A Marching Method for Computing Intersection Curves of Two Subdivision Solids

Xu-Ping Zhu[1], Shi-Min Hu[1], Chiew-Lan Tai[2], and Ralph R. Martin[3]

[1] Department of Computer Science and Technology, Tsinghua University, Beijing, China
[2] Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, China
[3] School of Computer Science, Cardiff University, Cardiff, Wales
ralph@cs.cf.ac.uk

Abstract. This paper presents a marching method for computing intersection curves between two solids represented by subdivision surfaces of Catmull-Clark or Loop type. It can be used in trimming and boolean operations for subdivision surfaces. The main idea is to apply a marching method with geometric interpretation to trace the intersection curves. We first determine all intersecting regions, then find pairs of initial intersection points, and trace the intersection curves from the initial intersection points. Various examples are given to demonstrate the robustness and efficiency of our algorithm.

1 Introduction

Subdivision surfaces are defined as the limit of repeated refinement of 3D control meshes using specific subdivision rules [4, 5, 9, 12]. Due to their advantages, such as being able to handle arbitrary topology and ease of coding, they are widely used in computer animation and game engines [16], for example. However, applications of subdivision surfaces to industrial design are still infrequent, one reason being that is difficult to construct complex subdivision models using the usual solid operations. This problem is mainly due to the lack of suitable geometric algorithms for computing intersection curves, offsets, blending, trimming, and Boolean operations.

Some such algorithms do already exist. Litke et al. [11] introduced a new method for trimming subdivision surfaces, which is based on the combined subdivision schemes to guarantee exact interpolation of trim curves. Biermann et al. [3] presented a method for computing approximate results of Boolean operations (union, intersection, difference) for free-form solids bounded by multiresolution subdivision surfaces. Both works cite the problem of computing intersection curves calculation as an open problem. Nasri [13] presented a general framework for intersecting two recursive subdivision surfaces based on divide and conquer methods to process complete surfaces. Instead, our goal is to apply a marching method with geometric interpretation to trace the intersection curves based on

R. Martin, H. Bez, M. Sabin (Eds.): Mathematics of Surfaces 2005, LNCS 3604, pp. 458–471, 2005.

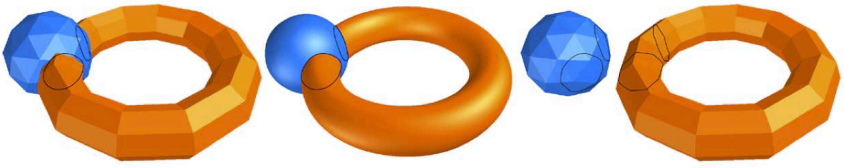

Fig. 1. Intersection curves

an inital intersection point. Grinspun et al. [6] developed an algorithm for detecting interference of subdivision surfaces. He used normal bounds to determine whether a surface interferes with itself or other surfaces. In this paper, we focus on computing all intersection curves between two subdivision surfaces.

Computing intersection curves for parametric and implicit surfaces has been extensively investigated [1, 2, 7, 10]. However, for subdivision surfaces, there are two main difficulties: analytical representation and parameterization. A breakthrough was made by Stam [15, 16], who described an approach for evaluating subdivision surfaces at arbitrary parameter values in the cases of both Catmull-Clark [4] and Loop [12] schemes. Building on the results of his work, we show how traditional algorithms for parametric surfaces can now be applied to subdivision surfaces. The main contribution of this paper is to extend the *moving affine frame* (MAF) marching method [8] to subdivision surfaces. In addition, we present a complete method to trace all intersection curves. Our approach has many applications, to trimming and Boolean operations, for example.

An example using our approach is shown in Figure 1 of the intersection between a Loop sphere and a Catmull-Clark torus. On the left, the two intersection curves are shown together with the given initial control meshes. The result plus both limit surfaces is shown in the centre middle, and the right hand figure gives pre-images of both curves.

The rest of the paper is organized as follows. Section 2 gives a brief review of parameterization of subdivision surfaces. In Section 3, we review the MAF marching method for tracing intersection curves for parametric surfaces, and extend it to subdivision surfaces. Section 4 presents an algorithm for calculating all intersection curves of subdivision surfaces. Section 5 shows various results and conclusions are given in Section 6.

2 Local Parameterization of Subdivision Surfaces

A subdivision surface is defined by an initial control mesh and a set of subdivision rules. As the control mesh is successively refined according to the rules, a sequence of meshes with an increasing numbers of faces is obtained. In the limit, a smooth surface is obtained (Figure 2). Intuitively, the initial control mesh can be considered the domain of the limit surface; each initial face of the control mesh is mapped to a patch on the limit surface. We call these faces the *domain faces* and denote the map as: $(i, u, v) \rightarrow (x, y, z)$ where i indexes the domain

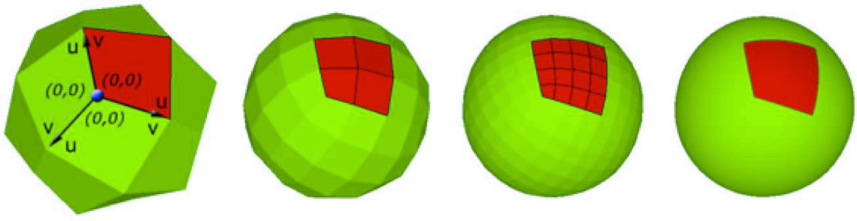

Fig. 2. Mapping a domain face to its corresponding patch. The extraordinary vertex (blue) has parameter (0,0) in all three surrounding extraordinary domain faces

faces, $u, v \in (0, 1)$ are parameter values on the i-th domain face, and (x, y, z) are 3D coordinates of the corresponding limit point on the surface. As in Stams work, we make two assumptions

- The initial control mesh has already been subdivided once so that each domain face of the control mesh contains at most one extraordinary vertex. For a quadrilateral (or triangular) mesh, a vertex having a valence not equal to four (or six, respectively) is called an *extraordinary vertex*. A domain face with an extraordinary vertex is called an *extraordinary domain face*. For example, the blue extraordinary vertex in Figure 2 belongs to three extraordinary domain faces.
- The parameterization is organized so that each extraordinary vertex has parameters $(0, 0)$ in all the extraordinary domain faces containing it.

3 Tracing a Branch of an Intersection Curve

3.1 MAF Marching Method for Parametric Surfaces

We first introduce the *moving affine frame* marching method [8] for parametric surfaces. Assume we are given two parametric surfaces $\mathbf{r}_1$ and $\mathbf{r}_2$, and an initial intersection point $\mathbf{M}$. Let $(u_i, v_i), i = 1, 2$ be the parameter values of $\mathbf{M}$ on each of the two surfaces. The goal is to step along the intersection curve and find the next intersection point. The MAF method gives a stepping size for finding the next intersection point. Let the desired stepping distance be δ. The stepping direction is along the tangent vector to the intersection curve, which is given by $\mathbf{T} = \mathbf{N}_1 \times \mathbf{N}_2$, where $\mathbf{N}_i$ is the normal vector of $\mathbf{r}_i$ at $\mathbf{M}$, i.e. $\mathbf{N}_i = \partial\mathbf{r}_i/\partial u_i \times \partial\mathbf{r}_i/\partial v_i, i = 1, 2$. Thus, the target point is $\mathbf{H} = \mathbf{M} + \delta\mathbf{T}$. Since $\mathbf{H}$ lies on the tangent planes of both surfaces, which are defined by the affine frames $\{\mathbf{M}; \partial\mathbf{r}_i/\partial u_i \partial\mathbf{r}_i/\partial v_i\}, i = 1, 2$, the vector $\mathbf{H} - \mathbf{M}$ can be written as:

$$\mathbf{H} - \mathbf{M} = \delta\mathbf{T} = (\partial\mathbf{r}_i/\partial u_i)\Delta u_i + (\partial\mathbf{r}_i/\partial v_i)\Delta v_i, \quad i = 1, 2 \tag{1}$$

Thus, the parameter increments, $\Delta u_i, \Delta v_i$, can be calculated as using the following triple products:

$$\begin{aligned} \Delta u_i &= [\mathbf{H} - \mathbf{M}, \partial\mathbf{r}_i/\partial v_i, \mathbf{N}_i]/[\partial\mathbf{r}_i/\partial u_i, \partial\mathbf{r}_i/\partial v_i, \mathbf{N}_i] \\ \Delta v_i &= [\mathbf{H} - \mathbf{M}, \partial\mathbf{r}_i/\partial u_i, \mathbf{N}_i]/[\partial\mathbf{r}_i/\partial v_i, \partial\mathbf{r}_i/\partial u_i, \mathbf{N}_i] \end{aligned} \tag{2}$$

With these increments, two new points can be computed as $\mathbf{P}_i = \mathbf{r}_i(u_i + \Delta u_i, v_i + \Delta v_i), i = 1, 2$. This procedure of computing $\mathbf{P}_i$ from a target point $\mathbf{H}$ is called a *sphere transformation*.

If $\mathbf{P}_1$ and $\mathbf{P}_2$ are sufficiently close, i.e. $||\mathbf{P}_1\mathbf{P}_2|| < \epsilon$, the MAF algorithm outputs $(\mathbf{P}_1 + \mathbf{P}_2)/2$ as the next intersection point. Otherwise, it performs a *mid-point transformation* on $\mathbf{P}_i$ as follows. Let π_i be the tangent plane of $\mathbf{r}_i$ at $\mathbf{P}_i$. If $\pi_i, i = 1, 2$ are not parallel, they must intersect in a line l. We project each $\mathbf{P}_i$ to the line l to get new points $\mathbf{R}_i$, and compute the mid-point $\mathbf{S} = (\mathbf{R}_1 + \mathbf{R}_2)/2$.

By applying a sphere transformation to $\mathbf{S}$, we obtain two new points $\mathbf{P}_i$ and repeat the above process of testing $||\mathbf{P}_1\mathbf{P}_2|| < \epsilon$ and, if again failing the test, performing the mid-point transformation, until $||\mathbf{P}_1\mathbf{P}_2|| < \epsilon$.

By repeatedly finding the next intersection point from a given intersection point, we can trace an entire intersection curve. But when should we stop tracing? If the initial surfaces are closed, their intersection curves are also closed. Hence, we use the distance between the current intersection point and the starting point to decide when to stop tracing. If the distance is decreasing and is less than a threshold, such as 2δ, we replace the stepping size δ by a smaller one. If the distance continues to decrease and fall below a smaller threshold, we shorten the stepping size once again and terminate tracing if the distance is less than a prescribed tolerance ϵ. If the initial curves are open, tracing terminates either on forming an intersection loop as above, or when reaching the boundary of either surface. In the latter case we again are using a decreasing step size near the boundary.

3.2 Extending the MAF Method to Subdivision Surfaces

The MAF method is an efficient iterative method with the benefit of a clear geometric interpretation; furthermore it is easy to implement, and it only requires evaluation of points and first-order derivatives. In order to extend the MAF method to subdivision surfaces, we need to resolve two issues:

- How to evaluate the surface and its first-order derivatives efficiently.
- How to find parameter values when updates move outside a domain face, given that we have a local piecewise parameterisation.

The first problem has already been solved by Stam, who has presented an efficient method to evaluate Catmull-Clark and Loop surfaces and all their derivatives at arbitrary parameter values [15, 16]. Hence, we just consider the second problem.

We deal with the quadrilateral mesh case first; we will then look at the triangular mesh case. Assume that the current parameter is (i, u, v). The sphere transformation step computes parameter increments, $\Delta u, \Delta v$, according to a target point $\mathbf{H}$. The new parameter values are then computed as $u = u + \Delta u$, $v = v + \Delta v$. If $u \notin [0, 1]$ or $v \notin [0, 1]$, the parameter will move out of the current domain face into an adjacent domain face. Hence, we must replace the computed parameter (i, u, v) by some (j, u', v'), where j is the index of the target domain face and (u', v') are the new parameter values in that domain face.

If the current domain face is not extraordinary, the target domain face must be one of the eight domain faces around it as depicted in Figure 3a. It is trivial

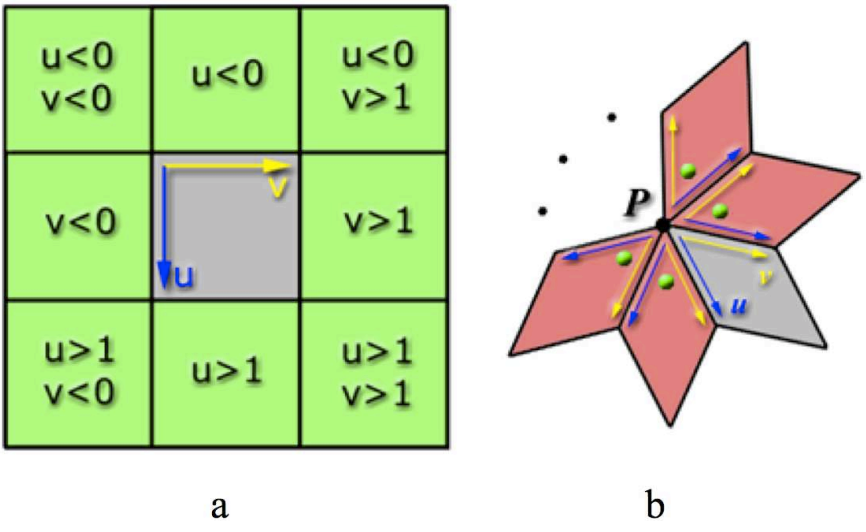

Fig. 3. Identifying parameter values when moving out of the current domain face (quadrilateral case): (a) regular domain face; (b) extraordinary domain face

to find the target domain face from u and v, and to calculate the new parameter values (u', v') according to the u and v directions in that domain face.

If the current domain face is extraordinary, we know that it has only one extraordinary vertex, denoted **P**. By assumption, **P** has parameter $(0, 0)$ in all extraordinary domain faces containing it. Since the other three vertices of the current domain face are all regular, if $u > 0$ or $v > 0$, we can always find a target domain face according to the u and v directions as we do above for the regular case. If $u < 0$ and $v < 0$, the target domain face must be one of the other extraordinary domain faces containing P (red faces in Figure 3b). We need to determine to which in which candidate domain face the parameter values lie, and compute the new parameter values (u', v') in that face. Since $u, v < 0$ and **P** has parameter $(0, 0)$ in all domain faces containing it, we can calculate the distance between (u, v) and P in parametric space, $d = \sqrt{u^2 + v^2}$. We estimate the new parameter values by $u' = v' = \sqrt{d^2/2}$ and evaluate the corresponding points at (u', v') for all the candidate domain faces. Finally, we select that point nearest to the target point **H** and let its parameter values and domain face determine the new parameter values (j, u', v').

Next, we consider the case of triangular mesh. Assume that the current parameter is (i, v, w), and that for symmetry we add an auxiliary parameter $u = 1 - v - w$. In the sphere transformation step, we obtain the parameter increments as before, $\Delta v, \Delta w$, and compute the new parameter values as $v = v + \Delta v, w = w + \Delta w$. If $v \notin [0, 1]$ or $w \notin [0, 1]$ or $u \notin [0, 1]$, the parametric point lies in an adjacent domain face, and we must find new parameters (j, v', w') as before. The target domain face is one of the domain faces in the 1-ring surrounding the current domain face, i.e. those which share at least one vertex with

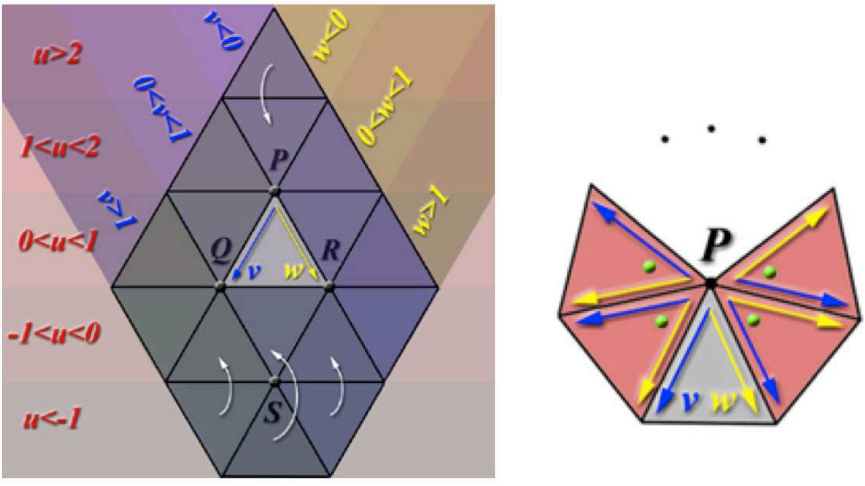

Fig. 4. Identifying parameter values when moving out of the current domain face (triangular case): (a) regular domain face; (b) extraordinary domain face

the current domain face (see Figure 4a). Thus, if $u < -1$ (i.e. $v + w > 2$), we recalculate the values (v, w) using $v = 2v/(v+w), w = 2w/(v+w)$, and if $u > 2$ (i.e. $v + w > -1$), we calculate $v = v/|v + w|, w = w/|v + w|$, so that in each case $-1 \leq u \leq 2$. Let **P** be the parametric point $(0, 0)$ in the current domain face. Regardless of whether **P** is regular or extraordinary, the other two vertices of the current domain face, **Q** and **R**, are both regular. Hence, if $-1 \leq u \leq 1$, we can easily find the target domain face from u, v and w (see Figure 4a) and calculate the new parameter (j, v', w') using the v and w directions in that domain face. For example, if $-1 < u < 0$ and $0 < v, w < 1$, the target domain face is the triangle **QSR**. If $1 < u < 2$, we must consider whether **P** is a regular or extraordinary vertex. If **P** is a regular vertex, we can still find the target domain face easily and calculate the new parameter values. If **P** is extraordinary (see Figure 4b), since $v, w < 0$ and **P** has parameter $(0, 0)$ in all domain faces containing it, we can calculate the distance between (v, w) and **P** in parametric space, $d = \sqrt{\min(1, v^2 + w^2)}$. Next, we compute $v' = w' = \sqrt{d^2/2}$ and evaluate the surface points corresponding to (v', w') for all candidate domain faces. Finally, we select the point nearest to the target point **H** and use it to give the new parameter values (j, v', w').

4 Surface-Surface Intersection Algorithm

As mentioned in Section 2, each patch in the limit surface corresponds to a domain face in the initial mesh. If X denotes the set of all patches for a given subdivision surface, its union gives the limit surface. When a domain face is

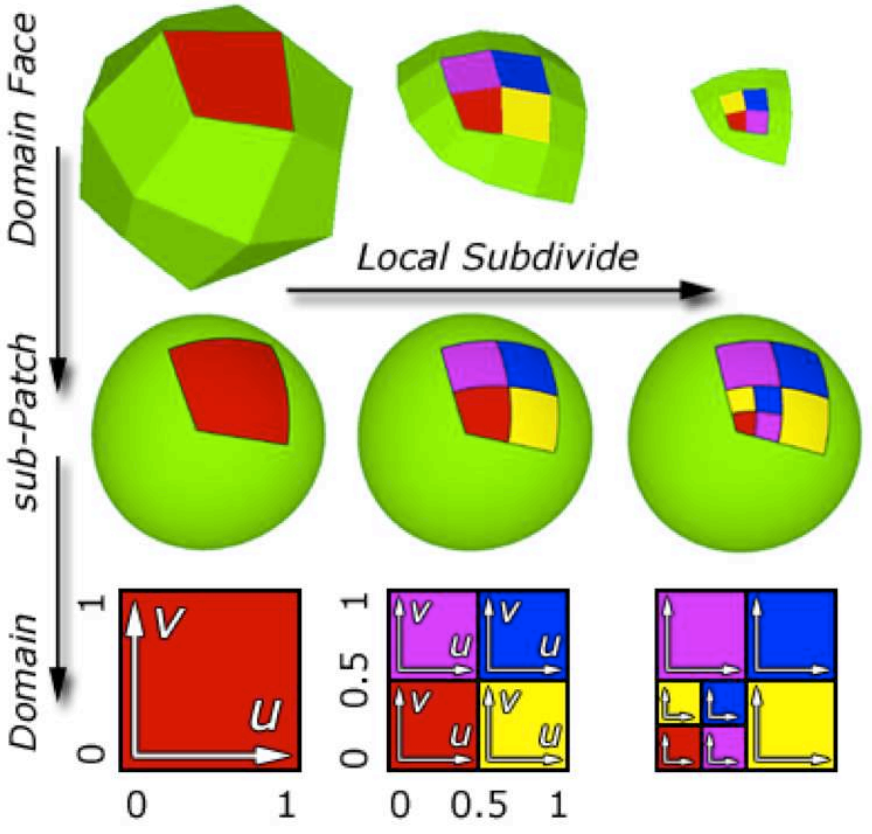

Fig. 5. Subdividing a domain face locally (top row), and splitting its sub-patches accordingly (middle row). The bottom row shows the domains of the sub-patches

subdivided into four sub-faces (using Catmull-Clark or Loop subdivision), the corresponding patch is also split into four sub-patches (see Figure 5). Henceforth, we refer to the elements of X as *sub-patches* (Figure 5, middle row), each having a corresponding sub-face as its domain face (Figure 5, top row), and having a parametric domain that is a subset of its parents parametric domain (Figure 5, bottom row). Additionally, when a face is subdivided locally, in order to be able to continue subdividing its four sub-faces, all the neighbouring faces that share at least a common vertex with those sub-faces (the green faces in the top middle and right of Figure 5) should be computed too.

4.1 Algorithm Overview

We now consider the overall algorithm. It finds starting points for tracing, and then march along the intersection from those starting points.

There are three main steps to this process; in the below, by *interfering*, we mean potentially but not necessarily intersecting:

– Split interfering sub-patches until they are approximately flat, and record all interfering sub-patch pairs.
– Find all pairs of intersecting sub-patches from the interfering pairs, and find an intersection point for each pair.
– Trace all intersection curves from these initial intersection points.

4.2 Convex Hull and Flatness Condition

From the subdivision rules, it can be shown that both Catmull-Clark and Loop surfaces possess the convex hull property, which means that each sub-patch is within the convex hull of its control vertices. A sub-patch has $2N + 8$ control vertices for a quadrilateral mesh and $N + 6$ control vertices in the case of a triangular mesh, where N is the valence of a regular vertex or, if present, of the only extraordinary vertex in the domain face. For simplicity, we use an axis-aligned bounding box (AABB) of the convex hull of each sub-patch as its bounding volume. This is used to detect interference between two sub-patches. If two bounding volumes intersect, we split both sub-patches by subdividing their domain faces, until all sub-patches are considered flat. To measure the flatness of a sub-patch S, we use the variable $f = 1 - \min(\mathbf{N}_0 \cdot \mathbf{N}_i)$, where $\mathbf{N}_0$ is the unit normal of Ss domain face and $\mathbf{N}_i$ is the unit normal of each of the neighbouring 1-ring faces of Ss domain face. (We estimate the normal of a quadrilateral face as the cross product of the two vectors connecting opposite vertices.) When the flatness f is less than a threshold T_f, the sub-patch is considered to be flat.

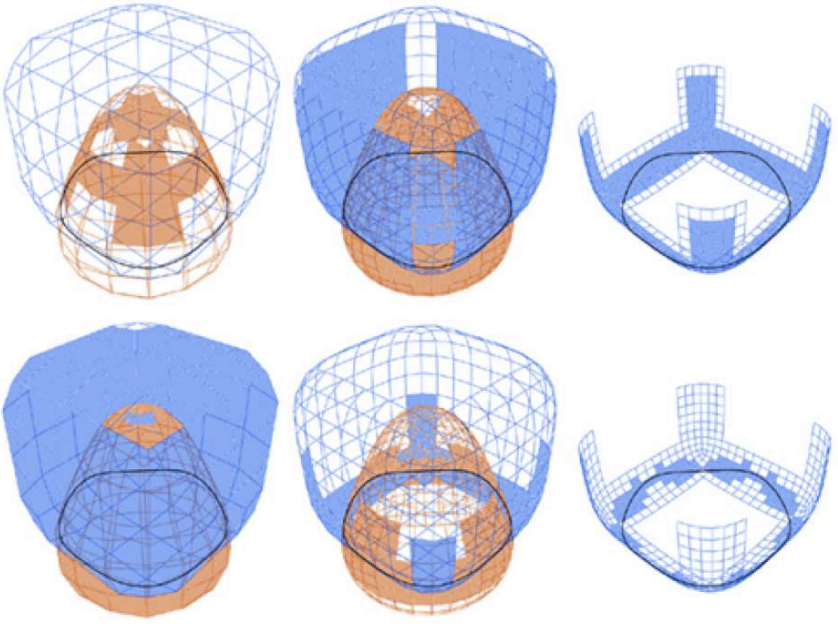

Fig. 6. Flat sub-patches during intersection between a cube and a cone

Additionally, if Ss domain face is quadrilateral, we must first ensure that the domain face is approximately planar by checking if $N_a \cdot N_b < T_f$, where N_a and N_b are the normals of the two triangles comprisong domain face. If the domain face is not planar, we consider the sub-patch not to be flat.

When a sub-patch is sufficiently flat, we take the average parameter values for its parametric domain and evaluate the limit point P_a at the average parameter. Instead of the original bounding volume, we then use the AABB of the convex hull formed by P_a and all the corner points of the sub-patch. We also evaluate the limit normal vector N_a at P_a, to be used as the normal of that sub-patch while finding an actual intersection point, as described in Section 4.3.

By detecting interference and splitting sub-patches, all interfering flat sub-patches are found, as illustrated in Figure 6. This example shows intersection between a cube and a cone. In this case, all interfering sub-patches are flat within tolerance after three levels of subdivision (see Figure 6 from left to right). The top row shows the domain faces of all flat sub-patches, and the bottom row shows the domain faces of all interfering sub-patches.

During this step, for each sub-patch on one surface, we record all the interfering sub-patches from the other surface. Thus, we obtain all pairs of interfering sub-patches. Most of these pairs of interfering sub-patches, however, do not truly intersect. In the next section, we identify those pairs that truly intersect and find an initial intersection point for each pair.

4.3 Intersection Between Two Sub-patches

If the flatness threshold T_f is sufficiently small, we can approximate all sub-patches by polygons and find all intersecting polygon pairs. However, using a small T_f is undesirable, as it leads to more levels of subdivision, incurring considerable cost. Hence, we prefer a patch-patch intersection method.

Inspired by the MAF method, given two sub-patches, we use the average parameters of their respective domains as their initial parameters and iterate the mid-point transformation and sphere transformation to search for an intersection point in their respective domains. If we find an intersection point within a prescribed number of iterations, n, we say that the sub-patch pair intersect. Their intersection point and the sub-patch pair are then added to a set of candidate intersecting pairs. If no intersection point is found after n iterations, it is still possible that the two sub-patches intersect. Such tricky cases are illustrated in Figure 7 for the simpler problem of curve-curve intersection in the plane: if we perform mid-point transformation on the two initial points (blue), both cases will search for the intersection point in the wrong direction and miss the intersection point in the domain of each curve.

To avoid such cases, it would suffice to ensure that for any point of the first curve, it is impossible to find a point in the second curve with parallel normal vector. But for surfaces, the above condition cannot eliminate such cases. Since our purpose is to approximately find initial intersection points for tracing, we simply try to avoid these cases by taking a hint from the following observation: in practice, such cases usually appear when the normal vectors of two sub-patches

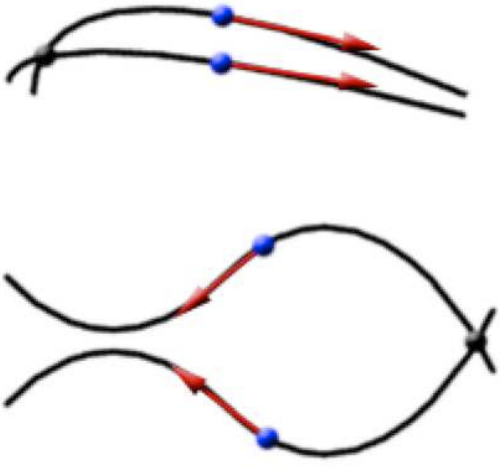

Fig. 7. Two cases which miss the intersection point when doing midpoint transformation

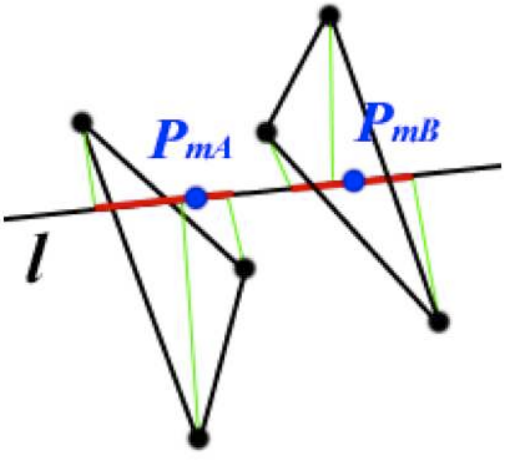

Fig. 8. Interference detection between two sub-patches

are almost parallel. Thus, before checking whether two sub-patches intersect, we estimate if their normals, $\mathbf{N}_{m_1}, \mathbf{N}_{m_2}$ (computed as in Section 4.2), are almost parallel as follows. Let $m = 1 - |\mathbf{N}_{m_1} \cdot \mathbf{N}_{m_2}|$. If $m < f_1 + f_2$, where f_1, f_2 are the flatnesses of the two sub-patches, we say that $\mathbf{N}_{m_1}$ and $\mathbf{N}_{m_2}$ are approximately parallel. If so, we use a tighter bounding volume to more strictly decide interference as follows. Consider two sub-patches A and B. If they are flat, their corner points and two further points P_{m_A}, P_{m_B} would have been calculated as described in Section 4.2. We project all the corner points to the line l connecting P_{m_A} and P_{m_B}, and obtain two maximal intervals on l (shown in red in Figure 8).

If the intervals do not overlap, then A and B do not intersect. Otherwise we split the sub-patch whose domain face has a lower subdivision level and recursively check if the resulting new pairs of sub-patches intersect. If both domain faces has reached the maximum subdivision level, since they are very close to each other and their normal vectors are almost parallel, we conclude that the two subdivision surfaces sharing a region. Perturbation schemes [14] may be used to resolve such degeneracies. For reasons of space, we do not consider this problem further here, and instead focus on presenting the efficient marching method.

We now have all the intersecting sub-patch pairs and an initial intersection point for each. Next, we trace all intersection curves.

4.4 Tracing

We now have all intersecting sub-patch pairs in the candidate set. For each pair, both sub-patches are flat and their normals are not parallel. Hence, we can reasonably assume that each intersecting sub-patch pair has only one intersection curve. Based on this assumption, we present the following tracing algorithm.

First, we randomly select an intersecting sub-patch pair from the candidate set and invoke the MAF procedure with the intersection point of the pair. When the algorithm marches to a new intersection point, it obtains a parameter for each of the two surfaces. For each parameter, we then find the sub-patch whose domain contains that parameter. This new sub-patch pair should also appear in the candidate set. Since there is only one intersection curve for each sub-patch pair, we mark that new sub-patch pair with the index of the current intersection curve and remove it from the candidate set. After tracing a curve, all the sub-patch pairs it has encountered by will have been marked, and thus will not be selected for initiating further curve tracing. If the step size is too large, the tracing curve may skip some pairs. However, tracing from these skipped pairs would result in a situation where most sub-patch pairs on the current curve already belong to another curve; the two curves obviously are identical and thus tracing is terminated. Starting points are selected from the candidate set and each time a corresponding intersection curve is traced. This is repeated until the candidate set is empty.

While we calculate all intersection curves between the two subdivision surfaces, we also record the parameter values of all the intersection points to obtain the pre-images of the intersection curves.

4.5 Parameter Settings

First, we consider the step size δ. We would like $|\delta u|$ and $|\delta v|$ both to be lass than 1 in the sphere transformation step. To achieve this, we estimate the step size according to the scale of the two meshes and adaptively adjust it according to the domain face containing the current tracing point.

In the first step of our algorithm, a sub-patch is considered to be sufficiently flat when its flatness is less than a threshold: $f < T_f$. We find a choice of $T_f = 0.1$ works well in practice. Usually, all sub-patches are found to be flat after two or three levels of subdivision, and for sub-patches with large curvature, six levels are

sufficient. Hence, we conservatively set the maximum subdivision level to be 10. In the second step, if two sub-patches really intersect, in practice we always find an intersection point within three iterations; again to be safe, we set the number of iterations $n = 5$. Since most intersecting pairs do not actually intersect, the average number of iterations used is close to 5.

4.6 Remarks

Since our method is an extension of a known marching method for parametric surfaces to subdivision surfaces, we can handle all special cases that it can also cope with for parametric surfaces. For the same reason, our method faces the same kinds of degeneracy problems as do traditional surface-surface intersection methods.

5 Results

We show some results of calculating intersection curves between subdivision solids using our algorithm. All the examples in Figure 9 were calculated within 10 seconds on a PC with 900 MHZ CPU. More complex examples are shown in Figure 10.

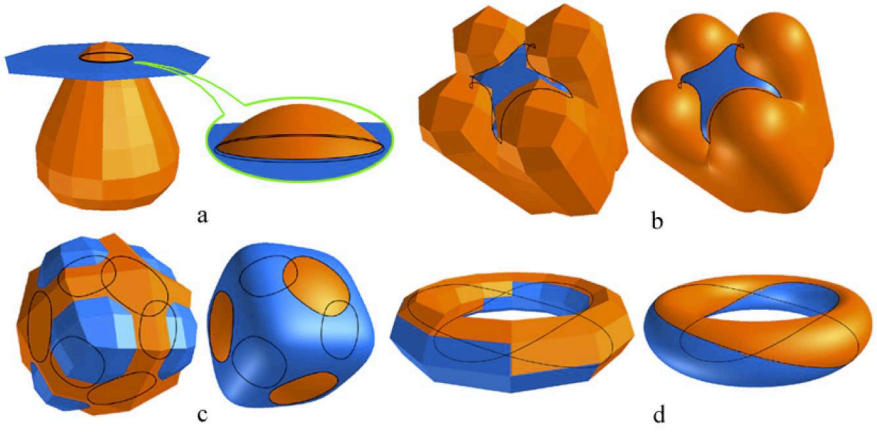

Fig. 9. Intersection curves between two subdivision solids

6 Conclusions

In this paper we have presented an efficient method to calculate intersection curves between two subdivision solids. We continued earlier work and have made a contribution that will extend subdivision surfaces to new applications. Our ultimate intention is to construct complex models from simple primitives using solid modelling operations, and so future works will include:

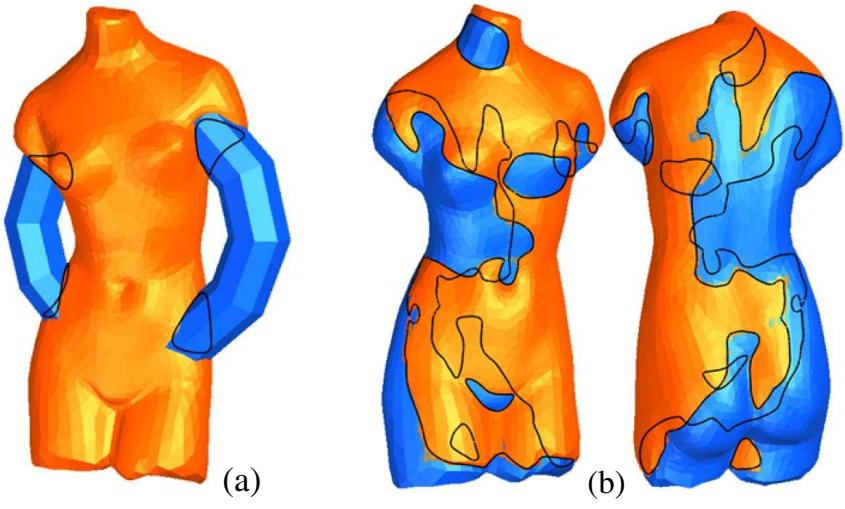

Fig. 10. (a) Intersecting a Venus (Loop surface) and a torus (Catmull-Clark) surface; (b) intersecting two Venuses (Venus model courtesy of the NYU Media Research Lab)

- Designing more robust algorithms that can better handle degeneracies and numerical instabilities.
- Extending our method to *piecewise* smooth subdivision solids using Zorin's work as a basis [18].
- Developing a CAD system based on subdivision and CSG representation by integrating a wide range of geometric computing algorithms, such as offseting, blending, trimming and Boolean operations.

Acknowledgements

This work was supported by the Natural Science Foundation of China (Project numbers 60225016, 60273012, 60333010).

References

1. R. E. Barnhill, G. Farin, M. Jordan, B. R. Piper (1987) Surface/surface intersection. *Computer Aided Geometric Design*, 4, 3–6
2. R. E. Barnhill, S. N. Kersey (1990) A marching method for parametric surface/surface intersection. *Computer Aided Geometric Design*, 17, 257–280
3. H. Biermann, D. Kristjansson, D. Zorin (2001) Approximate Boolean operations for subdivision surfaces. *Proc. SIGGRAPH 2001*, 185–194
4. E. Catmull, J. Clark (1978) Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10 (6), 350-?55
5. D. Doo, M. Sabin (1978) Analysis of the behavior of recursive division surfaces near extraordinary points, *Computer Aided Design*, 10 (6), 257–268

6. E. Grinspun, P. Schröder (2001) Normal bounds for subdivision-surface interference detection. *Proc. IEEE Scientific Visualization*
7. M. Hohmeyer (1991) A surface intersection algorithm based on loop detection. *International Journal of Computational Geometry and Applications*, 1 (4), 473–490
8. S. M. Hu, J. G. Sun, T. G. Jin, G. Z. Wang (2000) Computing the parameters of points on NURBS curves and surfaces via moving affine frame method. *Journal of Software*, 11 (1), 49–53
9. L. Kobbelt (2000) $\sqrt{3}$ subdivision. *SIGGRAPH 2000 proceedings*, 103–112
10. S. Krishnan, D. Manocha (1997) An efficient surface intersection algorithm based on lower dimensional formulation. *ACM transactions on Graphics* 16 (1)
11. N. Litke, A. Levin, P. Schröder (2000) *Trimming for Subdivision Surfaces*. Technical report, Caltech
12. C. T. Loop (1987) *Smooth Subdivision Surfaces Based on Triangles*. M.S. Thesis, Department of Mathematics, University of Utah
13. A. H. Nasri (1987) Polyhedral Subdivision Methods for Free-form Surfaces *ACM Trans. Graphics* 6 (1), 29–73
14. R. Seidel (1998) The nature and meaning of perturbations in geometric computing. *Discrete and Computational Geometry*, 19 (1), 1–17
15. J. Stam (1998) Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. *Proc. SIGGRAPH 1998*, 395–404
16. J. Stam. (1999) Evaluation of Loop subdivision surfaces. *SIGGRAPH'99 Course Notes*
17. D. Zorin, P. Schröder (2000) Subdivision for modeling and animation, *SIGGRAPH 2000 Course Notes*
18. D. Zorin, D. Kristjansson (2001) Evaluation of piecewise smooth subdivision surfaces. *Visual Computer*