subdivision rule based only on surface curvature, i.e., no shape control, and outlined in algorithm 2. By comparison, the subdivision that takes into account both surface curvature and patch aspect ratio results in the surface tessellations shown in Figure 3(b). The statistics corresponding to the two subdivision rules and detailed in Table 1 show that the subdivision rule that controls the patch aspect ratio does indeed improve triangle shape (min, mean and max). The metric *shape* used in all statistics reported in this paper refers to the triangle shape metric defined by Knupp in Ref. 14. For the sphere it also predictably increases the mean Hausdorff distance, although surprisingly the best triangles (in terms of minimum Hausdorff distance) are better that their counterparts from the pure curvature subdivision. The statistics shown in Table 1 point out the fact that the two sphere tessellations are qualitatively similar (similar shape and Hausdorff distance metrics), most likely due to the constant curvature of the sphere. On the other hand, the cylinder, cone and torus examples clearly demonstrate the different behavior of our two subdivision rules. The rule that controls both aspect ratio and curvature drastically improves the minimum, mean and maximum triangle shape on all three primitives, while the purely curvature based rule results in mean Hausdorff distances an order of magnitude smaller.



(a)



(b)

Fig. 3. Tessellations produced by (a) controlling only surface curvature; and (b) both curvature and aspect ratio.

Note that the meshes that possess lower mean Hausdorff distances in Table 1 tend to have less triangles, a feature that is more pronounced for the cone and the cylinder.

2.4. *Termination Criteria*

In order to capture the topology of the B-rep, we must clearly capture the topology of each face. 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:

- 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.

	# of Triangles	Shape			Approx. Hausdorf Distance		
		min	mean	max	min	mean	max
Sphere	384	0.727	0.906	0.991	0.00639	0.0681	0.0902
Cone	528	0.101	0.683	0.964	0	0.00126	0.0191
Cylinder	896	0.0452	0.709	0.937	0	0.000046	0.000428
Torus	800	0.339	0.850	0.991	0.000021	0.000470	0.000867

(a) Curvature Adaptive Subdivision *without* Shape Control

	# of Triangles	Shape			Approx. Hausdorf Distance		
		min	mean	max	min	mean	max
Sphere	368	0.830	0.908	0.999	0.00265	0.0701	0.0902
Cone	1044	0.200	0.801	0.999	0	0.0111	0.208
Cylinder	1392	0.240	0.813	0.997	0	0.000325	0.000428
Torus	856	0.522	0.910	0.999	0.000076	0.00637	0.0188

(b) Curvature Adaptive Subdivision *with* Shape Control

Table 1. Mesh statistics for common primitives shown in Figure 3. Metric *shape* refers to the triangle shape metric defined by Knupp. [14]

```
input  : parent_node, target_curvature
output: divU,divV

SUBDIVIDE(parent_node) begin
    divU = false, divV = false ;
    if parent_node.curvature < target_curvature OR parent_PPMC then
        if parent_node.Ucurvature > parent_node.Vcurvature then
            divU ← true;
        end
        else if parent_node.Vcurvature > parent_node.Ucurvature then
            divV ← true ;
        end
        else
            divU ← true; divV ← true ;
        end
    end
end
```

Algorithm 2: CURV_SUBDIVIDE(parent_node) performs curvature adaptive subdivision without controlling the aspect ratio of patches in the decomposition

- 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 Cell 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 face in the B-rep. Much but not all 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*

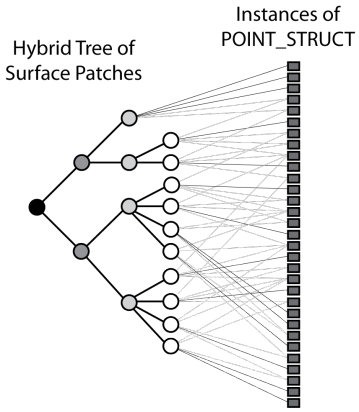
Because subdivision occurs at the centroid of the patch in parameter space, it results in a lattice such that all points can be grouped into rows and columns of constant parameter (figure 1). By inspecting the properties of this lattice, we can define a matrix, which functions as a discrete container for our piecewise linear surface. This matrix allows us to compute the remaining connectivity information in linear time with respect to the number of patches in order to complete the mesh, i.e., to close the cracks and to determine 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 4 (b).

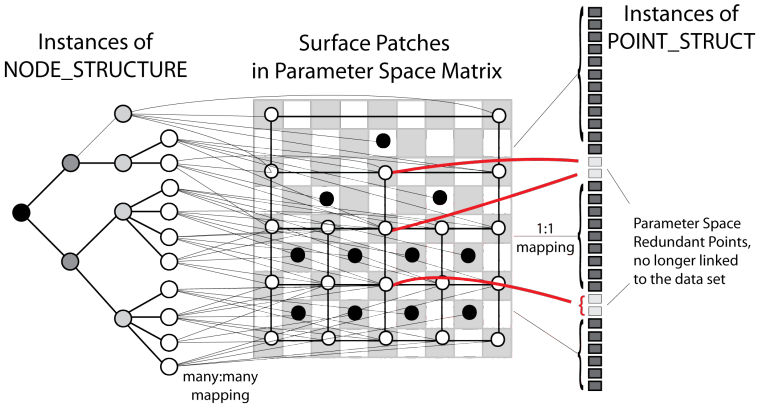
In order to populate the matrix, we traverse the tree and plot each sample point according to its parameter space coordinates. At the same time, we update the tree structure to refer to the appropriate matrix element. Redundant points overwrite their twins, which eliminates redundancy in parameter space from the data set. The tree is no longer directly linked to the point structures (figure 4 (a)), but rather to an element of the parameter space matrix, which is in turn linked to exactly one point structure (figure 4 (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 as discussed in sections 3.3.2 and 3.3.3.

3.2. *Deformation of Lattices*

For each face (trimmed surface) in the B-rep there will be surfaces patches that lie on the edges. These have PPMC=true (see section 2.3), and, as such, have one or more points which are exactly on the edge or outside the face (PMC='out'). By



(a) The tree directly linked to the points, some of which are redundant



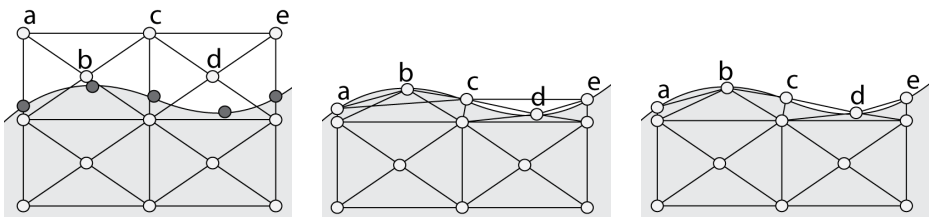
(b) The parameter space matrix filters redundancy and provides a space for performing marching procedures

Fig. 4. Discretized Parameter Space

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 (figures 5 (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 4) with a pointer to the vertex in the appropriate edge array. No two vertices of the B-rep can share a nearest 'out' point on the face, 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 face.

In the second step, we look at each 'out' point in the discrete parameter space



(a) Undeformed (b) After Deformation (c) Finished Edge

Fig. 5. Mesh Deformation and Triangle Selection

and replace it with a pointer to the nearest edge point. Here we can reuse the same edge point repeatedly without problems. Our implementation of this process is $O(n^2)$ in the number of edge-samples belonging to the face, but a better implementation would likely reduce it to $O(n \log(n))$. 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 4 (a)), and inspect the relevant points in the parameter space matrix (figure 4 (b)). We first check for any patch edges which have both points on an edge of the face due to the aforementioned lattice deformation. These patch edges, which are also face edges, are handled as described in section 3.3.1. The remaining edges, which are on the face, are checked for cracks. Finally, edges that are on the face and not involved in cracks may either be preserved or flipped as described in this section.

3.3.1. *'Sewing' the Mesh to Face Edges*

Since we have arbitrarily curved edges, triangles with all three points on the edge may be 'in' (figure 5 (b) triangle *abc*) or 'out' (figure 5 (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 5 (c).

In the previous example, all of the edge points were accounted for after deformation, but note 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 face. 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 4 (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 6(a), 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 [25], 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(b), but rather it would force oversampling and the creation of high aspect patches near the annulus of the torus.

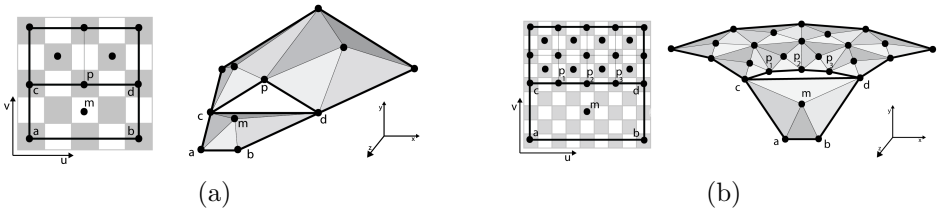


Fig. 6. A crack in a section of a conic(a) and torus (b)

In an effort to support the structure shown in figure 6(b), 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 6(a) and (b). 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

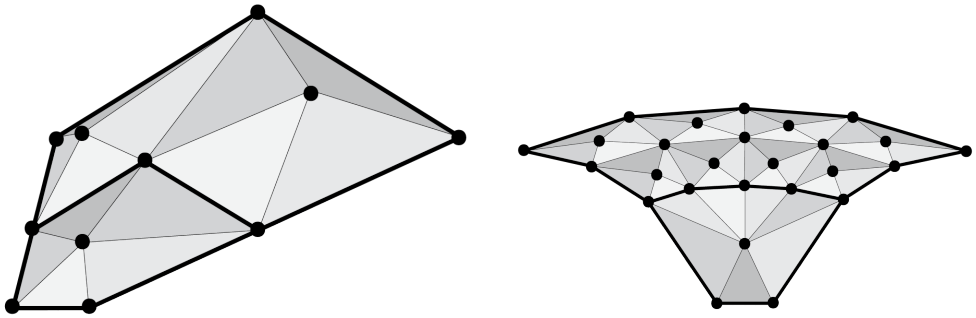


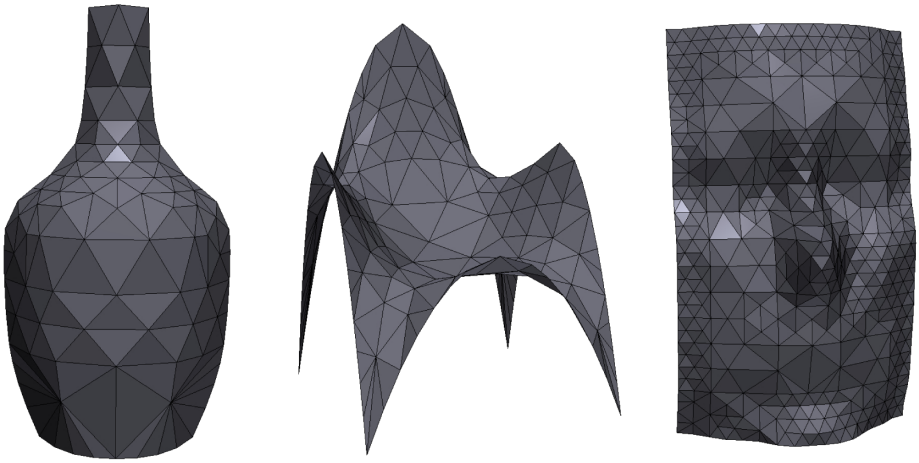
Fig. 7. Closed cracks

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.

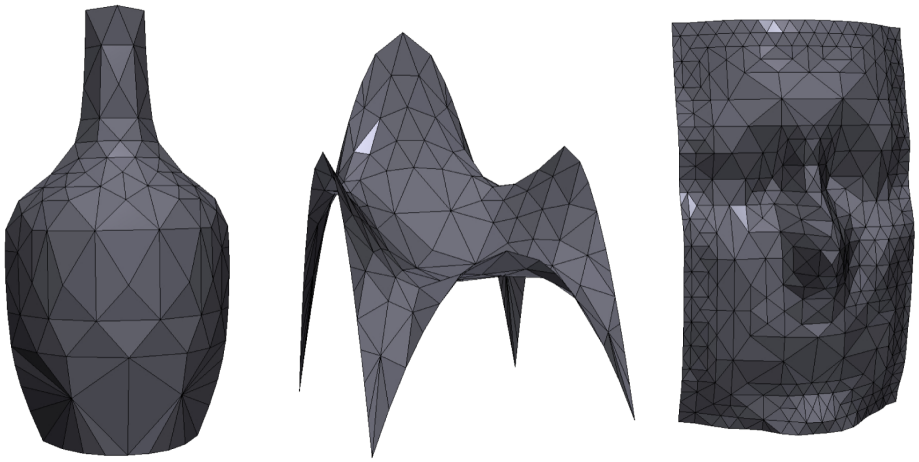
3.3.3. *Flipping Patch Edges*

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 1(b) and (c). 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.

Figure 8 shows the effect of 'optimizing' the edge flipping to improve either Hausdorff distance (i.e., decrease it), or triangle shape. In this experiment we take the same surfaces as before, and perform the first two levels of the subdivision, optimizing one mesh according to triangle shape (a) and the other according to the Hausdorff distance. All three surfaces, which exhibit many changes in curvature, show that the mesh optimization functions as expected. When we optimize the shape of the triangles we see a higher mean triangle shape and also a higher, and therefore detrimental mean Hausdorff distance as detailed in Table 2. Conversely, when we aim to decrease the Hausdorff distance, we see a lower mean Hausdorff distance and also a lower mean triangle shape. Our experiments indicate that mesh optimization does not usually affect the best and worst triangles in the mesh, although it did affect the worst shaped triangle on the face.



(a)



(b)

Fig. 8. Meshes of NURBS surfaces whose edges were flipped to improve (a) triangle shape; and (b) Hausdorff distance.

4. Experimental Results

In lieu of a standard set of benchmarking tests, we have chosen to present the results of two experiments: (1) a comparison of three hierarchical sampling strategies, and (2) meshes and timing on solid models which are similar to those in other recent papers (figure 10 elbow bracket, triangle bracket), as well a model with a deliber-

	# of Triangles	Shape			Approx. Hausdorff Distance		
		min	mean	max	min	mean	max
Bottle	152	0.311	0.863	1	0.000047	0.00107	0.00302
Chair	250	0.0423	0.755	0.999	0.000001	0.000058	0.000532
Face	728	0.426	0.891	1	0	0.000507	0.00557

(a) Meshes Optimized for Triangle Shape

	# of Triangles	Shape			Approx. Hausdorff Distance		
		min	mean	max	min	mean	max
Bottle	152	0.311	0.831	1	0.000047	0.000982	0.00302
Chair	250	0.0423	0.739	0.999	0.000001	0.000056	0.000532
Face	728	0.417	0.845	1	0	0.000452	0.00557

(b) Meshes Optimized for Hausdorff Distance

Table 2. Mesh statistics for NURBS surfaces shown in Figure 8.

ately undesirable parametrization (figure 10 rotated elbow), a real engineering part (figure 10 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 10 bottle, chair) and many local changes in curvature (figure 10 face).

Figure 9 illustrates two sets of meshes produced by a pure quaternary subdivision and our hybrid subdivision for a sphere (a), cone (b) and torus (c), which exhibit both geometric and parametric singularities, as well as regions of both constant and varying curvature. It can be observed that our hybrid subdivision produces meshes that are more regular throughout the surfaces, including those regions that are in the neighborhood of parametric and geometric singularities. Furthermore, Figure 9(d) shows that our algorithm is practically as fast as the computationally optimal algorithm based on a pure quadtree/quaternary decomposition.

The meshes in figure 10 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.

Table 3 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, but 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 for solid

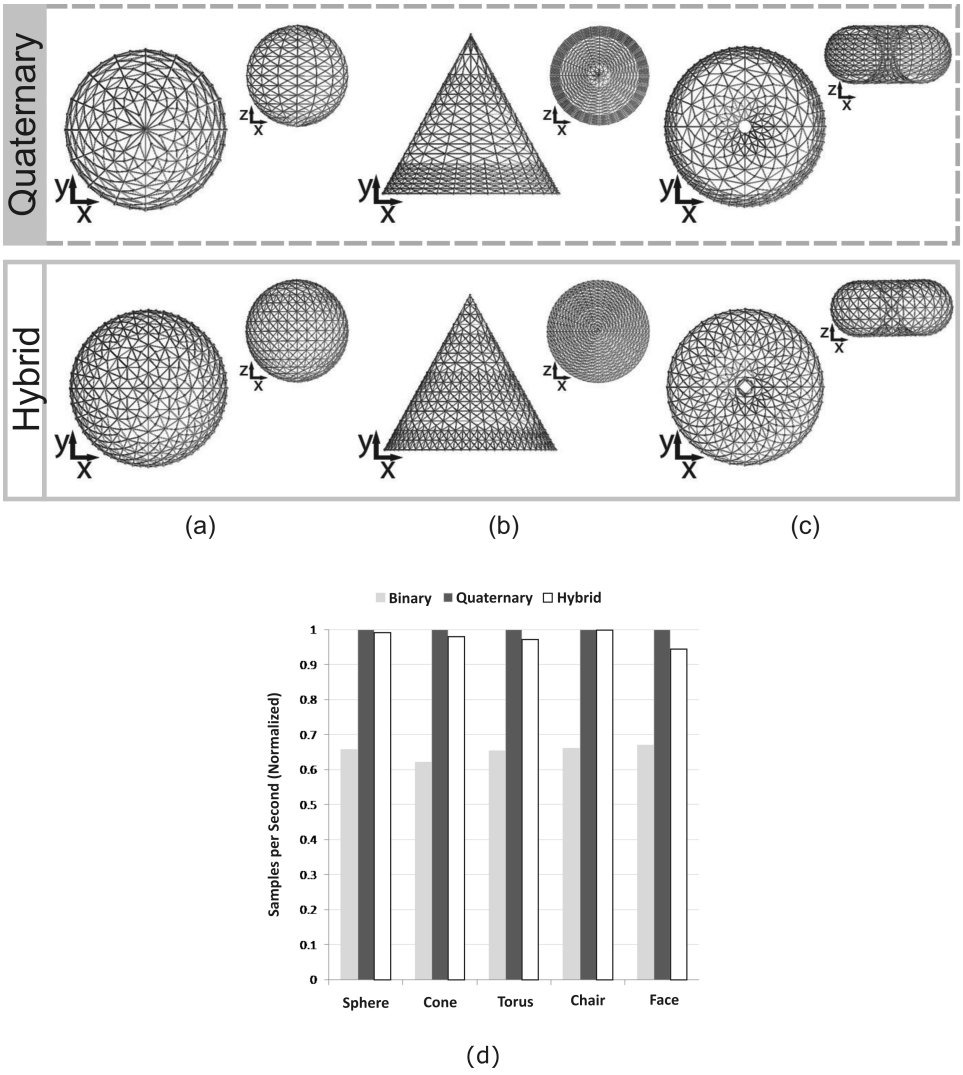


Fig. 9. A comparison of the meshes produced by a pure quaternary subdivision and our hybrid subdivision for a sphere (a), cone (b) and torus (c); The *normalized* number of samples per second for a binary, quaternary and hybrid subdivisions is shown in (d).

models of practical complexity. Moreover, unlike Delaunay and Advancing Front 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.

Fig. 10. From top-left to bottom-right: Elbow Bracket, Rotated Elbow Bracket (with rotated parameter space - note the difference in the resulting mesh particularly near the edges), Triangular Bracket, Caliper, Bottle, Chair, Face

5. 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

Part	# of Faces (CAD)	# of Triangles	Mean Shape	Execution Time
Elbow Bracket	16	14,204	0.890	0.437
Rotated Elbow	16	13,942	0.821	0.578
Triangle Bracket	26	18,956	0.859	0.671
Caliper	236	54,975	0.868	3.291
Bottle	1	1,180	0.875	0.063
Chair	1	2,144	0.770	0.120
Face	1	3,982	0.901	0.390

Table 3. Experimental Data: The metric *Shape* refers to the triangle shape metric defined by Knupp, [14] and time is reported in seconds.

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 significantly faster than other popular approaches such as Advancing Front and Delaunay based techniques, while still affording user-defined rules that control the sample distribution. Moreover, our algorithm is practically as fast as the computationally optimal algorithm based on a pure quaternary decomposition, but with a markedly better sample distribution and mesh regularity in the presence of parametric distortion. The hierarchical nature of the sample set and associated mesh can be exploited wherever hierarchical pruning is an appealing search strategy. Furthermore, our meshes are conducive to time-sensitive applications such as rendering, collision detection, and path planning. 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 adjacent to the edges of the B-rep, and the mechanism that causes this is clearly visible in figure 5 (b). A secondary source of poorly shaped triangles is the closing of large cracks in the mesh, i.e., 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 10. 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. For example, one immediate solution would be to run a standard remeshing algorithm such as the one detailed in [21] to locally remesh these triangles that are adjacent to the edges of the B-rep.

Acknowledgments

This work was supported in part by the National Science Foundation grants CMMI-0555937, CAREER award CMMI-0644769, CMMI-0733107, CMMI-0927105, CNS-0923158 and Connecticut Center for Advanced Technology. All 3D examples were

created by using Parasolid, courtesy of Siemens PLM.

References

1. M. Attene, B. Falcidieno, M. Spagnuolo, and G. Wyvill. A mapping-independent primitive for the triangulation of parametric surfaces. *Graphical Models*, 65(5):260–273, 2003.
2. JD Boissonnat and S. Oudot. Provably good surface sampling and approximation. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 9–18. Eurographics Association Aire-la-Ville, Switzerland, 2003.
3. G. Bradshaw and C. O'Sullivan. Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics (TOG)*, 23(1):1–26, 2004.
4. J. Chhugani and S. Kumar. View-dependent adaptive tessellation of spline surfaces. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pages 59–62. ACM New York, NY, USA, 2001.
5. J. Chhugani and S. Kumar. Budget sampling of parametric surface patches. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics*, pages 131–138. ACM New York, NY, USA, 2003.
6. Tamal K. Dey and Joshua A. Levine. Delaunay meshing of isosurfaces. *Vis. Comput.*, 24(6):411–422, 2008.
7. T.K. Dey and J.A. Levine. Delaunay meshing of piecewise smooth complexes without expensive predicates. Technical report, Technical Report OSU-CISRC-7108-TR40, 2008.
8. Tim Foley and Jeremy Sugerman. Kd-tree acceleration structures for a gpu raytracer. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 15–22, 2005.
9. A. Greß, M. Guthe, and R. Klein. GPU-based collision detection for deformable parameterized surfaces. In *Computer Graphics Forum*, volume 25, pages 497–506. Blackwell Publishing, Inc, 2006.
10. Z. Guan, J. Shan, Y. Zheng, and Y. Gu. An extended advancing front technique for closed surfaces mesh generation. *Int. J. Numer. Meth. Engng*, 74:642–667, 2008.
11. Daniel Reiter Horn, Jeremy Sugerman, Mike Houston, and Pat Hanrahan. Interactive k-d tree gpu raytracing. In *I3D '07: Proceedings of the 2007 Symposium on Interactive 3D Graphics and games*, pages 167–174, 2007.
12. D.L. James and D.K. Pai. BD-tree: output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics (TOG)*, 23(3):393–398, 2004.
13. P. Jimenez, F. Thomas, and C. Torras. 3D collision detection: a survey. *Computers & Graphics*, 25(2):269–285, 2001.
14. P.M. Knupp. Algebraic mesh quality metrics for unstructured initial meshes. *Finite Elements in Analysis & Design*, 39(3):217–241, 2003.
15. S.M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.
16. M. Lin and D. Manocha. Collision and proximity queries. *Handbook of discrete and computational geometry*, 2, 2003.
17. H. Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Society for Industrial Mathematics, 1992.
18. F. Page and F. Guibault. Collision detection algorithm for NURBS surfaces in interactive applications. In *IEEE CCECE 2003. Canadian Conference on Electrical and Computer Engineering.*, volume 2, 2003.
19. JA Quinn, FC Langbein, RR Martin, and G. Elber. Density-Controlled Sampling of Parametric Surfaces Using Adaptive Space-Filling Curves. *LECTURE NOTES IN*

COMPUTER SCIENCE, 4077:465, 2006.

20. J. Schöberl. NETGEN An advancing front 2D/3D-mesh generator based on abstract rules. *Computing and Visualization in Science*, 1(1):41–52, 1997.
21. V. Surazhsky and C. Gotsman. Explicit surface remeshing. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 20–30. Eurographics Association Aire-la-Ville, Switzerland, 2003.
22. L. Velho, L.H. De Figueiredo, and J. Gomes. A unified approach for hierarchical adaptive tesselation of surfaces. *ACM Transactions on Graphics*, 18(4):329–360, 1999.
23. L. Velho, L.H. Figueiredo, and J. Gomes. A methodology for piecewise linear approximation of surfaces. *Journal of the Brazilian Computer Society*, 3, 1997.
24. V. Vlassopoulos. Adaptive polygonization of parametric surfaces. *The Visual Computer*, 6(5):291–298, 1990.
25. B. Von Herzen. *Applications of surface networks to sampling problems in computer graphics*. PhD thesis, Pasadena, CA, USA, 1989.