Ray Tracing Triangular Trimmed Free Form Surfaces

Wolfgang Stürzlinger[1]
Dept. of Computer Science, University of North Carolina at Chapel Hill
stuerzl@cs.unc.edu

Abstract

This paper presents a new approach to rendering triangular algebraic free form surfaces. A hierarchical subdivision of the surface with associated tight bounding volumes provides for quick identification of the surface regions likely to be hit by a ray. For each leaf of the hierarchy an approximation to the corresponding surface region is stored. The approximation is used to compute a good starting point for the iteration, which ensures rapid convergence.
Trimming curves are described by a tree of trimming primitives such as squares, circles, polygons and free form curves combined with Boolean operations. For trimmed surfaces an irregular adaptive subdivision is constructed to quickly eliminate all parts outside the trimming curve from consideration during rendering. Cost heuristics are introduced to optimize the rendering time further.

1. Introduction

Free form surfaces are used in computer aided design, CAD, to model curved objects such as automobiles, planes, and design parts. The accurate visualization of free form surfaces helps significantly in the design process and reduces the need to build physical models.
This work focuses on algebraic free form surfaces. For an introduction see e.g. [7]. Algebraic free form surfaces are rational polynomial surfaces in two parameters. Two types of free form surfaces exist: rectangular or tensor product surfaces and triangular surfaces. The definition of rectangular surfaces is conceptually more simple and they are used in many applications. Triangular surfaces are nowadays used more frequently as they are topologically more flexible than tensor product constructions. Surfaces based on a triangular domain are used in modeling complex geometries, filling of three- or multi-sided holes in boundary representations, blending of multiple surfaces, smoothing of polyhedra, free form deformations of solids, and for scattered data interpolation.
There are two main alternatives for creating images of free form surfaces: methods based on conversion to a polygonal model and ray tracing methods. The first approximates the free form surface with ‘small enough’ polygons and renders these into a z-buffer. Ray tracing of free form surfaces determines the visible parts by constructing rays from the viewpoint through the pixels of the image plane into the scene. The rays are intersected with the surfaces of the scene and the first surface hit determines the color of the pixel. The disadvantages of polygonalization are that visible shading artifacts occur if the subdivision is not very fine, that the subdivision process must handle the cracking problem, and that reflection and refraction cannot be simulated properly. Ray tracing is computationally more expensive than polygon display but all desired lighting effects can be simulated accurately.
Trimmed free form surfaces have become popular in practice because they can represent holes in surfaces and they can be used to define patches with arbitrary shape. Also, boolean operations on boundary representations of solids yield trimmed surfaces.

1.1. Previous Work

In general the intersection point of a ray with an algebraic free form surface cannot be calculated directly, as the degree of the corresponding equations is too high. Therefore an iterative approach is used. To obtain the intersection point two main strategies have been proposed: interval methods and hierarchical bounding structures followed by Newton iteration.
The first class progressively narrows the parameter interval of the intersection point and includes algorithms based on interval arithmetic [21] and more recently Bézier clipping [16]. The advantage of Bézier clipping is its guaranteed convergence. The algorithm can take trimming curves into account as well. A major disadvantage is the cost of the clipping operation, which grows with the cube of the degree of the surface. Also the concept is not straightforwardly applicable to triangular surfaces.

[1] Part of the research was done at GUP, Johannes Kepler Universität Linz/Austria.

The second class of methods encloses each part of a hierarchical subdivision of the surface with a bounding volume. In a preprocessing step a subdivision process splits the surface recursively and continues until the parts are flat enough. Intersecting a ray with the surface recursively traverses the bounding volume hierarchy and identifies all parts that may be hit. For each leaf reached Newton iteration is used to find the intersection point. Almost planar parts have the advantage that iterative methods converge quickly in such areas. Additionally bounding volumes are tighter. This improves the efficiency during the search for potentially intersecting surface regions as large parts of the tree can be pruned quickly.

Examples for previously used bounding volumes include axes-parallel bounding boxes [20], Chebyshev boxing [10], oriented slabs [22], and parallelepipeds [2]. The last method also employs an approximation to each surface part to obtain an accurate starting point for the Newton iteration, thereby ensuring rapid convergence with relatively small tree depths. Campagna [4] experimented also with combinations of enclosing volumes to optimize the performance.

The simplest approach for trimmed surfaces is to compute the intersection point first and then to test if the point is inside or outside the trimming curve. If the point is outside of the trimming curve no hit is reported and the time spent computing the intersection has been wasted. A different approach is to split all trimmed surfaces into multiple non-trimmed sub-surfaces [13]. But for complex trimming curves the method may create very many sub-surfaces.

1.2. Motivation and Overview

Many design packages allow the rendering of diffuse free form surfaces only. This ignores the reflections, which are quite important to the designer to judge the aesthetic 'qualities' of a surface. Ray tracing is better suited to the visualization of free form surfaces because it can simulate reflections accurately.

The new method presented here is an optimized ray tracing method for triangular free form surfaces[2]. In the presence of trimming curves the algorithm adapts the surface subdivision to their shapes. To support complex trimming curves the method depends on the flexibility to subdivide along arbitrary directions. Only triangular surfaces and the associated subdivision schemes support this. Therefore, this work concentrates on the ray tracing of triangular free form surfaces exclusively. This is a general solution, since all rectangular free form surfaces can be converted to two triangular free form surfaces [8].

In section 2 the definition of triangular free form surfaces and their most important properties are reviewed. A new and compact description of arbitrarily complex trimming curves is presented as well. Section 3 introduces a new method for ray tracing triangular free form surfaces and presents a new method that adapts the subdivision to the trimming curve. Results are discussed in section 4, and section 5 concludes.

2. Trimmed Free Form Surfaces

Free form surfaces are used to model curved objects such as automobiles that cannot be modeled precisely with conventional primitive objects such as polygons, polyhedra and spheres. This work focuses on the most common type, algebraic free form surfaces. For an introduction see e.g. [7]. Trimming curves limit the surface to parts of its parameter domain which is used e.g. to introduce holes.

2.1. Triangular Free Form Surfaces

The general definition of an algebraic free form surface is based on an array of three-dimensional control points $P_{i,j}$ that approximate the shape of the surface. The surface is parameterized by two parameters u and v. A weight function $B_{i,j}^n(u,v)$ describes the influence a control point $P_{i,j}$ has onto the shape of surface, where n is the degree of the surface. Relevant to this work are free form surfaces based on a triangular parameter domain. One powerful variant of triangular free form surfaces is the rational triangular Bézier surface. Rational surfaces allow the exact modeling of quadratic surfaces, which are common in mechanical design. See [6] for a survey. Triangular rational Bézier surfaces are defined as follows:

$$P(u,v) = \frac{\sum_{j=0}^{n} \sum_{i=0}^{n-j} P_{i,j} w_{i,j} B_{i,j}^{n}(u,v)}{\sum_{j=0}^{n} \sum_{i=0}^{n-j} w_{i,j} B_{i,j}^{n}(u,v)} \qquad \text{with} \qquad u,v \geq 0,\ u+v \leq 1 \tag{1}$$

with the corresponding Bézier weight functions:

$$B_{i,j}^{n}(u,v) = \frac{n!}{i!\,j!\,k!} u^i v^j w^k \qquad \text{with} \qquad w = 1-u-v,\ k = n-i-j \tag{2}$$

[2]To the author's knowledge no previous work has been published on ray tracing triangular free form surfaces.

For even better control of each point's influence on the surface each control point is assigned a weight $w_{i,j}$. In the following all weights $w_{i,j}$ are constrained to be positive. Then the following two properties hold: The surface lies completely inside the convex hull of its control points and the surface can be repeatedly subdivided into sub-surfaces of the same type. Triangular surfaces based on B-Splines have been presented recently [5]. See Fig. 1 for an example of a simple quadratic Bézier surface and it's control points.

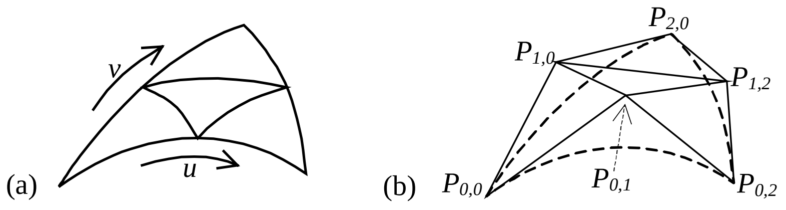

Fig. 1: (a) Bézier surface and (b) it's control point mesh.

For subdivision of two-dimensional Bézier curves the de Casteljau algorithm is used. The algorithm works by recursively applying linear interpolation with a fixed ratio to the control polygon. The final results are the control polygons for the two sub-curves and a point on the curve. For a more detailed description of the de Casteljau algorithm see e.g. [7]. Triangular free form surfaces can be subdivided with an extended form of the de Casteljau algorithm. The triangular control point mesh is recursively subdivided with tri-linear interpolation for a given pair of parameter values. The results are the control meshes for the three sub-surfaces and a point on the surface.
In Fig. 2(a) the geometric interpretation of the de Casteljau algorithm for a triangular surface (dashed) is illustrated. The light gray triangles represent the original control point mesh and progressively darker shades of gray represent subsequent recursions. The final result of the construction is the point of the surface corresponding to the chosen parameter values, the white point in the center triangle of Fig. 2(a). The vertices created by the de Casteljau algorithm form also the control point meshes of the three sub-surfaces. Fig. 2(b) depicts the control point mesh for the upper right sub-surface in (the shaded triangles).

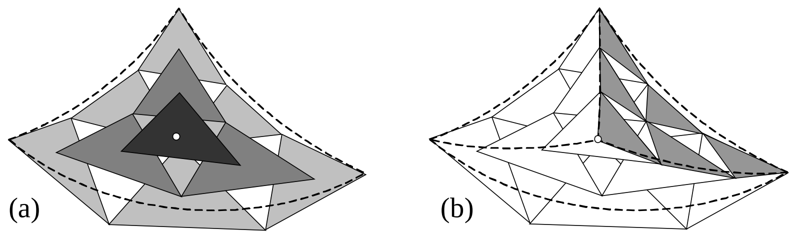

Fig. 2: (a) geometric interpretation of the de Casteljau algorithm for a triangular surface and (b) visualization of a control point mesh for the upper right sub-surface.

Applying the de Casteljau algorithm recursively to the resulting sub-surfaces leads to long thin triangles, see Fig. 3(a). Such triangles often cause numerical problems, which makes this subdivision algorithm impractical in this context.
A more general subdivision algorithm was presented by Goldman [11]. It is based on the following idea. Each of the three corner points $P_j(u_j,v_j)$, $j=1..3$ of an arbitrary triangular sub-surface can be computed by evaluating a sequence of n de Casteljau steps $C_j^i(u_j,v_j)$, $i=0..n$, where n is the degree of the surface. Choosing an arbitrary de Casteljau step from among the three $C_j^i(u_j,v_j)$ for each of the n steps of the sequence will construct a control point of the sub-triangle. Evaluating all possible permutations of this step sequence constructs all control points of the triangular sub-surface, see Fig. 3(b) and (c). This technique is now also known as the blossoming principle (see also [7] section 18.3).

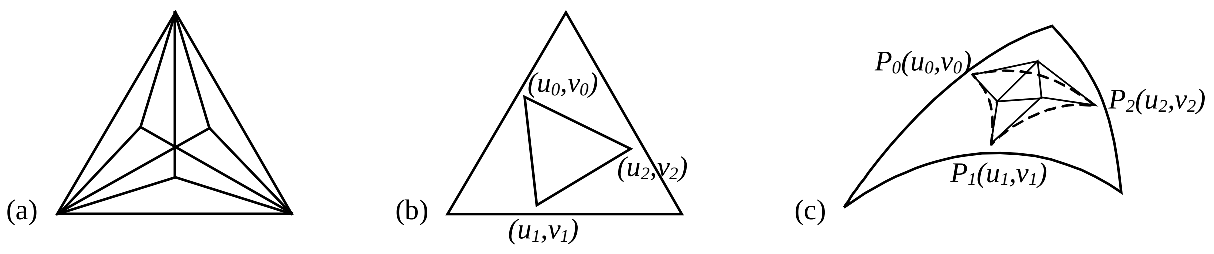

Fig. 3: (a) long thin triangles from repeated recursive subdivision with the de Casteljau algorithm, (b) arbitrary parameter domain sub-triangle and (c) corresponding sub-surface and its control point net.

The flexibility of the Goldman algorithm to subdivide the parameter domain in arbitrarily oriented triangles will be used in the following method for the adaptive subdivision of free form surfaces.

2.2. Trimming Curves

Trimming curves are used to limit the surface to parts of its parameter domain. They enable the user to define holes in surfaces and allow for arbitrary boundary shapes. Fig. 4 shows an untrimmed free form surface, a trimming curve in the parameter domain and the trimmed surface.

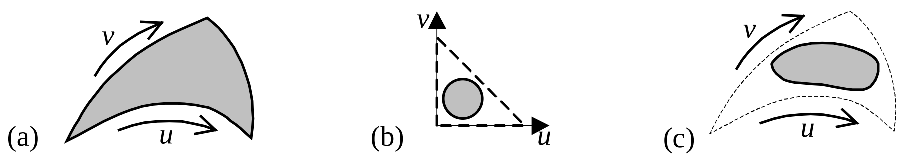

Fig. 4: (a) un-trimmed surface, (b) trimming curve in the parameter domain, and (c) trimmed surface.

In this work trimming curves are specified in a very general and compact way. Closed contour primitives are ‘composited’ in a tree using constructive planar geometry, CPG, operators analogous to the well-known three-dimensional CSG operators. The border contour of the resulting shape defines the trimmed region(s). These trimming curve(s) are clipped implicitly at the border of the parameter domain for simplicity.

One source of trimming curves is a solid modeling system that represents the surfaces of solids as free form surfaces. A basic operation in such a system is the intersection of two free form surfaces. The resulting three-dimensional intersection curve is of high degree in general. One alternative to find the two-dimensional trimming curves in the parameter domains of the two surfaces is a technique called inversion. A better possibility is to compute the two-dimensional trimming curves directly [15]. Implementations of both approaches represent the resulting trimming curve as a piecewise linear or cubic curve.

Furthermore, most solid modeling systems allow the combination of objects with CSG operators. A CSG combination of multiple objects necessitates a corresponding combination of trimmed regions on the surfaces. Instead of computing the trimmed regions directly, the final trimmed regions can be represented more efficiently as a set of trimming curves combined by a tree of CPG operator nodes. One important benefit of describing trimmed regions by a tree of CPG operators is that no intersection computation for trimming curves is needed.

In the following the CPG operator nodes, the contour primitives, and their corresponding inside tests are described. The inside test determines if a point in the parameter domain is inside or outside with respect to the trimming curve. Later on, during ray tracing (section 3), this inside test is used for each intersection point of a ray with the surface to determine if the intersection point is valid, i.e. inside the trimming curve.

Constructive Planar Geometry - CPG

Constructive planar geometry operators are used to combine contours. The operators combine the shapes of the two child nodes (the operands). The result is the border contour of the shape remaining after the operation. The operators include union, intersection, and difference. See Fig. 5 for simple examples.

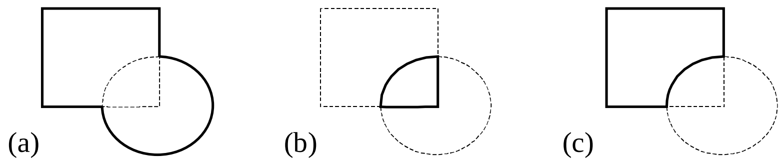

Fig. 5: CPG operators: (a) union OR, (b) intersection AND, and (c) difference SUB.

The inside test for an operator node recursively processes the two child nodes first. For the union operator the point has to lie inside at least one of the two children. The intersection operator tests if the point lies inside both children. For the difference operator the point has to be inside the left operand but not inside the right.

Contour Primitives

The following contour primitives are used as leaves in the CPG-tree: unit square, unit circle, closed polygon with straight or free form curve (i.e. Bézier, B-Spline or NURBS) segments. Examples are shown in Fig. 6. The usual affine transformation operators can be applied to all these primitives: translation, rotation, scale, and shear.

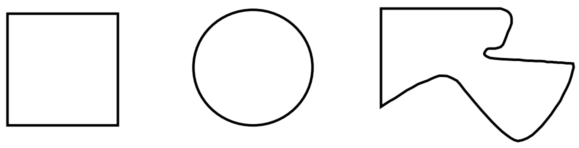

Fig. 6: Contour primitives: square, circle and closed polygon with straight and curved edges.

The inside test for the square and the circle is simple. For polygons a ray is traced from the query point in direction of the positive *x*-axis to infinity and the number of intersections with the polygon edges are counted. If the number is odd the point is considered inside [9]. For free form curves edge segments the ray-curve intersection test is performed by the two-dimensional equivalent of the free form surface ray casting procedure described by [2]. In two dimensions the method is very similar to strip trees [1]. If the number of edges, straight or curved, in a contour polygon is higher than an implementation dependent number, e.g. 8, another tree of parallelograms is built for the set of the edges. This optimizes the point-polygon test even for trimming polygons defined by a large number of segments, such as a piecewise cubic curve obtained from intersecting free form surfaces.
For transformed primitives the query point is first transformed back into the original coordinate frame of the primitive before performing the inside test.

3. Ray Tracing Trimmed Free Form Surfaces

The general idea is to construct a bounding volume hierarchy in a preprocessing step by subdividing each triangular surface successively into parts and enclosing the parts in appropriate volumes. Subdivision continues until the surface parts are almost planar or a given subdivision limit is reached. Tight bounding volumes are generated for each node of the resulting hierarchy. While all interior nodes of the bounding volume hierarchy store only a bounding volume the leaf nodes also store a planar approximation to the surface part. Control points for surface parts are only needed temporarily during the preprocessing stage and are not stored. This reduces memory usage significantly.
Depending on the shape of the trimming curve either a regular or an irregular subdivision pattern is used. Furthermore the bounding volume hierarchy is constructed only for surface parts which do not lie outside the trimming curve. During ray tracing the ray is checked recursively against this bounding volume hierarchy. If the ray does not reach a leaf it cannot intersect the surface in this region. In each leaf hit by the ray the intersection point is calculated by iterating on the original surface. For the leaves crossed by the trimming curve an inside test is performed to determine if the intersection point is valid.

3.1. Construction of the Bounding Volume Tree

In earlier works parallelepipeds were used to construct tight bounding volumes for quadrilateral free form surfaces [2]. Kajiya [13] used triangular prisms to optimize the ray tracing of fractals. Additionally recent work for collision detection shows that orientable solids result in more efficient bounding volume hierarchies compared to axis-aligned solids [12].

Tripipeds

As the bounding volume used in this work may also be skewed in the third direction the term *tripiped*, instead of skewed triangular prism, is introduced here to emphasize the similarity with the parallelepipeds (see Fig. 7).

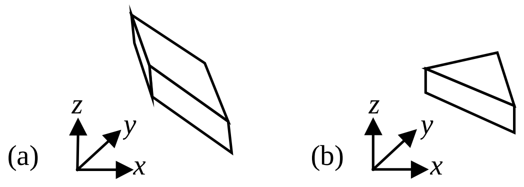

Fig. 7: (a) parallelepiped and (b) tripiped.

The algorithm splits the surface recursively until almost planar parts have been obtained or a maximum subdivision limit is reached. The planes delimiting the tripiped are constructed as follows. First the tangent vectors in the three directions *u*, *v*, *w* are approximated by computing the three difference vectors *du*, *dv*, *dw* of the corner points of the surface. The normal vector *N* obtained from the cross product of two tangent vectors defines the orientation of the bottom and top faces of the tripiped. The orientation of the three remaining planes is defined by the cross products of the difference vectors with the normal vector. The five planes are then moved outwards until the growing tripiped encloses all control points. The convex hull property of the triangular free form surfaces guarantees that all points of the surface part will be inside the constructed tripiped. Fig. 8(a) shows an example of a surface and it's enclosing tripiped.
The distance between the top and bottom planes, the 'thickness' of the tripiped, is the main criterion for controlling the subdivision depth. A 'thick' tripiped indicates that the surface might still be curved. A 'thin' tripiped signifies that the surface part is almost planar. Planar surface parts are desirable, as the Newton iteration used for intersection computations will converge much more rapidly in such regions.

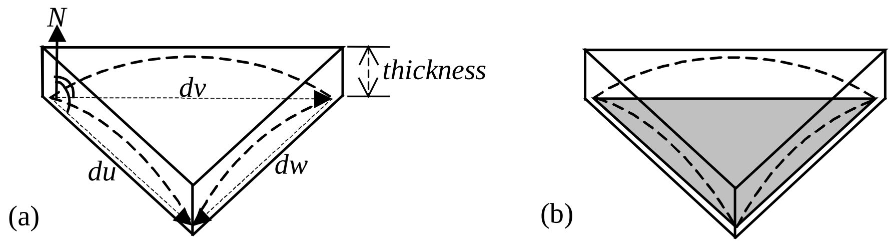

Fig. 8: (a) tripiped enclosing the surface and (b) planar approximation for surface.

The plane of the triangle defined by the three corner points is used as an approximation to the surface. It is depicted in Fig. 8(b). Note that this triangle is a first order approximation to the surface. The plane equation of this triangle is stored and used later on to compute the starting point for the Newton iteration.
The tripipeds in the coarser levels of the hierarchy enclose the control points of the corresponding subdivision of the surface. A consequence is that the tripipeds of coarser-level nodes provide only a very 'loose' fit for the free form surface in general. By shrinking each bounding volume to enclose the union of the children all bounding volumes are optimized in a second bottom-up pass. This optimization is illustrated in Fig. 9. The shaded region indicates the improvement.

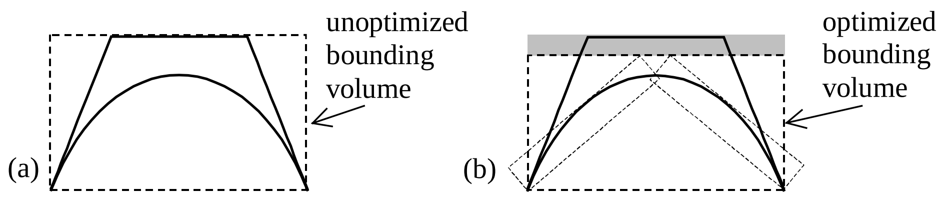

Fig. 9: (a) un-optimized and (b) optimized bounding volume.

For a multi-level hierarchy the benefits are larger in general, as the lowest level nodes enclose the surface very tightly. Consequently the optimized top-level bounding volume encloses, in general, the whole surface much tighter than the original bounding volume.

Subdivision for Contour Primitives

As discussed in section 2.1 a triangular free form surface can be subdivided in any triangular pattern by the Goldman algorithm. For untrimmed surface parts the domain is subdivided regularly into four equal parts, see Fig. 10. Compared to the results of the de Casteljau algorithm this pattern creates a more 'balanced' subdivision of the parameter domain.

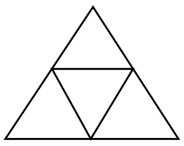

Fig. 10: Regular subdivision of triangular parameter domain.

Later on, when calculating ray-surface intersections each potential intersection point needs to be tested against the trimming curve to detect if it is inside the trimmed region. Depending on the complexity of the trimming curve this test may be a costly operation.
An optimization is to classify each region of the regular subdivision with respect to the trimming curve during preprocessing. Surface parts completely inside the trimming curve need no further inside test during ray tracing and are marked accordingly. Parts fully outside are pruned completely. Only for parts crossed by the trimming curve the inside test needs to be performed during ray tracing. See Fig. 11(a) for an illustration.
An even better result is obtained by utilizing the Goldman algorithm to subdivide along the trimming curve. If only one straight edge segment crosses the triangular parameter domain the subdivision algorithm splits along this edge. A curved segment is approximated by a secant. If too many edge segments cross the surface region or if the subdivision pattern yields very long thin triangles regular subdivision is used. A more detailed description of the algorithm is available in the Appendix. Pseudo code is shown in Fig. 12. This algorithm classifies much larger surface regions as fully inside or fully outside with respect to the trimming curve, compare Fig. 11(a) and (b).

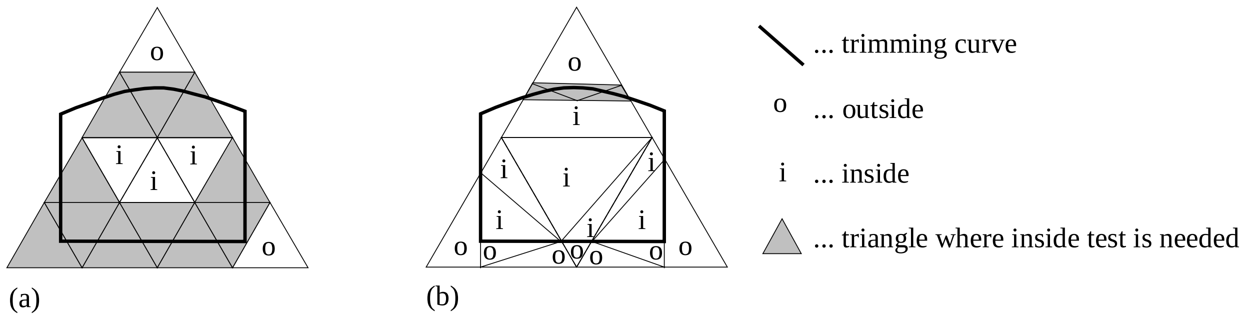

Fig. 11: (a) regular and (b) adaptive subdivision of the parameter domain with triangle classification.

```
Procedure Subdivide (Triangle T, Curve C)
      Find all segments of C crossing or inside T
      If subdivision limit is reached
            Store a reference to C with T
            Return
      If all segments of C are outside of T
            Return
      If one segment crosses or is inside T
            Create subdivision S of T with this edge
      Else-if two segments cross or are inside T and at least one is straight
            Choose one of the two to create subdivision S
      Else
            Create regular subdivision S
      If irregular subdivision S contains long, thin triangles
            Delete irregular subdivision S
            Create regular subdivision S
      Classify all triangles of S with respect to C
      For all triangles X which are not fully inside nor fully outside
            Subdivide (X, C)
```

Fig. 12: Pseudo-code for subdivision algorithm.

Note that all parts completely inside the trimming curve may still need further subdivision depending on the curvature of the surface (not shown in Fig. 12).

Subdivision for CPG trees

In general, it is not always necessary to process the whole CPG tree for the inside test of a surface part. Determining beforehand the nodes of the CPG tree, which can influence the result inside a given triangular domain, allows the construction of an optimized CPG tree and reduces evaluation times. In the simplest case an optimized tree is easily represented by a reference to a sub-tree of the original CPG tree. Fig. 13(a) shows the dashed triangular parameter domain, a trimming curve consisting of two parts and the corresponding CPG tree. In this case a reference to the right child suffices to characterize the trimming curve inside the triangular region. The general case can be handled by selectively copying operator nodes from the original tree, see Fig. 13(b).

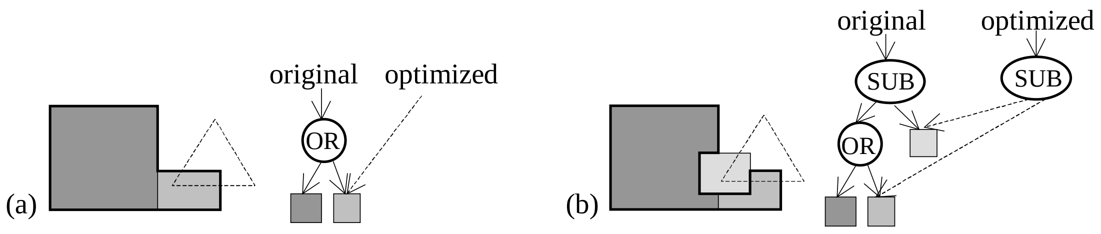

Fig. 13: (a), (b) Two examples for optimized CPG trees for inside tests (see text).

This optimization is applied during pre-processing to simplify the classifications against the trimming curve. Moreover, the reference to the optimized CPG tree is stored for each leaf crossed to limits the complexity of the inside test during ray tracing.

3.2. Cost Adaptive Construction of the Subdivision Tree

The subdivision method presented so far proceeds until each part of the free form surface corresponding to the parameter domain triangle is flat enough or a predefined limit is reached. If the subdivision process were continued further even more parts of the surface would need no inside test because more regions can be pre-classified. Furthermore, optimized subdivision leads to simpler trimming curves in the leaves and therefore yields faster inside tests, too. Clearly this is only beneficial if the cost for the descent in the subdivision tree is less than the cost of the inside test. In this context the cost for the inside test is defined as the average amount of work required for the test of a point with respect to the trimming curve.

To implement this strategy an estimate to the cost for the inside test is computed for each node of the CPG tree during the construction. For square and circle the cost for the test is assumed to be constant. The cost for polygons depends mainly on the number of edge segments and their type. Straight polygon segments have constant cost per segment. For curved segments modeled by Bézier-curves the cost depends mainly on the degree of the curve. B-Spline and NURBS segments are associated with cost depending on the order of the curve. The cost of a CPG-operator can be approximated using the cost of the two children. See Table 3 for an example of values used in the implementation.
During the subdivision phase a comparison of the cost of a recursive descent and the cost of the inside test associated with the trimming curve of the current triangle part allows to decide quickly if further subdivision is beneficial.

3.3. Ray-Surface Intersection

The intersection of a ray with the free form surface is performed with an adapted version of the iteration procedure presented by Barth and Stürzlinger [2]. This method is summarized in the following paragraphs and a few relevant details are discussed.
Each ray is intersected with the root tripiped of the bounding volume tree. If there is an intersection the successors of the node are examined recursively. For each leaf encountered the plane approximating the surface part is used to determine a good starting point for the Newton iteration, see Fig. 14(a). Iteration is performed on the original, whole surface. Each iteration step computes the tangent plane to the current point on the surface and approximates the true intersection point with the intersection of the ray with this plane. Iteration stops as soon as the change in parameter values falls below a certain limit. If the iteration has not converged after a pre-defined number of iteration steps, the algorithm assumes divergence. In this case the ray misses the surface part corresponding to the leaf of the subdivision tree and no hit is reported. Note that the ray may still converge to the (correct) intersection point in a neighboring part as the iteration is performed for each leaf hit by the ray. And as every point of the surface is enclosed by at least one tripiped no intersections will be missed. Previous experience with quadrilateral free form surfaces showed that a maximum of three iteration steps suffices for most rectangular surfaces [2].
The current approximation to the intersection point in the above iteration procedure may lie outside the parameter domain of a surface part associated with a leaf node. If the intersection point leaves the parameter domain while iterating by more than a predefined relative margin the point is re-projected onto the nearest border of the triangular domain and this point is used for the next iteration step. Whenever the iteration leaves the parameter domain more than once this indicates divergence. In this case the iteration is abandoned and no hit is reported for this leaf node.
Whenever the angle between the approximating plane and the ray is small there is a significant possibility that multiple intersection points with the surface exist. This is detected by computing the absolute value of the dot product of the ray direction and the plane normal. If it is smaller than a pre-defined threshold the ray intersects the tripiped at a grazing angle; i.e. the ray is almost parallel to the surface. In this situation the iteration may converge to the wrong intersection point. To guard against this the iteration is started twice using the projection of the two intersection points with the bounding tripiped onto the approximating plane. The intersection point nearer to the eye is returned. Fig. 14(b) shows a side view of a tripiped hit by a 'grazing' ray. The intersection points with the bounding tripiped are shown in gray and the projected points are shown in black.

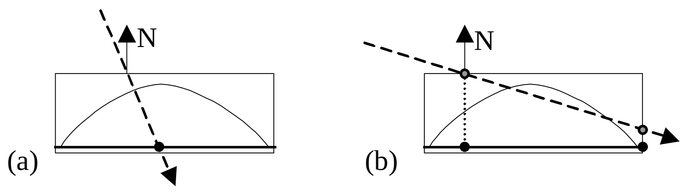

Fig. 14: Side view of the tripiped. (a) Starting point for normal ray and (b) starting points for ray at grazing angle (see text).

After an intersection point has been computed it may need to be tested against the trimming curve in the parameter domain. But with the presented method this happens only for a relatively small number of leaves in the subdivision tree. All surface parts entirely outside the trimmed region were culled during pre-processing. Everything entirely inside the trimming curve was marked as needing no inside test. Therefore the inside test is performed only for leaves whose parameter region is near a trimming curve.

4. Implementation and Results

The new rendering method for trimmed triangular free form surfaces has been implemented in the framework of the FLIRT ray tracer [19] in ANSI-C. All timings shown are in seconds for 512 by 512 images on a 250Mhz R4400 SGI-Onyx. One ray was shot through the midpoint of each pixel. The grazing angle criterion from section 3.3 was set to 78.5 degrees as in [2].

4.1. *Triangular Free Form Surfaces*

In Fig. 15(a) a cubic Bézier surface and in Fig. 15(b) the tripipeds corresponding to 2 recursive subdivisions are shown to illustrate the tight fit of the bounding volumes. A more complex model consisting of 288 triangular cubic Bézier surfaces is shown in Fig. 15(c) and a larger view of the upper part is depicted in Fig. 15(d). Fig. 15(e) shows a rational cubic Bézier surface. Another rational Bézier surface of degree 9 is shown in Fig. 15(f).

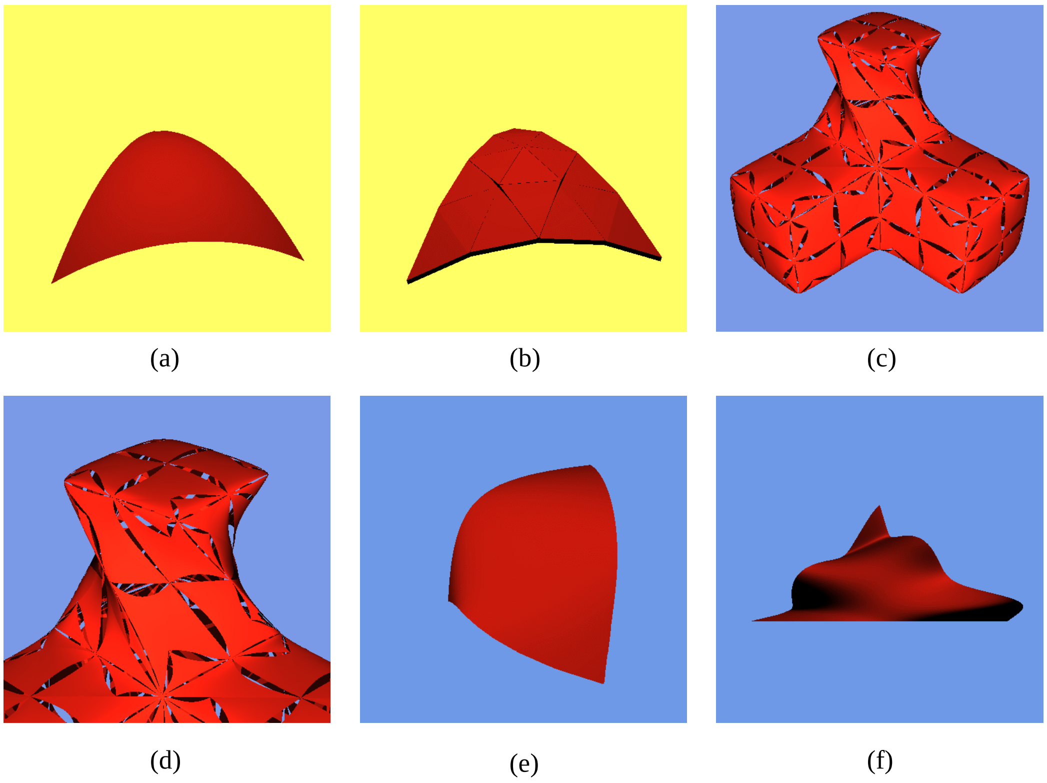

Fig. 15: Triangular free form surfaces.

To demonstrate the shape of the rational cubic surface more clearly a zoomed version with a reflective material is shown in Fig. 16(a). For comparison purposes the surface was approximated by subdividing into triangles up to the same maximum subdivision depth. Ray traced images of this polygonal approximation are shown in Fig. 16(b) without and in Fig. 16 (c) with linear normal vector interpolation. The last image is similar to a rendering of the surface with high-end graphics hardware and reflection mapping.

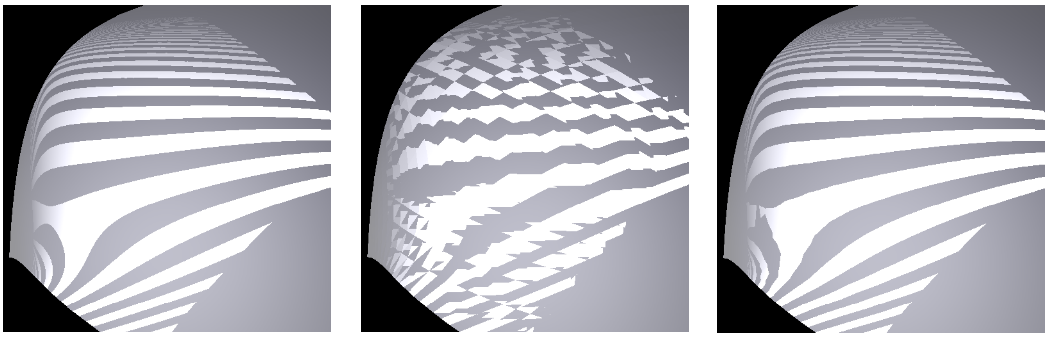
(a) (b) (c)

Fig. 16: Reflective free form surface (a) rendered with new ray tracing method and (b), (c) rendered from polygonal approximation, without and with normal vector interpolation.

The statistics for all above images are shown in Table 1. Setup time denotes the time spent in pre-processing for the creation of the subdivision tree and rendering time is the time spent ray tracing 512 by 512 images.

Fig.	Surface type	Avg. tree depth	Max. tree depth	# of tree leaves	Setup time	Memory in kb	Rendering time	Avg. # of iterations
15(a)	Cubic Bézier	3	3	64	0.01	23	12.47	2.33
15(b)	Tripipeds of 12(a)	2	2	16	0.01	6	10.87	N/A
15(c)	288 cubic Bézier	2.1	3	25	0.81	1233	86.64	2.79
15(d)	288 cubic Bézier	2.1	3	25	0.81	1233	105.1	2.79
15(e)	Cubic rat. Bézier	3.53	5	88	0.02	32	20.95	2.62
15(f)	Order 9 rat. Bézier	5.41	7	829	2.47	307	22.96	2.27
16(a)	Cubic rat. Bézier	3.86	5	88	0.28	33	94.43	2.56
16(b)	Tessellated 13(a)	5	5	1024	0.84	588	89.01	N/A
16(c)	Tessellated 13(a)	5	5	1024	0.88	588	90.33	N/A

Table 1: Rendering statistics for triangular surfaces (times in seconds).

The ray tracing times for Fig. 15(c), (d), and Fig. 16(b), (c) may not be optimal as a regular grid of dimension 8 was used for acceleration. The statistics show that the memory consumption of the method is moderate, only higher order surfaces consume significantly more memory. Furthermore, the results show that on average convergence can be achieved in fewer than three iterations. This is partly due to the fact that the tripipeds form tight bounding volumes for the surface. Even more important is the use of an accurate surface approximation to calculate the starting point for the iteration. The maximum number of iteration steps was limited to four in all tests. During ray tracing this limit was reached in only a few pixels in each image, mostly along the outline of the objects.

Compared to the polygonal approximations, the quality of the ray-traced images is superior. See e.g. the shadows cast into the inside of the object visible in Fig. 15(d) and the accurate reflections in the moderately curved surface in Fig. 16. The ray tracing based approach renders the reflection of the rectangular light source array in Fig. 16(a) accurately as the correct normal vector is computed at each point. The polygonal approximation in Fig. 16(b) is clearly not fine enough to cope with the detail. The same approximation rendered with linear normal vector interpolation in Fig. 16(c) gives a better result but some interpolation artifacts are visible and the topology of the reflections is not correct. The differences are most visible near the left corner of the surface. Further subdivision, optionally adaptive, or a better normal vector interpolation method, such as Phong-shading, may improve the image further but cannot guarantee a correct image for all viewpoints.

In general the subdivision depth needed to achieve rapid convergence depends on the surface curvature and the degree of the surface. Table 2 shows statistics for Fig. 15(f) to demonstrate the trade-off between subdivision depth and iteration count. All corresponding images are identical to Fig. 15(f).

Avg. tree depth	Max. tree depth	Thickness criterion	# of tree leaves	Setup time	Memory in kb	Rendering time	Avg. # of iterations	Max. # of iterations
5.41	7	0.05	829	2.47	307	22.96	2.27	4
6.45	8	0.025	3325	9.80	1201	19.49	1.87	3
7.46	9	0.0125	13426	39.40	4812	18.83	1.57	2
8.46	10	0.00625	53725	157.25	19216	20.12	1.51	2

Table 2: Rendering statistics for varying subdivision depths for Fig. 15(f).

A smaller 'thickness' criterion for the tripipeds results in more planar leaf nodes, which results in faster convergence of the Newton iteration. But at a certain subdivision level the cost for the traversal of the hierarchy outweighs the gains and the rendering time increases again, compare the data for the last two rows. Note also the increase in memory consumption and pre-processing time.

4.2. Trimmed Free Form Surfaces

For trimmed surfaces Table 3 summarizes the cost metrics for the various inside tests and operations. The values were estimated by measuring the average time an inside test takes for the primitive. For tests with effort depending on the characteristics of the primitive, e.g. the order of B-Spline curves, several values were measured and the measurements were fitted with the equations in the table. These cost values are implementation dependent and vary from machine to machine. Note also that the numbers given in the table do not directly correspond to the timing results. Only the relative magnitudes are relevant, therefore the table entries were transformed to more convenient integer values.

Object	Estimated average cost for inside test
Square	270
Circle	390
Polygon	$300 + \Sigma_{i=1..\#seg}\ Cost(segment_i)$
Straight polygon segment	45
Curved polygon segment	*#iterations * Costs(curve)*
Bézier	6950 + 70 *n*
B-Spline	$11070 + 150\, k^2$
NURBS	$11635 + 310\, k^2$
CPG-operator	510 + *Costs(left_tree)* + 0.5 *Costs(right_tree)*
Recursive descent in tree	2360

Table 3: Estimated average cost for inside test (see text).

In the table *#seg* is the number of edges of the polygon, *#iterations* is the estimated average number of Newton iteration steps (two for strip trees in our implementation) and *n* and *k* are the number of control points respective the order of the free form trimming curve. The cost for the CPG operator is calculated from the cost of the left child and half the cost of the right child as it can be assumed that in half the cases the evaluation of the left sub-tree suffices to determine the result.
In Fig. 17(a) a circle has been cut out from a triangular free form surfaces. A simple CPG-tree was used to create the shape in Fig. 17(b). Another CPG-tree was used to create Fig. 17(c). This image shows the power of CPG-trees, as only a few minutes were necessary to construct the shapes using 10 squares and 8 circles and 17 CPG-operator nodes. In Fig. 17(d) twelve NURBS curves were used to generate the outer contour and another was used to cut out the interior using a CPG-operator. To emphasize the qualities of the constructed subdivision the surface regions where an inside test is necessary are depicted in Fig. 17(e). A coarser subdivision was used to generate the image; otherwise the structures are almost too thin to be visible.

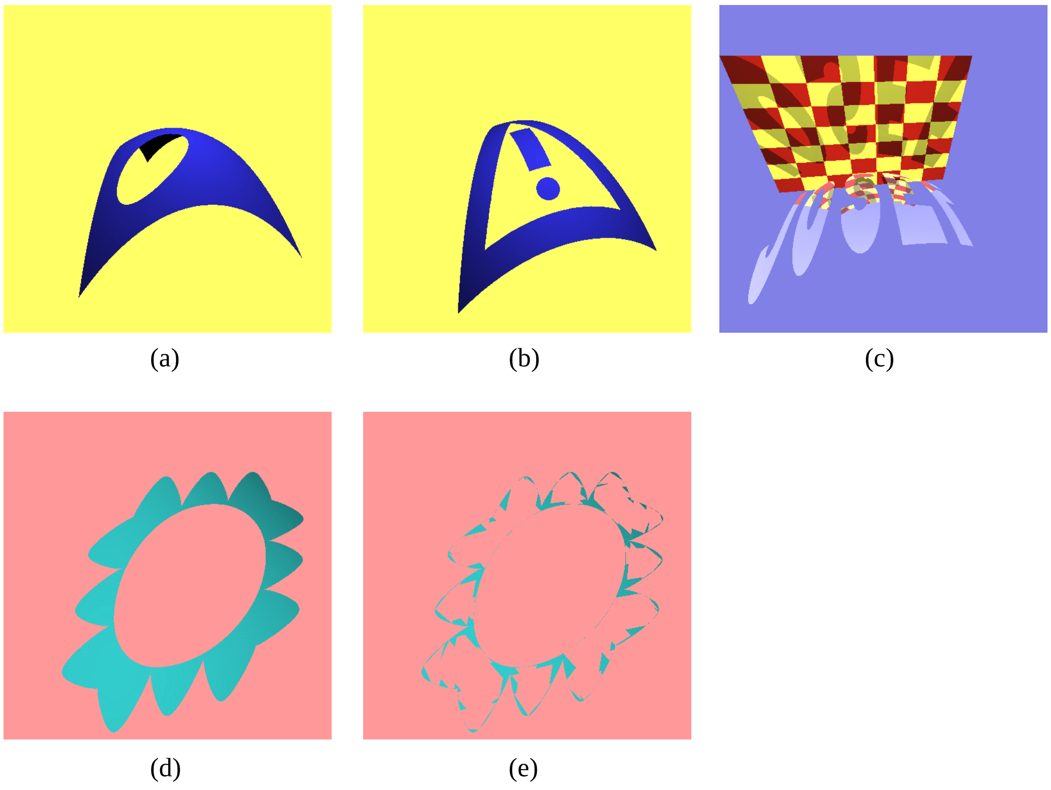

Fig. 17: Trimmed free form surfaces (see text).

The statistics are shown in Table 4. No optimization signifies that the inside test was performed on the whole surface. The tree optimization means that the tripiped tree has been built only for parts that are not outside the trimming curve and the cost optimization used the cost adaptive scheme presented in section 3.2. The second set of entries for Fig. 17(d) was produced using a tessellation of each NURBS trimming curve into 64 segments. The rendered image is indistinguishable from Fig. 17(d).

Fig.	Optimization	Trimming curve type	Avg. Tree depth	Max. tree depth	# of tree leaves	Setup time	Memory in kb	Rendering time	Avg. # of iterations
17(a)	No	Circle	3.86	4	184	0.03	65	11.46	2.37
17(a)	Tree	Circle	4.62	6	318	0.05	116	11.14	2.38
17(a)	Cost	Circle	4.4	5	258	0.05	94	10.75	2.38
17(b)	No	CPG-tree	3.86	4	184	0.03	65	12.62	2.23
17(b)	Tree	CPG-tree	5.35	12	386	0.11	156	11.15	2.17
17(b)	Cost	CPG-tree	4.88	12	252	0.06	94	10.98	2.18
17(c)	No	CPG-tree	6.38	7	4054	2.47	1426	45.64	1.56
17(c)	Tree	CPG-tree	6.83	7	1776	1.47	697	33.65	1.73
17(c)	Cost	CPG-tree	6.72	7	1505	1.18	583	32.14	1.74
17(d)	No	13 NURBS	3	3	64	0.15	289	57.01	2.16
17(d)	Tree	13 NURBS	6.52	8	1024	2.33	679	19.15	1.76
17(d)	Cost	13 NURBS	6.52	8	1024	2.37	679	18.38	1.76
17(d)	No	tessellation	3	3	64	0.11	198	54.65	2.16
17(d)	Tree	tessellation	6.8	7	1429	1.83	450	19.09	1.47
17(d)	Cost	tessellation	6.8	7	1429	1.9	450	18.42	1.47
17(e)	N/A	13 NURBS	5.95	6	409	1.43	602	14.63	N/A

Table 4: Rendering statistics for trimmed surfaces (times in seconds).

The results show that a large speedup is due to the optimized construction of the tripiped tree. Even for the simplest case, a circle, a speedup is possible. For more complex trimming curves the speedup increases proportionally. Of course the achievable speed-up depends on the ratio between inside and outside area with respect to the trimming curve. If the area inside the trimming curve is small then better speed-ups are to be expected.
The cost heuristic introduced in section 3.2 provides only small benefit. One reason is that the tripiped tree is already a very good optimization for the inside test, and the cost heuristic has little 'room' to work in.
No comparison with other methods was performed because no other approaches for ray tracing triangular free form surfaces are known. Although decomposing a quadrilateral surface into two triangular surfaces is possible, it involves a degree elevation for the triangular surfaces, which precludes a fair comparison!

5. Conclusions and Future Work

This work presented a new method for an optimized rendering method for triangular free form surfaces. The technique is based on a hierarchical subdivision of the free form surface. Tripipeds, triangular skewed prisms, are used as tight bounding volumes for the hierarchy. Each leaf stores an approximation for the corresponding part of the surface. This approximation ensures that the Newton iteration converges rapidly to the intersection point.
Results demonstrate that the method is able to compute an image in competitive times compared to a polygonal approximation. It was shown that the image quality is superior as all lighting effects can be depicted accurately. The memory usage of the method is moderate and pre-processing times are small.
For trimmed surfaces a compact description of the trimmed regions using constructive planar geometry, CPG, was presented. The algorithm adapts the hierarchical subdivision pattern to the shape of the trimming curve(s). This allows predetermining the outcome of the inside test for almost all parts of the surface, thereby providing significant speed-up as unnecessary computations are avoided. Cost heuristics were introduced to improve the results even further, although they achieved only a minor speed-up in our tests.
It is straightforward to apply the presented adaptive subdivision method to rendering polygonal free form surface approximations. Note that in this case no inside test can be performed at run-time, therefore a finer subdivision will be necessary in general.
Future work includes methods to compute the subdivision of the surface on demand, e.g. as in [18]. This will reduce the memory usage even further and is needed for models with a large number of free form surfaces. Also the extension of the presented techniques to quadrilateral free form surfaces is considered. Evaluation of a new iteration technique [17] is also planned.

Acknowledgements

Thanks to J. Bramsteidl for implementing parts of the software and to W. Barth, J. Volkert, D. Manocha, T. Whitted, T. Hudson, and G. Meenakshi for helpful comments on previous versions of this manuscript. Many thanks to the anonymous reviewers for their constructive suggestions. Thanks also to Jörg Peters for the 288-patch model. The author acknowledges support from the Austrian Science Foundation (FWF).

References

[1] D. H. Ballard, *Strip Trees: A Hierarchical Representation for Curves*, Communications of the ACM 24(5), pp. 310-321, 1981.
[2] W. Barth, W. Stürzlinger, *Efficient Ray tracing for Bézier and B-Spline Surfaces*, Computers & Graphics 17(4), pp. 423-430, July 1993.
[3] J. Bramsteidl, *Ray tracing von Getrimmten Freiformflächen*, Master thesis, Univ. of Linz, Austria, March 1996.
[4] S. Campagna, Vergleich und Erweiterung von Verfahren für die Schnittpunktberechnung von Strahlen mit Polynomflächen, Master thesis, Univ. of Erlangen, Germany, August 1995.
[5] W. Dahmen, C. A. Micchelli, H. -P. Seidel, *Blossoming begets B-Splines built better by B-Patches*, Math. Comp., vol. 59, pp. 97-115, 1992.
[6] G. Farin, *Triangular Bernstein-Bézier patches*, Computer Aided Geometric Design 3(2), pp. 83-128, 1986.
[7] G. E. Farin, *Curves and Surfaces for Computer Aided Geometric Design*, 4th edition, Academic Press, New York, 1997.
[8] D. J. Filip, R. N. Goldman, *Conversion from Bézier-rectangles to Bézier-triangles*, Computer Aided Design, vol. 19, pp. 25-27, 1987.
[9] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, *Computer Graphics: Principles and Practice*, second edition, Addison-Wesley, 1990.
[10] A. Fournier, J. Buchanan. *Chebyshev Polynomials for Boxing and Intersections of Parametric Curves and Surfaces*, Computer Graphics Forum 13(3) (Proceedings EUROGRAPHICS '94), pp. 127-142, September 1994.
[11] R. N. Goldman, *Subdivision Algorithms for Bézier Triangles*, Computer Aided Design, vol. 15, pp. 159-166, 1983.

[12] S. Gottschalk, M. C. Lin, D. Manocha, *OBBTree: A Hierarchical Structure for Rapid Interference Detection*, Computer Graphics (Proceedings SIGGRAPH '96), pp. 171-180, August 1996.
[13] B. Hamann, P. -Y. Tsai, *A Tessellation Algorithm for the Representation of Trimmed NURBS Surfaces with Arbitrary Trimming Curves*, Computer-Aided Design, vol. 28, pp. 461-472, 1996.
[14] J. T. Kajiya, *New Techniques for Ray Tracing Procedurally Defined Objects*, Computer Graphics 17(3) (Proceedings SIGGRAPH '83), pp. 91-102, July 1983.
[15] S. Krishnan, D. Manocha, *An Efficient Surface Intersection Algorithm based on Lower Dimensional Formulation*, ACM Trans. On Computer Graphics, 16(1), pp. 74-106, 1997.
[16] T. Nishita, T. W. Sederberg, M. Kakimoto, *Ray Tracing Trimmed Rational Surface Patches*, Computer Graphics (Proceedings SIGGRAPH '90), vol. 24, pp. 337-345, August 1990.
[17] K. Qin, M. Gong, Y. Guan, W. Wang, *A New Method for Speeding Up Ray Tracing NURBS Surfaces*, Computers & Graphics 21(5) pp. 577-586, 1997.
[18] S. M. Rubin, T. Whitted, *A 3-Dimensional Representation for Fast Rendering of Complex Scenes*, Computer Graphics 14(3) (Proceedings SIGGRAPH '80), pp. 110-116, July 1980.
[19] W. Stürzlinger, R. F. Tobler, M. Schindler, *FLIRT - Faster than Light Ray Tracer*, Technical Report, Institut für Computergraphik, Technical University of Vienna, Aug. 1993.
[20] M. Sweeney, R. Bartels, *Ray Tracing Free-Form B-Spline Surfaces*, IEEE Computer Graphics & Applications 6(2), pp. 41-49, 1986.
[21] D. Toth, *On Ray Tracing Parametric Surfaces*, Computer Graphics (Proceedings SIGGRAPH '85), vol. 19, pp. 171-179, July 1985.
[22] J. Yen, S. Spach, M. Smith, R. Pulleyblank, *Parallel Boxing in B-Spline Intersection*, IEEE Computer Graphics & Applications 11(1), pp. 72-79, Jan. 1991.

Appendix

The following rules are used to adapt the subdivision pattern to the trimming curve: The simplest case is that one straight edge segment of the trimming curve crosses the triangular parameter domain. If the subdivision pattern incorporates this edge directly no further inside tests are necessary in this region of the surface. Fig. 11(b) is an example where the entire triangular domain has been subdivided *regularly* once (as there are many segments crossing the triangle), and subsequently the center, top-down triangle has been subdivided along the straight crossing edge.

For curved edge segments of the trimming curve the subdivision pattern can only approximate the shape of the curve. The freedom to subdivide in any direction is still exploited to limit the region needing an inside test to a small portion of the whole surface. First all intersection points of the curve with the triangle borders are computed. If there are more than two the triangle is subdivided regularly. Otherwise, the secant is constructed, which gives one edge of the subdivision pattern. A second edge is obtained by computing the curve point corresponding to the center value of the parameter interval and constructing a tangent parallel to the secant. The two edges define a trapezoidal region that approximates an enclosure for the trimming curve. If the distance between secant and tangent is small compared to the size of the triangle only the secant is used to subdivide the triangle. Otherwise, the trapezoid is split into three triangles using the midpoint of the longer of the two edges. The upper part of Fig. 11(b) shows an example for a curved segment. The secant to the curved edge segment has to be moved a short distance, relative to the triangle edge length, away from the curve to avoid problems with numerical instabilities. In Fig. 11(b) this has been exaggerated to illustrate the principle.

For the general case of multiple edge segments of the trimming curve crossing or inside a triangle, the following heuristic rules are used:

1. Two straight edges: For each edge the maximum of the deviations of the two intersection points from the middle point of the corresponding triangle edge is computed. The edge with the smaller value is chosen to achieve a more balanced subdivision pattern. See the lower left and right part of Fig. 11(b). If an edge intersects a triangle vertex this edge is used to subdivide the triangle into two parts.
2. A straight edge and a curved edge: For simplicity the subdivision proceeds along the straight edge.
3. For all other configurations regular subdivision into four triangles is performed.

For all cases discussed above the triangles of the resulting subdivision pattern are examined to avoid very long thin triangles, which may cause numerical problems. If the aspect ratio of a triangle of the subdivision pattern falls below a pre-defined threshold regular subdivision is used instead of irregular subdivision.

Once subdivision reaches a pre-defined limit a leaf is marked as needing an inside test and a reference to the corresponding trimming curve is stored. Only such leaf nodes need inside tests during ray tracing, all other leaves are classified by the algorithm to be either completely inside or completely outside the trimming curve.