Ray Tracing of Subdivision Surfaces

Leif P. Kobbelt   K. Daubert   H-P. Seidel
Computer Science Department, University of Erlangen-Nürnberg
Am Weichselgarten 9, 91058 Erlangen, Germany

Abstract. We present the necessary theory for the integration of subdivision surfaces into general purpose rendering systems. The most important functionality that has to be provided via an abstract geometry interface are the computation of surface points and normals as well as the ray intersection test. We demonstrate how to derive the corresponding formulas and how to construct tight bounding volumes for subdivision surfaces. We introduce envelope meshes which have the same topology as the control meshes but tightly circumscribe the limit surface. An efficient and simple algorithm is presented to trace a ray recursively through the forest of triangles emerging from adaptive refinement of an envelope mesh.

1 Introduction

The general concept of subdivision techniques for the construction and representation of free-form surfaces is gaining more and more attention in computer graphics and related fields [19, 22]. The efficiency of subdivision algorithms and the flexibility with respect to the topology and connectivity of the control meshes makes this approach suitable for many applications such as surface reconstruction [2, 4] and interactive modeling [23]. The close connection to multi-resolution analysis of parametric surfaces provides access to the combination of classical modeling paradigms with hierarchical representations of geometric shape.

So far subdivision surfaces have been used mainly in the context of geometric modeling, i.e., issues like the asymptotic behavior of the scheme [6, 24] and the (discrete) fairness of the resulting meshes were investigated [11, 12]. Meanwhile the related mathematical theory has reached a state of maturity which allows the programmer to choose among the many schemes proposed in the literature [2, 5, 10, 13]. In this paper we do not investigate such properties but we address an important issue for the integration of subdivision schemes into a wider range of potential applications: While the subdivision methods have become a standard in *surface design*, the connection to *rendering applications* is still based on raw triangle data exchange.

There has been a considerable amount of work on the integration of higher order basic shapes like spline surfaces into the generic setup of sophisticated rendering algorithms [1, 7, 9, 15, 17, 20, 21]. Most of the approaches derive a more or less tight, preferably convex, bounding volume for each patch. The size of this bounding volume provides an upper bound on the spatial extent of the object such that ray intersection tests can be implemented much more efficiently by discarding rays according to simple tests against the bounding volume. If the bounding volume aligns to the local geometry of a patch then its shape can be used as an oracle to rate the local flatness, i.e., the approximation error if the true geometry would be replaced by a planar face.

In this paper we will present the basic prerequisites which are necessary to transfer the generic bounding volume technique to subdivision surfaces. The mathematical difficulties emerge from the fact that in general there is no explicit description for the limit surface and hence possible bounds have to be derived from the coefficients of the underlying refinement equation (i.e., from the coefficients of the subdivision masks).

We start by finding points and normal vectors on the limit surface corresponding to the initial control vertices. The triangles of the initial control mesh imply a decomposition of the limit surface into triangular patches with these limit points at their corners. For each patch we compute a bounding prism by sweeping the chord triangle spanned by the three corners in the (triangle-) normal direction.

Based on the individual bounding prisms, we build envelope meshes for the composite surface by moving the limit points in (point-) normal direction until the bounding prisms are completely contained. This provides a continuous polyhedral hull. When a given ray intersects one of the envelope triangles, we perform local subdivision to obtain a better piecewise linear approximation of the limit surface. Since the neighboring triangles in a mesh data structure can be found in $O(1)$, we can formulate an efficient recursive scheme that traces the ray through the hierarchy of envelope triangles. The recursion stops when a local flatness criterion is met, indicating that the intersection with the chord triangle does not deviate from the true solution by more than a prescribed ε. The algorithm has been integrated as a new geometric primitive into a general purpose ray tracing tool to compute the pictures shown in the result section.

Throughout the paper we will explain the theoretic concepts and general methods in the context of *univariate* subdivision. Transferring the results to the bivariate setting is rather obvious but depending on the vertices' valences several special cases have to be considered. To make the reproduction of the results as easy as possible, we apply the corresponding formulas explicitly to Loop's subdivision scheme [13].

2 Limit points and normals for subdivision surfaces

Let a control polygon $P_0 = [\mathbf{p}_i^0]$ with $\mathbf{p}_i^0 \in \mathbb{R}^3$ be given which represents the curve

$$P(t) \;=\; \sum_i \mathbf{p}_i^0\, \phi(t-i) \qquad \text{with} \qquad \phi(t) \;=\; \sum_j \alpha_j\, \phi(2t-j). \tag{1}$$

It is well known that the subdivision rule

$$\mathbf{p}_i^1 \;:=\; \sum_j \alpha_{i-2j}\, \mathbf{p}_j^0 \tag{2}$$

generates a new control polygon $P_1 = [\mathbf{p}_i^1]$ such that

$$P(t) \;=\; \sum_i \mathbf{p}_i^1\, \phi(2t-i).$$

provides a *refined* representation of the same curve. Notice that (2) actually combines *two* rules triggered by the parity of i. By iterating the refinement rule we obtain a sequence of polygons $P_m = [\mathbf{p}_i^m]$, $m = 0, 1, \ldots$ which — depending on the coefficients α_j — converges to a smooth limit curve P_∞.

We are interested in computing points on the limit curve P_∞ directly from the control points $\mathbf{p}_i^m$ on some level m without going through the iterative refinement. The following technique has become standard in the analysis of subdivision schemes: Construct a local subdivision matrix and transform it into its basis of (generalized) eigenvectors.

Let the coefficients $[\alpha_j]_{j=0}^{2n}$ define a univariate subdivision scheme. If we rewrite the polygon P_m as a *vector* then the subdivision step $P_m \to P_{m+1}$ corresponds to the multiplication of P_m by a suitable matrix

$$\widetilde{S} \;=\; \left(\begin{array}{ccccccccc}
\ddots & & \ddots & & \ddots & & & & \\
 & \alpha_1 & \ldots & \alpha_{2j+1} & \ldots & \alpha_{2n-1} & 0 & 0 & \\
 & \alpha_0 & \ldots & \alpha_{2j} & \ldots & \ldots & \alpha_{2n} & 0 & \\
 & 0 & \alpha_1 & \ldots & \alpha_{2j+1} & \ldots & \alpha_{2n-1} & 0 & \\
 & 0 & \alpha_0 & \ldots & \alpha_{2j} & \ldots & \ldots & \alpha_{2n} & \\
 & & & \ddots & & \ddots & & & \ddots
\end{array}\right).$$

Due to the fixed bandwidth of $\widetilde{S}$ we observe that the non-zero coefficients of $2n-1$ successive rows form a square matrix S. From an algorithmic point of view this means that the $(n-1)$-neighborhood of $\mathbf{p}_{2i}^{m+1}$ in P_{m+1} is completely determined by the $(n-1)$-neighborhood of $\mathbf{p}_i^m$ in P_m.

Since the control vertices $\mathbf{p}_i^m$ represent the function $P(t)$ with respect to the functional basis $\phi(2^m t - i)$ it is natural to associate the vertex $\mathbf{p}_i^m$ with the parameter value $t_i^m = i\,2^{-m}$. Hence, through all subdivision levels, the vertices $\mathbf{p}_{i2^r}^{m+r}$ correspond to the same parameter value and this sequence of vertices converges for $r \to \infty$ to the point $P(t_i^m)$ on the limit curve.

The sub-polygons of $2n-1$ successive vertices $\mathbf{p}_i^m$ and $\mathbf{p}_i^{m+1}$ correspond to the nested parameter intervals $t_i^m + [1-n, n-1]\,2^{-m}$ and $t_i^m + [1-n, n-1]\,2^{-(m+1)}$ respectively and since the same subdivision mask S is applied in every step, it is obvious that the limit point $P(t_i^m)$ is determined by the $(n-1)$-neighborhood of the vertex $\mathbf{p}_i^m$ through

$$P(t_i^m)\,[1,\ldots,1]^T \;=\; \lim_{r\to\infty} S^r\;[\mathbf{p}_{i-n+1}^m,\ldots,\mathbf{p}_{i+n-1}^m]^T.$$

The direct computation of $P(t_i^m)$ requires the decomposition of $S = V^{-1}DV$ into a diagonal matrix D and a transform V into the basis of eigenvectors[1]. The convergence of the iterative scheme implies that the dominant eigenvalue of S be $\lambda_1 = 1$ and the affine invariance of the subdivision operator indicates that the corresponding eigenvector is $[1,\ldots,1]$. Therefore the coefficients $l_{1-n},\ldots,l_{n-1}$ such that

$$P(t_i^m) = \sum_j l_j\,\mathbf{p}_{i+j}^m$$

can be read off that row of V which is associated with this eigenvector. This is obvious since components of the input vector which belong to eigenspaces of smaller eigenvalues fade out during the iteration of S.

If the subdivision scheme generates C^1 curves then there exists a *difference scheme* $\widetilde{S}'$ which maps the divided differences $\triangle\mathbf{p}_i^m = 2^m(\mathbf{p}_{i+1}^m - \mathbf{p}_i^m)$ of P_m to the divided differences of P_{m+1} [6]. For repeated subdivision the differences converge to the derivative of the limit function $P'(t)$. The limit point analysis applied to the difference scheme hence provides limit tangent vectors.

Due to the simple relation between $\widetilde{S}$ and $\widetilde{S}'$ it turns out that the eigenvector $[1,\ldots,1]$ of S' with eigenvalue $\lambda_1 = 1$ corresponds to the eigenvector $[1-n,\ldots,n-1]$ of S with eigenvalue $\lambda_2 = \frac{1}{2}$ (*constant differences*). Hence, just as the eigenvector for the dominant eigenvalue $\lambda_1 = 1$ allows us to compute the limit point, the eigenvector for the subdominant eigenvalue $\lambda_2 = \frac{1}{2}$ determines the limit tangent at $P(t_i^m)$ since it describes the line which is asymptotically approached by the sequence:

$$P'(t_i^m)\,[1-n,\ldots,n-1]^T \;=\; \lim_{r\to\infty} 2^r\,S^r\;\left[\,\mathbf{p}_{i-n+1}^m - P(t_i^m),\ldots,\mathbf{p}_{i+n-1}^m - P(t_i^m)\,\right].$$

If the subdivision scheme converges to a C^1 limit then the modulus of all other eigenvalues $\lambda_3 \geq \ldots \geq \lambda_{2n-1}$ is less than $\frac{1}{2}$. The rate by which the deviation of the sub-polygon $[\mathbf{p}_{i-n+1}^m,\ldots,\mathbf{p}_{i+n-1}^m]$ from a straight line fades out is $|\lambda_3|$. As expected (by Taylor's theorem) the local flattening rate $1/\lambda_3$ is higher than the contraction rate $1/\lambda_2 = 2$ if the limit curve is smooth.

In the bivariate setting, we have two partial derivatives of first order and accordingly the local subdivision matrix S of a refinement scheme which generates C^1 limit surfaces

[1]In general D is the Jordan normal form of S but for convergent subdivision schemes with smooth limit surfaces, the leading eigenvalues have algebraic multiplicity one [16, 24].

has a double subdominant eigenvalue $\lambda_2 = \lambda_3$. The components of the input mesh which lie in the corresponding eigenspaces span the tangent plane at the limit point.
Example: Consider the three directional grid spanned by $(1,0)$, $(0,1)$, and $(1,1)$. The quartic box spline M_{222} defined on this grid satisfies the refinement equation [3]

$$M_{222}(u,v) = \sum_{i,j=0}^{4} \alpha_{i,j}\, M_{222}(2u-i, 2v-j) \quad \text{with} \quad [\alpha_{i,j}] = \frac{1}{16}\begin{pmatrix} 0 & 0 & 1 & 2 & 1 \\ 0 & 2 & 6 & 6 & 2 \\ 1 & 6 & 10 & 6 & 1 \\ 2 & 6 & 6 & 2 & 0 \\ 1 & 2 & 1 & 0 & 0 \end{pmatrix}$$

Hence, the corresponding subdivision rules are

$$\mathbf{p}_{i,j}^{m+1} := \begin{cases} \frac{1}{16}(10\mathbf{p}_{l,k}^m + \mathbf{p}_{l,k-1}^m + \mathbf{p}_{l,k+1}^m + \mathbf{p}_{l-1,k}^m + \mathbf{p}_{l+1,k}^m + \mathbf{p}_{l-1,k-1}^m + \mathbf{p}_{l+1,k+1}^m) & i=2l, \quad j=2k \\ \frac{1}{16}(2\,\mathbf{p}_{l,k-1}^m + 6\,\mathbf{p}_{l,k}^m + 6\,\mathbf{p}_{l+1,k}^m + 2\,\mathbf{p}_{l+1,k+1}^m) & i=2l+1, j=2k \\ \frac{1}{16}(2\,\mathbf{p}_{l-1,k}^m + 6\,\mathbf{p}_{l,k}^m + 6\,\mathbf{p}_{l,k+1}^m + 2\,\mathbf{p}_{l+1,k+1}^m) & i=2l, \quad j=2k+1 \\ \frac{1}{16}(2\,\mathbf{p}_{l+1,k}^m + 6\,\mathbf{p}_{l,k}^m + 6\,\mathbf{p}_{l+1,k+1}^m + 2\,\mathbf{p}_{l,k+1}^m) & i=2l+1, j=2k+1 \end{cases} \tag{3}$$

See Figure 1 for several generations of a recursively refined triangle mesh approximating the box-spline basis function and Fig. 2 for a geometric interpretation of the rules.

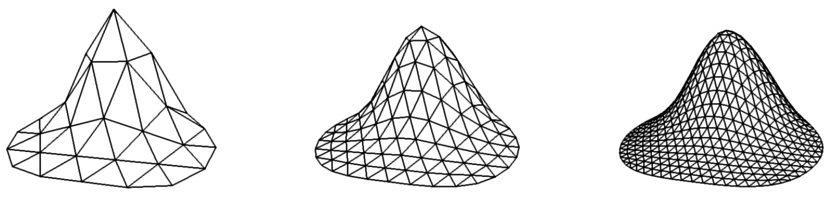

Fig. 1. Uniformly subdivided regular triangle meshes converging to the quartic box-spline M_{222}.

Several authors have generalized these rules to meshes with arbitrary connectivity [13, 23]. To do this we have to leave the formal setup of refinement rules of the type (3) since regularly indexing the vertices is no longer possible. Due to the small support of the refinement masks in Loop's scheme (cf. Fig. 2), there is no need to modify the "edge"-rules: An inner edge is always adjacent to two triangles. Hence the generalization can be restricted to the definition of alternative "vertex"-rules

$$\mathbf{p}^{m+1} := \frac{\alpha(k)}{\alpha(k)+k}\,\mathbf{p}^m + \frac{1}{\alpha(k)+k}\sum_i \mathbf{p}_i^m$$

for vertices $\mathbf{p}^m$ with valence $k \neq 6$ and $\mathbf{p}_i^m$ being the direct neighbors of $\mathbf{p}^m$ in P_m. A good choice leading to overall C^1 limit surfaces is [24]

$$\alpha(k) = k\,\frac{1-\beta(k)}{\beta(k)}, \qquad \beta(k) = \frac{5}{8} - \frac{(3+2\,\cos(2\,\pi/k))^2}{64}.$$

Notice that this rule coincides with the original "vertex"-rule (3) if the valence of $\mathbf{p}^m$ is 6.
Remark: At the boundary of *open* triangle meshes, we cannot apply the above masks since some of the neighboring vertices are missing. We avoid this problem by treating the boundary of a mesh as a closed polygon and applying univariate subdivision. By doing this we additionally guarantee that no internal control vertex influences the shape of the boundary curve. This is important if we want to generate creases or join two separate subdivision surfaces along a common curve in a C^0 fashion [10].

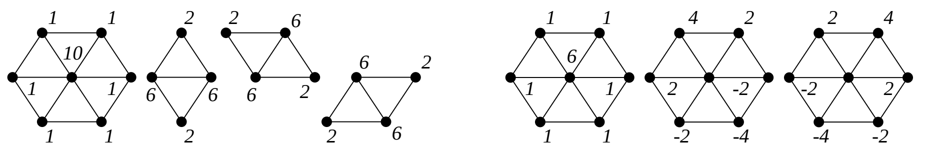

Fig. 2. Geometric notation for the refinement rules (3) and for the limit rules (limit point and the two partial derivatives).

The support of the refinement rules implies that the 1-ring neighborhood of a vertex $\mathbf{p}^{m+1}$ only depends on the 1-ring neighborhood of $\mathbf{p}^m$ (cf. Fig. 2). Hence, the local subdivision matrix S for a regular vertex with valence 6 is

$$S \;=\; \frac{1}{16}\begin{pmatrix} 10 & 1 & 1 & 1 & 1 & 1 & 1 \\ 6 & 6 & 2 & 0 & 0 & 0 & 2 \\ 6 & 2 & 6 & 2 & 0 & 0 & 0 \\ 6 & 0 & 2 & 6 & 2 & 0 & 0 \\ 6 & 0 & 0 & 2 & 6 & 2 & 0 \\ 6 & 0 & 0 & 0 & 2 & 6 & 2 \\ 6 & 2 & 0 & 0 & 0 & 2 & 6 \end{pmatrix} \;=\; V^{-1} D V$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & -1 & 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & -1 & -1 & -3 & 1 \\ 1 & -1 & 0 & 1 & 0 & 0 & -1 \\ 1 & -1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & -1 & -1 & -3 & -1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \frac{1}{16}\begin{pmatrix} 16 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix} \frac{1}{12}\begin{pmatrix} 6 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 2 & -2 & -4 & -2 & 2 & 4 \\ 0 & -2 & -4 & -2 & 2 & 4 & 2 \\ -6 & -1 & -1 & 5 & -1 & -1 & 5 \\ -6 & 5 & -1 & -1 & 5 & -1 & -1 \\ 6 & -1 & -1 & -1 & -1 & -1 & -1 \\ 0 & -2 & 2 & -2 & 2 & -2 & 2 \end{pmatrix}$$

The coefficients of the linear combination for the limit point and the tangents are given by the first three rows of V (cf. Fig 2 for the explicit masks). The tangent masks yield the two partial derivatives at the limit point, and the normal vector can be obtained by their cross product. In [8] we give a complete table of the limit mask coefficients for valences $k = 3, \ldots, 12$.

3 Oriented bounding volumes

To derive bounding volumes for subdivision curves and surfaces we have to know the *ranges* of the basis functions $\phi(t-i)$ over the considered interval. This is not straight forward since we just know the coefficients of the refinement equation and do not have an explicit parameterization in terms of polynomials or such. We will first explain how to compute oriented bounding boxes before we present a simple iterative procedure which yields tight estimates for the actual bounds.

Let a curve $P(t)$ be given by a linear combination of scalar valued basis functions $\phi(t-i)$ as in (1). Its range with respect to some direction $\mathbf{n}$ for $t \in [a,b]$ can be estimated by

$$\sum_i (\mathbf{n}^T\mathbf{p}_i)_+ \min_{t\in[a,b]} \phi(t-i) + \sum_i (\mathbf{n}^T\mathbf{p}_i)_- \max_{t\in[a,b]} \phi(t-i) \;\le\; P(t),$$

$$P(t) \;\le\; \sum_i (\mathbf{n}^T\mathbf{p}_i)_+ \max_{t\in[a,b]} \phi(t-i) + \sum_i (\mathbf{n}^T\mathbf{p}_i)_- \min_{t\in[a,b]} \phi(t-i)$$

where

$$(\mathbf{n}^T\mathbf{p}_i)_+ \;:=\; \max\{\mathbf{n}^T\mathbf{p}_i, 0\}, \qquad (\mathbf{n}^T\mathbf{p}_i)_- \;:=\; \min\{\mathbf{n}^T\mathbf{p}_i, 0\}.$$

Example: For the cubic B-spline it is known that

$$N(\{0,1,2,3,4\}) \;=\; \{0, \tfrac{1}{6}, \tfrac{4}{6}, \tfrac{1}{6}, 0\}$$

and N is monotonic on each integer interval such that the extremal values occur at the uniform knots. Consider the spline curve $P(t) = \sum_i \mathbf{p}_i N(t-i)$ over the interval $t \in$

$[j, j+1]$. Here the curve is completely determined by the control vertices $\mathbf{p}_{j-1}, \ldots \mathbf{p}_{j+2}$ due to the finite support of N. With the limit point rule we obtain the two points $\mathbf{q}_j := P(j)$ and $\mathbf{q}_{j+1} := P(j+1)$ on the limit curve. The chordal error of the straight line $\overline{\mathbf{q}_j\,\mathbf{q}_{j+1}}$ with respect to the arc $P([j, j+1])$ can be estimated by computing the range of P in the normal direction $\mathbf{n}_j$ perpendicular to the chord $\overline{\mathbf{q}_j\,\mathbf{q}_{j+1}}$. Due to the affine invariance we can shift the control vertices by $-\mathbf{q}_j$ and obtain the control vertices' normal distances $r_i := \mathbf{n}_j^T\,(\mathbf{p}_i - \mathbf{q}_j)$ for $i = j-1, \ldots, j+2$. The normal range

$$[l_j, u_j] \;=\; \frac{1}{6}\Big[\big(r_j^+ + r_{j+1}^+ + r_{j-1}^- + 4\,r_j^- + 4\,r_{j+1}^- + r_{j+2}^-\big),\, \big(r_j^- + r_{j+1}^- + r_{j-1}^+ + 4\,r_j^+ + 4\,r_{j+1}^+ + r_{j+2}^+\big)\Big]$$

defines a rectangular box which is aligned to the chord $\overline{\mathbf{q}_j\,\mathbf{q}_{j+1}}$ and completely contains the arc $P([j, j+1])$.

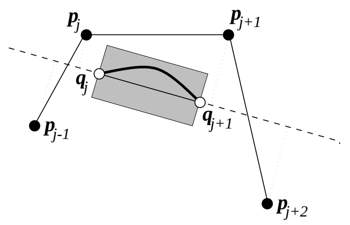

Fig. 3. The gray bounding box is spanned by shifts of the chord $\overline{\mathbf{q}_j\,\mathbf{q}_{j+1}}$ (hollow dots) in normal direction $\mathbf{n}_j$. The actual normal range is computed by a weighted sum of the normal distances r_i of the control vertices $\mathbf{p}_i$ from the supporting line of the chord (dashed line).

In the bivariate setting the procedure to estimate the chordal approximation error is exactly the same: given a submesh which defines one segment of the limit surface (corresponding to one triangle in the control mesh) we use the limit masks to obtain points on the surface spanning a chordal triangle T. As in the univariate case we derive the normal distances r_i of all involved control vertices $\mathbf{p}_i$ from the supporting plane of T and compute a weighted sum according to the ranges of the associated basis functions. The result is an orthogonal triangular prism with the top and bottom face being shifted versions of the chordal triangle T. The possibility that the patch might intersect the quadrilateral sides of the prism will be addressed in Section 4.

For irregular meshes we have to consider the different special cases that occur at extraordinary vertices since the ranges of the basis functions do depend on the local connectivity of the mesh. For the sake of simplicity we assume that the mesh has been uniformly subdivided once before the bounding boxes are to be computed. This reduces the number of special cases since each extraordinary vertex with valence $\neq 6$ has only regular direct neighbors (cf. Fig. 4). Since there is no explicit parameterization for the limit surface of Loop's subdivision scheme, we have to find a reliable numerical algorithm to estimate the true ranges. It turns out that this already is a first application of the bounding volume technique in itself.

The control mesh defining a triangular *Loop-patch* is shown in Fig 4. To approximate the basis function corresponding to one of the vertices, we assign $z = 1$ to it and zero z-values to all other vertices (*Dirac-mesh*). The iterative refinement of such a mesh will approach the specific basis function as depicted in Fig. 1 but it won't provide a reliable bound on the actual range. Notice that we are only interested in the basis function's range over the center triangle.

Let us start with a very coarse over-estimation of the true ranges, e.g. $[-1, 1]$. We apply the subdivision rules to the Dirac-mesh in Fig. 4 and after several subdivision steps we compute the chord aligned bounding boxes for every triangle. Since the mesh has become locally flat, the chord aligned boxes are also flat and provide a much tighter

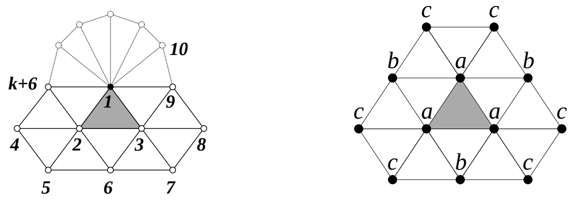

Fig. 4. The mesh on the left contains all vertices that have influence on the surface patch corresponding to the center triangle. Each vertex is associated with a specific basis function. On the right the vertices are symmetrically labeled according to the ranges of their basis functions over the center triangle ($a = [\frac{1}{12}, \frac{1}{2}]$, $b = [0, \frac{13}{96}]$, $c = [0, \frac{1}{12}]$).

bound for the range of the current basis function. Once we have done this for all involved basis functions, we can iterate the whole procedure to obtain even better bounds in every step. Fig. 4 shows the exact ranges for the 12 basis functions whose support covers the center triangle in the regular setting. For a complete table with min/max range values of the $k + 6$ basis functions over a triangle being adjacent to a valence $k = 3, \ldots, 12$ vertex cf. [8].

4 Bounding envelopes

Each triangle of a subdivision surface's control mesh corresponds to a patch segment of the limit surface. When uniformly subdividing the mesh we generate a sequence of meshes whose triangles can be grouped as a forest of quad-trees with each triangle of the original mesh being a root node. In Section 3 we showed how to compute a bounding prism for each triangular sub-patch (on any subdivision level). However, using the individual bounding prisms directly for ray intersection tests is quite complicated since each is aligned to the particular normal direction of the underlying chord triangle.

Further, when we derived the bounding prisms, we did not address the problem that bounding the range in the direction perpendicular to the chordal triangle is not sufficient since the triangular patch might intersect one of the prism's quadrilateral faces. Hence we would have to enlarge the prism to guarantee inclusion. Moreover, testing whether a ray intersects the prism is not trivial since several special configurations have to be checked.

We therefore introduce a pre-processing step where we combine all chord aligned bounding prism to build a global *bounding envelope*. This is a continuous triangle mesh having the same topology and complexity as the control mesh and which tightly circumscribes the limit surface. Obviously for open meshes we have to compute *two* envelopes: one covering the front side and one for the back side.

Each local refinement operation of the control mesh induces a corresponding refinement of the envelope meshes. As the refinement proceeds the envelopes quickly approach the limit surface. In the next section we will explain a simple ray intersection procedure which traces recursively through the forest of mesh triangles thereby testing as few prisms as possible. By using the envelope structure to navigate, we exploit the topological coherence in the mesh, i.e., the fact that a triangle's neighbor can be found with $O(1)$ complexity. The *continuity* of the hull on each subdivision level guarantees that no intersection is missed.

Again, we explain the general envelope construction in the univariate setting for the sake of simplicity. So far we have derived an individual bounding box for each *segment* by estimating the limit curve's range $[l_j, u_j]$ in the direction $\mathbf{n}_j$ perpendicular to the chord $\overline{\mathbf{q}_j\,\mathbf{q}_{j+1}}$. In order to obtain continuous bounding polygons we use the limit tangent rule to compute a (normalized) normal vector $\tilde{\mathbf{n}}_j$ for each limit point $\mathbf{q}_j$. The four points

$$\mathbf{q}_j + \frac{l_j}{\mathbf{n}_j^T \tilde{\mathbf{n}}_j}\,\tilde{\mathbf{n}}_j, \quad \mathbf{q}_j + \frac{u_j}{\mathbf{n}_j^T \tilde{\mathbf{n}}_j}\,\tilde{\mathbf{n}}_j, \quad \mathbf{q}_{j+1} + \frac{l_j}{\mathbf{n}_j^T \tilde{\mathbf{n}}_{j+1}}\,\tilde{\mathbf{n}}_{j+1}, \quad \mathbf{q}_{j+1} + \frac{u_j}{\mathbf{n}_j^T \tilde{\mathbf{n}}_{j+1}}\,\tilde{\mathbf{n}}_{j+1}$$

define a minimal trapezoid containing the curve segment $P([j, j+1])$. Since two segments meet at each limit point $\mathbf{q}_j$, we define the conservative range

$$[\tilde{l}_j, \tilde{u}_j] \;=\; \left[\min\{\frac{l_{j-1}}{\mathbf{n}_{j-1}^T \tilde{\mathbf{n}}_j}, \frac{l_j}{\mathbf{n}_j^T \tilde{\mathbf{n}}_j}\}, \max\{\frac{u_{j-1}}{\mathbf{n}_{j-1}^T \tilde{\mathbf{n}}_j}, \frac{u_j}{\mathbf{n}_j^T \tilde{\mathbf{n}}_j}\}\right]$$

for each vertex of the control polygon. Hence, for a given control polygon $[\mathbf{p}_i^m]$ we obtain a *limit polygon* $[\mathbf{q}_i^m]$, an *upper envelope* $[\mathbf{q}_i + \tilde{u}_i\,\tilde{\mathbf{n}}_i]$ and a *lower envelope* $[\mathbf{q}_i + \tilde{l}_i\,\tilde{\mathbf{n}}_i]$.

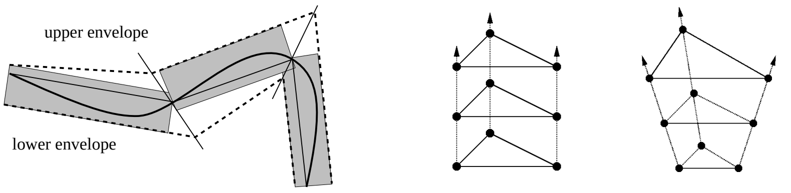

Fig. 5. By computing the ranges $[\tilde{l}_j, \tilde{u}_j]$ for each vertex we find an envelope for the chord based bounding boxes. The envelope's boundaries have the same "topology" as the control polygon (left). In the bivariate case we also compute normal ranges with respect to vertices instead of triangles and replace the orthogonal prism (center) by more general bounding volumes (right). Although each individual bounding object becomes more complicated, the continuity of the bounding envelope enables a simple recursive tracing procedure.

A closed envelope in the bivariate setting is obtained by replacing the orthogonal prisms of Section 3 by more general prisms that are generated by moving each vertex $\mathbf{q}_j$ of the chord triangle T in the corresponding vertex normal direction $\tilde{\mathbf{n}}_j$ (cf. Fig 5). As in the univariate case we compute a normal range for each vertex by taking the minimum and maximum of the *projected* normal ranges of all adjacent triangles (and their orthogonal prisms). After this procedure we end up with three topologically equivalent meshes (derived from the original control mesh), one being a chordal approximant to the limit surface and two meshes (outer and inner envelope) which can be used as bounding volume.

5 Efficient ray intersection

Free form surfaces are usually defined as a collection of individual patches. The standard way to implement a ray intersection test with such objects is to compute a simple bounding volume for each patch. These bounding volumes can be used as a middle layer of a bounding volume hierarchy which is propagated towards the root by enclosing neighboring bounding volumes into a circumscribed larger volume and towards the leaves by subdividing the patches and computing bounding volumes for the sub-patches.

Tracing a ray through such a bounding volume hierarchy means traversing a tree data structure with the descent being controlled by ray intersection tests for the current node's volume. If a leaf is reached, a numerical procedure like Newton iteration is applied to the explicit polynomial parameterization of the patch.

Since subdivision surfaces do not have an explicit parameterization, we propose to base the recursive ray tracing procedure mainly on the information provided by the envelope meshes. Many techniques have been suggested to accelerate the ray intersection tests with a collection of triangles [1]. The most effective one is to build a hierarchical space partition (e.g., BSP-trees) for the whole mesh and incrementally trace the ray through this decomposition. We do use this technique in our algorithm but only to find those triangles where the ray enters the outer envelope corresponding to the *initial* control mesh. Once the entry triangle is found, we trace the ray through the hierarchy of adaptively refined envelope triangles. The navigation is controlled by simple ray–triangle tests. The efficiency of the tracing procedure results from the fact that chord aligned bounding boxes converge much faster than axis-aligned ones (cf. Sect. 6). Another advantage of this strategy is that no redundancy is introduced by overlapping bounding prisms or ambiguous assignment of bounding prisms to a space partition.

To simplify the explanation we assume that all necessary local refinement operations have already been performed when the tracing path intersects a certain triangle. In our actual implementation we used a *lazy-evaluation* mechanism which computes the control vertices of the refined meshes on demand while keeping the overall data structure consistent (i.e. adjacent leaf-triangles may only differ by one generation) and caching all computed information for future requests.

The tracing algorithm has to handle all special cases of the ray intersecting a triangle of the envelope mesh (on a certain refinement level) but intersecting the surface itself at a neighboring patch (if it intersects at all). The most important feature that we exploit for the tracing algorithm is that the envelopes corresponding to the same subdivision level form a continuous surface and the ray cannot pass through without hitting at least one triangle. In one tracing step we either descend in the forest of refined envelope triangles or we proceed the search in a neighboring tree if no intersection can be found below the current one.

We base the local decision on *intersection masks* specifying the seven possible spatial configurations between a ray and a triangle (cf. Fig. 6). For a triangle $T = \triangle(A,B,C)$ and a ray with origin O and direction $\mathbf{r}$ we distinguish the configurations according the signs of the coefficients α, β, and γ in the unique linear combination

$$\mathbf{r} \;=\; \alpha\,(A-O) + \beta\,(B-O) + \gamma\,(C-O).$$

Only the $+\,+\,+$-case reports an intersection of the ray with T. All other cases indicate failure but they provide useful information about where to search for an intersection, i.e. which neighboring triangle to check next.

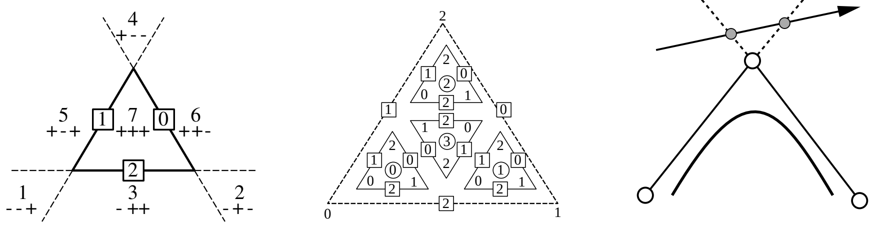

Fig. 6. The bits in the intersection mask define a partition of the triangle's supporting plane. They indicate in which direction the tracing algorithm has to proceed (left). The indexing of triangle corners and edges is inherited from the parent triangle to preserve the orientation (center). A ray passing the envelope triangle mesh without intersection does intersect the associated supporting planes in reverse order. Translated into the intersection masks of the corresponding triangles this configuration would cause a loop in the tracing algorithm (right).

In each step we check if one of the current node's children is intersected, too. If so, we proceed the search with the descendents of this child. If not, we use the intersection masks obtained while testing the children to decide which neighbor of T to test next. The following pseudo code implements the algorithm

```
trace($T$)
    for child$_0$ to child$_3$
        mask[i] = test_intersect(child$_i$)
        if mask[i] = '+++'
            trace(child$_i$)
    lookup neighbor(mask[3],mask[sibling(mask[3])])
    trace($T$->neighbor)
```

Cf. Fig. 6 for our indexing convention of a triangle and its four children. If none of the child triangles is intersected by the ray the tracing is controlled by two tables `neighbor` and `sibling`. The `child`$_3$ is the center child and if the corresponding intersection mask has two negative signs then the next neighbor of T is uniquely determined. If only one of the signs is negative, then we use the corresponding sibling's intersection mask to determine where to proceed. If the sibling's mask has two negative signs then we proceed (by convention) in the counter clockwise direction to make the decision deterministic. `sibling` is indexed by `mask[3]` and `neighbor` is row indexed by `mask[sibling(mask[3])]` and column indexed by `mask[3]`.

```
int sibling[7] = {0,0,0,2,0,1,0};
int neighbor[49] = {
    -1, -1, -1, -1, -1, -1, -1,      /* 2 1 0       */
    -1,  0,  0,  0,  0,  0,  0,      /* - - +  :  1 */
    -1,  1,  1,  1,  1,  1,  1,      /* - + -  :  2 */
    -1,  1,  0, -1,  1,  1,  0,      /* - + +  :  3 */
    -1,  2,  2,  2,  2,  2,  2,      /* + - -  :  4 */
    -1,  2,  0,  2,  0, -1,  0,      /* + - +  :  5 */
    -1,  2,  2,  2,  1,  1, -1 };    /* + + -  :  6 */
```

Don't-care-cases in `sibling` are set to zero and `-1`s in `neighbor` indicate configurations where no intersection occurs. Notice that the case `mask[i] = 7` cannot occur since in this case the algorithm descends recursively and the procedure to determine the next neighbor is not called.

Failure of the recursive algorithm (i.e. no intersection occurs) can be detected as follows: The generic situation of a ray passing the object without intersection is depicted in Fig. 6. Such configurations are characterized by the fact that the intersection mask of a triangle T_1 indicates the search to be continued in triangle T_2 while T_2's intersection masks suggest to test T_1. Hence failure can easily be detected by avoiding loops in the tracing path. We implemented this feature by giving each ray a unique identification (an integer) and putting a temporary stamp on each tested triangle:

```
trace($T$)
    if touched($T$) exit on FAILURE
    touch(T)
    ...
```

6 Approximation tolerance

To eventually compute an intersection point of the ray with the surface, we have to define a stopping criterion to determine whether a bounding prism is sufficiently small in the sense that replacing the true surface geometry by a linear approximant does not lead to an error larger than some prescribed ε. We approximate the location of the true intersection point by computing the intersection of the ray with a chord triangle. For the orthogonal bounding prisms obtained by shifting the chordal triangle in normal direction the error can be estimated by

$$E \;\leq\; \min\left\{ l, \frac{h}{|\mathbf{n}^T \mathbf{r}|} \right\} \tag{4}$$

with l being the longest edge of the chord triangle, h being the height of the prism, $\mathbf{n}$ the normal vector and $\mathbf{r}$ the direction of the ray (cf. Fig 7).

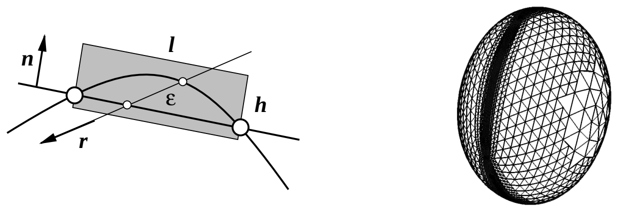

Fig. 7. The maximum distance between the true intersection point and its approximation depends on the height h of the bounding prism *and* the relative direction of the ray. For very steep angles the maximum edge length of the supporting chord triangle becomes the dominant bound (left). Since the stopping criterion for the subdivision depends on the direction of the ray, the refinement tends to be much finer close to the contour (right).

This reveals the significant advantage of geometry aligned boxes over axis-aligned ones: Subdivision of axis-aligned bounding boxes approximately halves the size of the box in every step, hence going down one level in the bounding volume hierarchy provides one more bit in the precision of the result. In our case however, the *height* of the boxes shrinks much faster than the length of the triangle edges. This behavior can easily be explained by looking at the eigenstructure of the subdivision matrix S. The first eigenvalue $\lambda_1 = 1$ and the associated eigenvector is responsible for the local convergence to a point on the limit surface. The second and third subdominant eigenvalues $\lambda_2 = \lambda_3$ and the associated (two dimensional) eigenspace define the tangent plane. The fourth eigenvalue λ_4 controls the flattening rate by which the deviation of the local sub-mesh from a plane configuration reduces.

In the above example of Loop's scheme in the vicinity of a regular vertex, we have $\lambda_4 = \frac{1}{4}$ and hence one subdivision step approximately bisects the edges of the chord triangles but their height is reduced by the factor 4. As a consequence the tracing algorithm turns out to be significantly faster, especially if high precision is required.

The reciprocal factor $|\mathbf{n}^T\mathbf{r}|$ implies a certain degree of view dependency in the refinement during ray tracing. If the ray intersects almost perpendicularly, the refinement can stop as soon as the height of the bounding prism is smaller than the prescribed tolerance ε. The bounding prisms for triangles close to the visual contour of the object have to be refined much further. Fig. 7 shows the adaptive refinement resulting from rendering a simple convex object by a ray tracing algorithm with fixed tolerance for the intersection tests. To avoid numerical instabilities, we consider intersection tests with sufficiently small bounding prisms (according to (4)) where the ray does not hit the interior of the associated chord triangle as failure.

7 Results and conclusions

Loop subdivision surfaces were integrated as a new geometric primitive into a generic architecture for rendering algorithms [18]. The color plates show several pictures rendered with a simple ray tracer. We chose one very simple example (the initial control mesh being an octahedron) to demonstrate that intersection tests at the contours can be evaluated in a stable manner (upper left). The reflectivity of the material was set to a high value, making the surrounding room's striped wallpaper fully visible on the resulting surface. The smooth reflection lines demonstrate the surface's C^2-continuity.

The initial triangle mesh defining the cat model in the lower right image consists of 728 faces. As in the image of the reflecting octahedron the material properties were set to full reflection to obtain clearly visible reflection lines. The image on the right shows a raytraced subdivision surface with a control mesh consisting of 1374 triangles. A texture was added to define the head's color. Again the smooth shape of the highlights indicate curvature continuity of the surface.

In practice we found subdivision surfaces a good extension to our rendering system, as the control meshes are easy to manipulate without having to worry about continuity issues. Even meshes only roughly outlining an object's shape produce good looking surfaces. We presented the necessary theory for the derivation of tight chord aligned bounding volumes for subdivision surfaces which allow us to effectively handle ray intersection inquiries due to their fast convergence under subdivision. The introduction of continuous envelope meshes completely containing the limit surface gives rise to an efficient algorithm that enumerates all potentially intersected envelope triangles.

References

1. W Barth, W. Stürzlinger, *Efficient ray tracing for Bezier and B-spline surfaces*, Comput Graph 17:423-430
2. E. Catmull, J. Clark, *Recursively generated B-spline surfaces on arbitrary topological meshes*, CAD 10 (1978), 350–355
3. C. de Boor, K. Höllig, S. Riemenschneider, *Box Splines*, Springer Verlag, 1993
4. D. Doo, M. Sabin, *Behaviour of Recursive Division Surfaces Near Extraordinary Points*, CAD 10 (1978), 356–360
5. N. Dyn, J. Gregory, D. Levin, *A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control*, ACM ToG 9 (1990), 160-169
6. N. Dyn, *Subdivision Schemes in Computer Aided Geometric Design*, Adv. Num. Anal. II, W.A. Light ed., Oxford Univ. Press, 1991, 36–104.
7. A. Fournier, J. Buchanan *Chebyshev polynomials for boxing and intersections of parametric curves and surfaces*, Comp Graph Forum (Eurographics 94 issue), 127–142
8. `http://www9.informatik.uni-erlangen.de/~kobbelt/RT_of_SS.html`
9. J. Kajiya, *Ray tracing parametric patches* SIGGRAPH 82 proceedings, 245–254
10. L. Kobbelt, *Interpolatory Subdivision on Open Quadrilateral Nets with Arbitrary Topology*, Comp Graph Forum (Eurographics '96 issue), C407–C420
11. L. Kobbelt, *Discrete Fairing*, Proceedings of the 7th IMA Conference on the Mathematics of Surfaces, Information Geometers, 1997, 101 – 131
12. L. Kobbelt, S. Campagna, J. Vorsatz, H-P. Seidel, *Interactive Multi-Resolution Modeling on Arbitrary Meshes*, to appear in SIGGRAPH 98 proceedings
13. C. Loop, *Smooth Subdivision for Surfaces Based on Triangles*, University of Utah, 1987
14. M. Lounsbery, T. DeRose, J. Warren *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*, ACM ToG 16 (1997), 34–73
15. T. Nishita, T. Sederberg, M. Kakimoto *Ray tracing trimmed rational surface patches* SIGGRAPH 90 proceedings, 337–345
16. U. Reif, *Neue Aspekte in der Theorie der Freiformaflächen beliebiger Topologie*, Universität Stuttgart, 1993
17. M. Sweeney, R. Bartels, *Ray tracing free-form B-spline surfaces*, IEEE Comput Graph Appl 6 (1986), 41–49
18. Slusallek P, Seidel H-P, *Vision - an architecture for global illumination calculations*, IEEE Trans Vis Comput Graph 1 (1995), 77–96
19. E. Stollnitz, T. DeRose, D. Salesin *Wavelets for Computer Graphics*, Morgan Kaufmann Publishers, 1996
20. D. Toth, *On ray tracing parametric surfaces*, SIGGRAPH '85 proceedings, 171–179
21. C. Yang, *On speeding up ray tracing of B-spline surfaces*, CAD 19 (1987), 122–130
22. D. Zorin, P. Schröder, W. Sweldens, *Interpolating subdivision for meshes with arbitrary topology*, SIGGRAPH 96 proceedings, 189–192
23. D. Zorin, P. Schröder, W. Sweldens, *Interactive Multiresolution Mesh Editing*, SIGGRAPH 97 proceedings, 259–269
24. D.Zorin C^k *Continuity of Subdivision Surfaces*, California Institute of Technology, 1996