공학박사학위논문

Surface Intersections with Bounding Volume Hierarchy and Osculating Toroidal Patches

BVH와 토러스 패치를 이용한 곡면 교차곡선 연산

2021년 8월

서울대학교 대학원

컴퓨터공학부

박 영 진

Surface Intersections with Bounding Volume Hierarchy and Osculating Toroidal Patches

BVH와 토러스 패치를 이용한 곡면 교차곡선 연산

지도교수 김명수

이 논문을 공학박사 학위논문으로 제출함

2021년 5월

서울대학교 대학원

컴퓨터공학부

박 영 진

박영진의 공학박사 학위논문을 인준함

2021년 7월

위 원 장 _____ 신 영 길

부위원장 _____ 김 명 수

위 원 _____ 이 제 희

위 원 _____ 경 민 호

위 원 _____ 윤 승 현

Abstract

We present a new approach to the development of efficient and stable algorithms for intersecting freeform surfaces, including the surface-surface-intersection and the surface self-intersection of bivariate rational B-spline surfaces. Our new approach is based on a hybrid Bounding Volume Hierarchy(BVH) that stores osculating toroidal patches in the leaf nodes. The BVH structure accelerates the geometric search for the potential pairs of local surface patches that may intersect or self-intersect. Osculating toroidal patches have second-order contact with C^2-continuous freeform surfaces that they approximate, which plays an essential role in improving the precision of various geometric operations on the given surfaces.

To support efficient computation of the surface-surface-intersection curve, we design a hybrid binary BVH that is basically a pre-built Rectangle-Swept Sphere(RSS) tree enhanced with osculating toroidal patches in their leaf nodes. Osculating toroidal patches provide efficient and robust solutions to the problem even in the non-trivial cases of handling two freeform surfaces intersecting almost tangentially everywhere.

The surface self-intersection problem is considerably more difficult than computing the intersection of two different surfaces, mainly due to

the existence of miter points. A self-intersecting surface changes its normal direction dramatically around miter points, located at the open endpoints of the self-intersection curve. This undesirable behavior causes serious problems in the stability of geometric algorithms on self-intersecting surfaces. To facilitate surface self-intersection computation with a stable detection of miter points, we propose a ternary tree structure for the hybrid BVH of freeform surfaces. In particular, we propose a special representation of miter points using sufficiently small quadrangles in the parameter domain of bivariate surfaces and expand ideas to offset surfaces.

We demonstrate the effectiveness of the proposed new approach using some highly non-trivial examples of freeform surfaces with tangential intersections and miter points. In all the test examples, the closeness of geometric entities is measured under the Hausdorff distance upper bound.

Keywords: Surface Surface Intersection, Surface Self Intersection, Bounding Volume Hierarchy, Rectangle Swept Sphere, Osculating Toroidal Patches, Miter Point

Student Number: 2016-21202

Contents

List of Figures

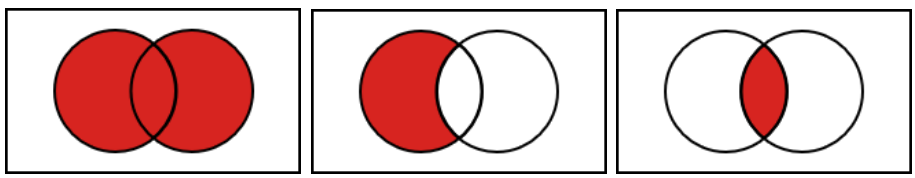
List of Tables

Chapter 1

Introduction

1.1 Background

Boolean operations have been used in many science and engineering disciplines such as set theory, electronic circuits, programming languages, search engines(e.g. Google search), geometry, solid modeling, etc. Set theory is one of the most fundamental areas in mathematics, where Boolean



(a) Union **(b)** Difference **(c)** Intersection

Figure 1.1: Venn diagram of Boolean operations.

operations such as union($A \cup B$), difference($A - B$), and intersection($A \cap B$) often define more complex and structured sets than A and B. Logical circuits are designed by combining Boolean operators with electric signals, and the circuits are the basic components for smartphones, computers, and many other electronic devices. In search engines, when we want to search for 'Beethoven Symphony' but do not want to listen to Karajan's direction, we type in 'Beethoven Symphony − Karajan' to the Google engine. This expression can be translated to the following Boolean operation: 'Beethoven' AND 'Symphony' NOT 'Karajan'.

Back to the set theory, Venn diagrams are used to represent the Boolean operations on sets intuitively. In Figure 1.1, there are Venn diagrams for Boolean operations. In order to illustrate a Boolean operation using a Venn diagram, it is necessary to delineate the boundary curve of the region corresponding to the Boolean operation.

In geometric and solid modeling, Boolean operations do the same roles in other areas. Given two solid models, Boolean operations combine the two models(union), subtract one from the other(difference), and create a new overlapping model between the two models(intersection) [4]. Boolean operations thus support the construction of complex solid models starting from simple ones. Typical commercial CAD softwares provide Boolean operations. For example, in AutoCAD, SolidWorks, Rhino, Shapr3D, and game engines such as Unreal Engine and Unity, one can find some forms of

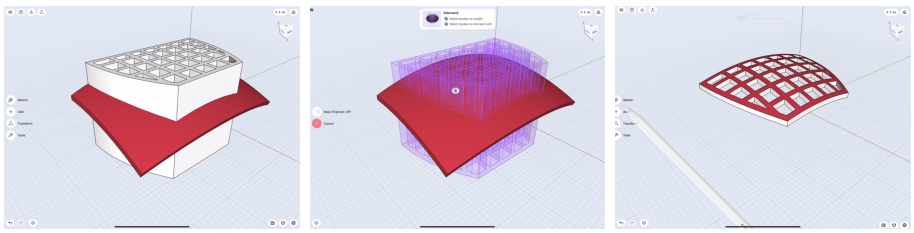


Figure 1.2: Boolean operation in Shapr3D

Boolean operations in their basic facilities. In order to perform Boolean operations, these software systems compute the boundary surface for the model corresponding to the Boolean operations under consideration, which is implemented by computing the intersection curve between the boundary surfaces of the two models under consideration. There are two major representation schemata used in solid modeling: Constructive Solid Geometry(CSG) and Boundary-representation(B-rep) [5].

In Constructive Solid Geometry(CSG), a solid model is represented by an expression tree of geometric transformations and Boolean operations on primitive objects, typically including cylinders, cones, spheres, tori, blocks, etc. In Figure 1.3, the top model is the result of Boolean operations on these primitives. CSG has long been used since the early days of CAD/CAM systems. There are many good reasons for the popularity: Using less memory, it is simple, valid, and easy to modify [6]. For Boolean operations for CSG models, the operation has to find the intersection curve(s) among the primitives and then leave or remove some

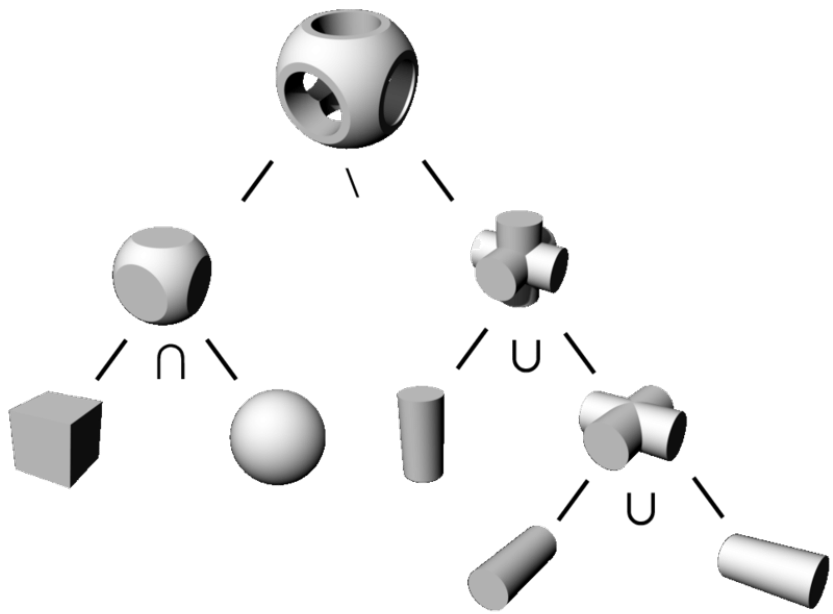
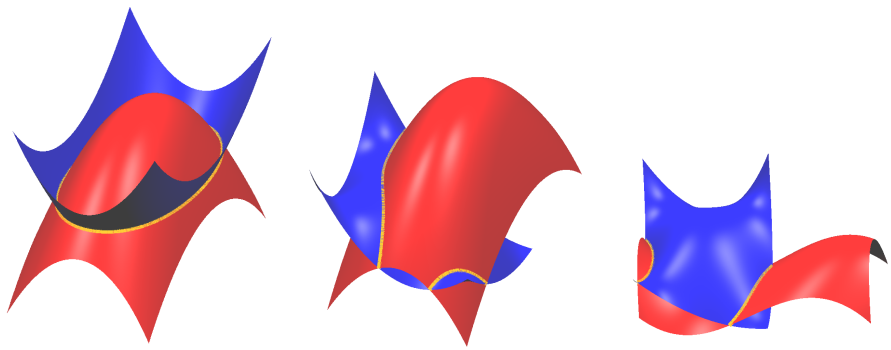


Figure 1.3: A CSG tree example.

fragments of trimmed surface patches that fit to each Boolean operation. Since simple equations are used in representing primitives, it is relatively easy to construct the intersection curves of primitives. However, there are certain limitations. CSG models rely heavily on the primitives first chosen. Design limitations also exist depending on the primitives offered by the modeling system. CSG also has limitations in representing various different solid models with pure CSG operations only. Boundary-representation is often used in parallel to support *hybrid representations* in recent CAD/CAM systems.



(a) **(b)** **(c)**

Figure 1.4: (a)-(c) Intersection curve of two freeform surfaces.

In Boundary representation(B-rep), a solid model is designed with a set of freeform surfaces surrounding the model. It is more complex, while consuming more memory space. Moreover, it is difficult to guarantee the validity of model. On the other hand, it can represent general solid models of various different complexities. For Boolean operations for two general B-rep models, we need to compute the intersection curves of general freeform surfaces in a highly stable way, which is the main goal of this thesis work.

1.2 Surface-Surface-Intersection

The problem of intersecting two freeform surfaces reliably is one of the main technical challenges in geometric and solid modeling. [5, 7, 8, 9]. Unfortunately, in the last decade, there have been only a few publications

in this fundamental research area [10, 11], because of its difficulties. For example, according to Bèzout's Theorem, the degree of intersection curve of two bicubic freeform surfaces is more than 300 [5, 12].

On the other hand, the problem of intersecting two surfaces appears in many different forms as the solution of non-linear geometric constraints [13, 14, 15], where the constraints are often represented or automatically generated as rational freeform curves, surfaces, and multivariate volumes. Consequently, it is still extremely important to develop new techniques that can handle the surface-surface-intersection (SSI) problem efficiently and reliably, even if input surface is highly non-trivial case (for example, intersect tangentially almost everywhere).

A natural question arises: what makes revisiting this problem worthwhile for some meaningful technical advancements? To answer this question in a positive perspective, we would like to mention the recent developments of spatial data structures (such as BVH) for freeform curves and surfaces, which have made possible the acceleration of many important geometric algorithms, including collision detection, the minimum and Hausdorff distance computations, offset curve trimming, Minkowski sum computation, medial axis and Voronoi diagram construction, convex hull computation, etc [16, 17, 18, 19, 20].

In Chapter 5, we present an efficient and robust algorithm for surface-surface-intersection of freeform surfaces, using a BVH with osculating

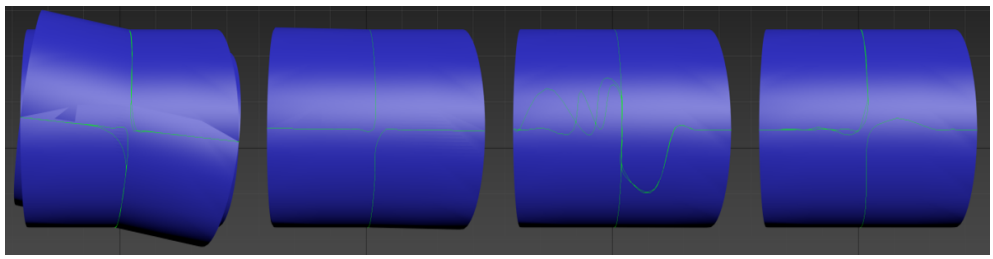
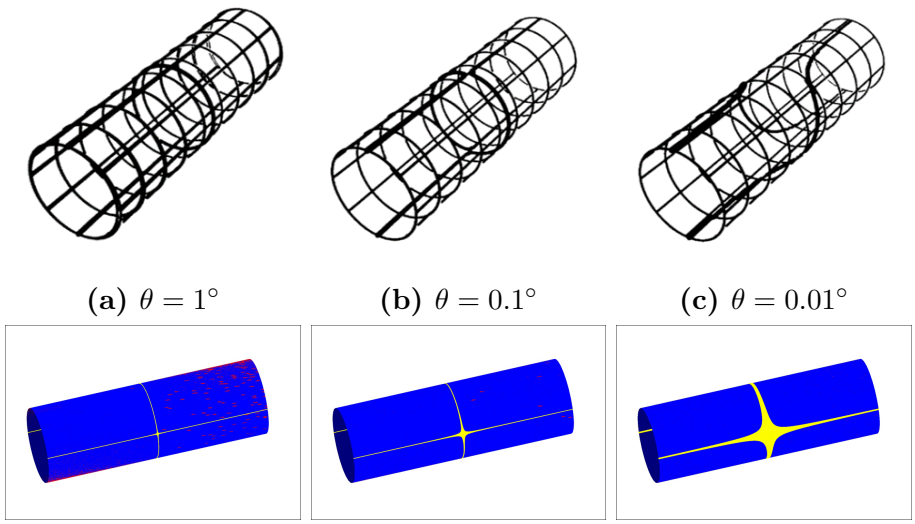


Figure 1.5: Intersection examples of two cylinders in 3ds Max. The rotation angle is 10°, 1°, 0.1°, and 0.01°, respectively.

toroidal patches in their leaf nodes. The internal nodes contain rectangle-swept spheres as their bounding volume, and the leaf nodes contain osculating toroidal patches. The bounding volume of the internal nodes accelerates the geometric search for the potential pairs of surface patches that may generate some curve segments in the SSI, and the osculating toroidal patches provide reasonable approximate solutions to the SSI problem thanks to the higher approximation order [21].

We demonstrate the effectiveness and robustness of our new approach, by using some highly non-trivial examples of freeform surfaces, which intersect tangentially almost everywhere. In Figure 1.6, two almost identical cylinders are intersected, one cylinder is rotated version of the other cylinder with angle θ. In Figure 1.6(a)–(c), with Heo et al. [1] method, the X-junctions of intersection curve are slightly missed at a small angle $\theta = 0.1°$, and then completely missed at a smaller angle $\theta = 0.01°$. In Figure 1.5, Even the latest version of Autodesk 3ds Max does not



(a) $\theta = 1°$ **(b)** $\theta = 0.1°$ **(c)** $\theta = 0.01°$

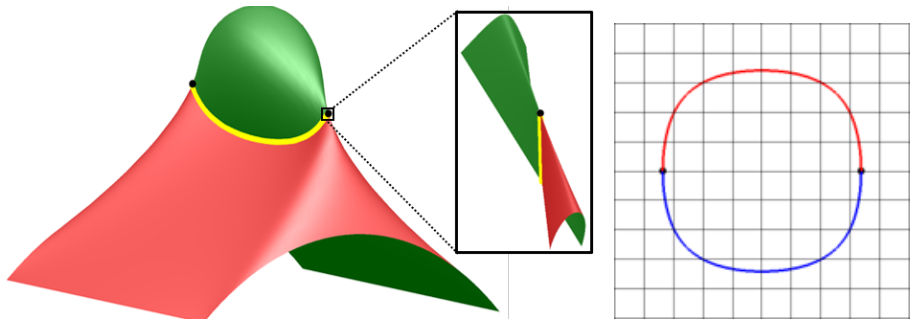
(d) $\theta = 0.01°$ **(e)** $\theta = 0.001°$ **(f)** $\theta = 0.0001°$

Figure 1.6: Tangential intersection of two almost identical cylinders: (a)–(c) from Heo et al. [1], (d)–(f) from our approach.

compute the intersection curve properly. On the other hand, in Figure 1.6(d)–(f), our new approach can detect x-junctions correctly, even the rotation angle is much smaller.

1.3 Surface Self-Intersection

In the majority of previous algorithms for SSI, the input surfaces are usually assumed to be self-intersection-free. However, there are some cases where we need to consider the possibility of self-intersecting input surfaces, particularly when the surfaces are given as offset or sweep



(a) **(b)**

Figure 1.7: An example of surface with self-intersection curve and miter points: (a) in the Euclidean xyz-space, (b) in the (u, v)-parameter domain.

surfaces [22, 23]. The self-intersection of a freeform surface $S(u, v)$ is formally defined as the following set:

$$\mathcal{SI} = \{S(u, v) \mid S(u, v) = S(s, t),\ (u, v) \neq (s, t)\}\,, \tag{1.1}$$

which contains all surface locations $S(u, v)$ shared with some other-parameter points $S(s, t)$ of the same surface. The required condition for different parameters, $(u, v) \neq (s, t)$, makes the self-intersection problem considerably more difficult than the usual case of intersecting two non-identical surfaces. The difficulty is mainly due to the existence of *miter points*, in the neighborhood of which the two different parameters can be arbitrarily close to each other [2, 24, 25].

For example, Galligo and Pavone [2] constructed the self-intersection

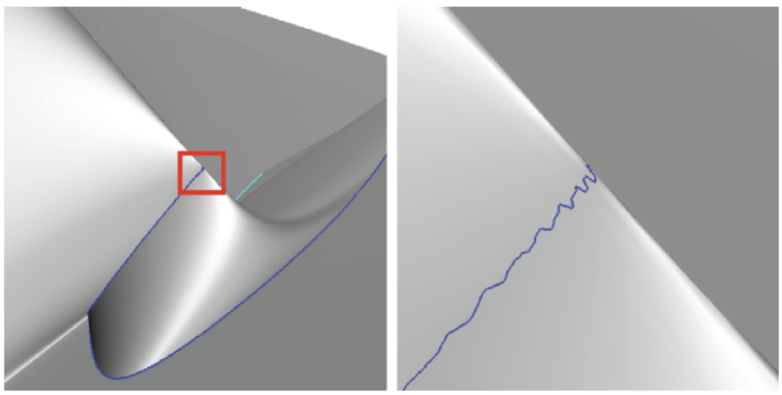


Figure 1.8: An example taken from Galligo and Pavone [2]. There are serious robustness issues near the miter points.

curve using a triangular mesh approximation to the freeform surface. As shown in Figure 1.8, the curve approximation near a miter point is highly unstable, producing a curve approaching a miter point in a zigzag fashion.

Figure 1.7(a) shows an example of two miter points (in black) on the self-intersection curve of a freeform surface. They appear (in this example) as the terminal endpoints of the self-intersection curve. In Figure 1.7(b), the corresponding points in the parameter domain are shown (in black) as the common endpoints of two curve segments (in red and blue). These two (u, v) and (s, t)-parameter curves are the solution curves for the surface self-intersection problem: $S(u, v) = S(s, t)$. When we glue the two parameter curves together, according to the same-location condition: $S(u, v) = S(s, t)$, the final result will be the self-intersection curve

in the Euclidean xyz-space. Moreover, the folding endpoints (in the glue operation) correspond to the two miter points.

In another physical analogy, we assume a walking along the loop composed of the red and blue solution curves. When we walk along the blue (or red) curve counterclockwise in the parameter space, the self-intersection curve in the xyz-space is traced in the rightward (or leftward) direction. In the right half of the loop, walking from the blue curve below to the red one above passing through the black dot on the right, the corresponding tracing on the self-intersection curve (in the xyz-space) approaches to the miter point on the right in a slower and slower speed, finally stops at the miter point momentarily, and then suddenly flips to the opposite direction and moves away from the miter point gradually speeding up the tracing. The miter points are thus non-regular on a freeform surface [26].

In Chapter 6, we present an efficient and robust algorithm for surface self-intersection of freeform surfaces with a stable detection of miter points. We use a ternary hierarchy structure for BVH, make a self-intersection problem to a rather conventional problem of intersecting two non-adjacent subpatches of the same surface $S(u, v)$, the solution of which is known to be relatively stable. We take a practical approach to the detection of miter point locations by osculating toroidal patches, and bound the miter point with small quadrangle in the parameter domain.

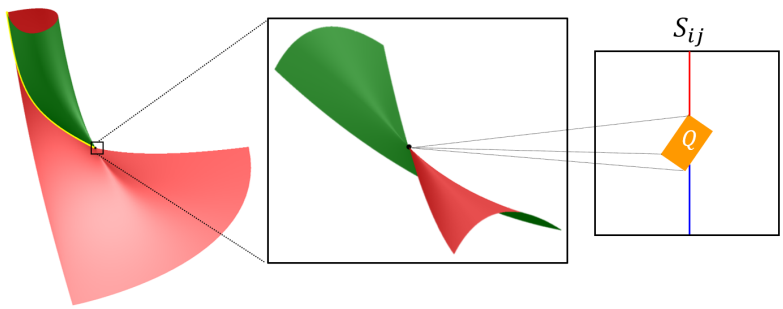


Figure 1.9: The quadrangle Q is mapped into a very tiny surface patch in the Euclidean XYZ-space. It is totally contained in an ϵ-ball.

Then the exact location of the miter point is bounded by a tiny ϵ-ball in the Euclidean space. Using the ϵ-ball, we can control the influence of miter points within certain error bounds.

1.4 Main Contribution

The main contributions of our approach to the surface-surface-intersection problem can be summarized as follows:

- We propose a new BVH-based algorithm for computing the SSI curves for freeform surfaces, which outperforms other conventional algorithms in computing speed and robustness by computing the BVH structure in a preprocessing time.

- The improvement is based on the high approximation order of osculating toroidal patches to the freeform surfaces and the geometric

simplicity of toroidal patches in supporting primitive operations.

- The main advantage of our SSI algorithm is in handling the degenerate case of (almost) tangential surface intersections, where the conventional subdivision methods have difficulty pruning a large number of potentially overlapping pairs of small surface patches.

The main contributions to the surface self-intersection problem can be summarized as follows:

- We propose a new representation scheme for miter points using small quadrangles (in the parameter domain), each of which can be mapped to a tiny surface patch (in the Euclidean xyz-space) totally contained in an ϵ-ball, where the approximation error bound $\epsilon > 0$ is typically taken as $10^{-5}, 10^{-6}$, depending on each application.

- Using osculating toroidal patches, we develop a stable method for detecting and bounding miter points, where the degeneracy of toroidal patches is used as a signal for the existence of miter points.

- We modify the conventional BVH structures so that the leaf nodes containing miter points are represented and processed properly. This modification is also an important step for the extension of many other geometric algorithms so that they can handle input surfaces that may self-intersect.

- In Section 6.3, we solve a non-trivial self-intersection problem between a small neighborhood of miter point and other parts of the surface, by converting it to a simple line-surface intersection. This is based on an observation that the self-intersection curve has an almost linear shape near miter points.

1.5 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 presents some preliminary material on differential geometry, Bézier representation, surface approximation and torus. Chapter 3 reviews related previous work on surface-surface-intersection and surface-self-intersection. Chapter 4 presents a Bounding Volume Hierarchy for freeform surfaces, a binary tree structure for the surface-surface-intersection problem, and a ternary tree structure for the surface self-intersection problem. Chapter 5 presents a surface-surface-intersection algorithm with the hybrid BVH using rectangle swept-spheres in the nodes and osculating toroidal patches in their leaf nodes. Chapter 6 presents a surface self-intersection algorithm based on a regional representation scheme for miter points and Chapter 7 shows our surface self-intersection algorithm can be applied to offset surface. Finally, Chapter 8 presents experimental results and Chapter 9 concludes this thesis.

Chapter 2

Preliminaries

2.1 Differential geometry of surfaces

In this thesis, the algorithm to be described later can be applied to any tensor product surfaces to an arbitrary degree. However, for the convenience of representation, we only consider Bézier surfaces as an original surface. Therefore, we briefly review the differential geometry of surfaces, Bézier curve, Bézier surfaces, and torus in the following sections. For more details, please see [26, 27, 28, 29]. A parametric surface is a surface defined by a parametric equation with (usually) two parameters, $S : U \to \mathbb{R}^3$ such that

$$S(u, v) = (x(u, v), y(u, v), z(u, v)), \tag{2.1}$$

where U is a subset of $\mathbb{R}^2$. When tangent planes exist for all points on a surface, or satisfy the following constraints for all $(u, v) \in U$:

$$S_u(u, v) \times S_v(u, v) \neq 0. \tag{2.2}$$

A unit normal vector to the parametrized surface at a regular point is defined as follows:

$$\mathbf{N} = \frac{S_u(u, v) \times S_v(u, v)}{|S_u(u, v) \times S_v(u, v)|} \neq 0. \tag{2.3}$$

The Gauss map of $S(u, v)$ is defined by the map $N : S \to S^2$, where S^2 is a unit sphere.

The first fundamental form of regular surface are defined as follows:

$$E = \langle S_u(u, v), S_u(u, v) \rangle \tag{2.4}$$

$$F = \langle S_u(u, v), S_v(u, v) \rangle \tag{2.5}$$

$$G = \langle S_v(u, v), S_v(u, v) \rangle , \tag{2.6}$$

where $\langle \cdot, \cdot \rangle$ is a dot product. The second fundamental form of regular surface are defined as follows:

$$e = \langle S_{uu}(u, v), \mathbf{N} \rangle \tag{2.7}$$

$$f = \langle S_{uv}(u,v), \mathbf{N} \rangle \tag{2.8}$$

$$g = \langle S_{vv}(u,v), \mathbf{N} \rangle . \tag{2.9}$$

The normal curvature κ_n of surface $S(u,v)$ for arbitrary direction $v = aS_u + bS_v$, $a^2 + b^2 = 1$ is defined as follows:

$$\kappa_n = \frac{ea^2 + 2fab + gb^2}{Ea^2 + 2Fab + Gb^2}. \tag{2.10}$$

The minimum value κ_1 and the maximum values κ_2 of the normal curvature κ_n are the principal curvature, and the direction v of the principal curvature is the principal direction$\mathbf{e_1}$ and $\mathbf{e_2}$. The Gaussian curvature K and the mean curvature H of $S(u,v)$ are defined as follows:

$$K = \frac{eg - f^2}{EG - F^2} = \kappa_1 \kappa_2 \tag{2.11}$$

$$H = \frac{eG - 2fF + gE}{2(EG - f^2)} = \frac{\kappa_1 + \kappa_2}{2}. \tag{2.12}$$

2.2 Bézier curves and surfaces

A *Bernstein basis polynomial* is a polynomial function defined as follows:

$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} t^i. \tag{2.13}$$

A bézier curve of degree n, parametrized by u in $[0, 1]$ is defined as follows:

$$C(u) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(u), \tag{2.14}$$

where $\mathbf{b}_i$ are control points of the curve, form the *Bézier polygon* of the curve. The k-th order derivative of a Bézier curve is defined as follows:

$$\frac{d^k c(u)}{du^k} = \frac{n!}{(n-k)!} \sum_{i=0}^{n-k} \Delta^k \mathbf{b}_i B_i^{n-k}(u), \tag{2.15}$$

where $\Delta^k \mathbf{b}_i = \Delta^{k-1} \mathbf{b}_{i+1} - \Delta^{k-1} \mathbf{b}_i$.

Bézier curves are used in the most CAD/CAM systems, because of following properties.

- *Endpoint interpolation*: The start and end points of the curve are the same as the start and end points of the control point. For example, in a cubic Bézier curve, $c(0) = \mathbf{b}_0$ and $c(1) = \mathbf{b}_3$.

- *Symmetry*: The two control points pairs, $\mathbf{b}_0 ... \mathbf{b}_n$ and $\mathbf{b}_n .. \mathbf{b}_0$ make same Bézier curve, but the direction of movement according to parameter is the opposite.

- *Convex hull property*: Every point on the curve belongs to the convex hull of control points of the curve.

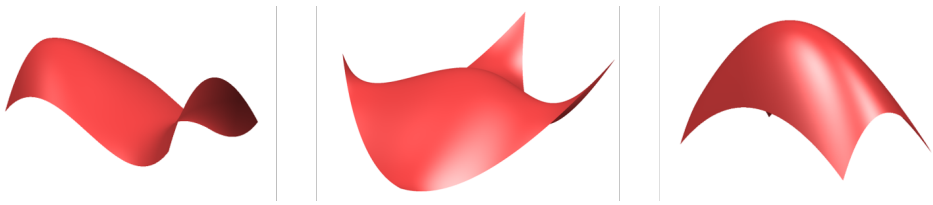


Figure 2.1: Example of Bézier Surfaces

- *Invariance under affine maps*: If an affine map is applied to the control points of the curve, the curve is mapped by the same map.

A Bézier surface is a A natural dimension-extension of a Bézier curve. A Bézier surface of degree $m \times n$, parametrized by u and v in $[0,1] \times [0,1]$ is defined as follows.

$$S(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \mathbf{b}_{i,j} B_i^m(u) B_j^n(v), \tag{2.16}$$

where $\mathbf{b}_{i,j}$ are control points of the surface, $B_i^m(u)$ are Bernstein basis polynomials of degree m, and $B_j^n(v)$ are Bernstein basis polynomials of degree n.

2.3 Surface approximation

Taking uniform samples along each parameter direction, $S\left(\frac{i}{2^H}, \frac{j}{2^H}\right)$, where $i, j = 0, \cdots, 2^H$, the BVH structure for a bicubic Bézier surface $S(u,v)$, $(0 \leq u, v \leq 1)$, can be represented in about the same way as that for the quadtree (of height H) for regular quad meshes (Figure 2.2). The

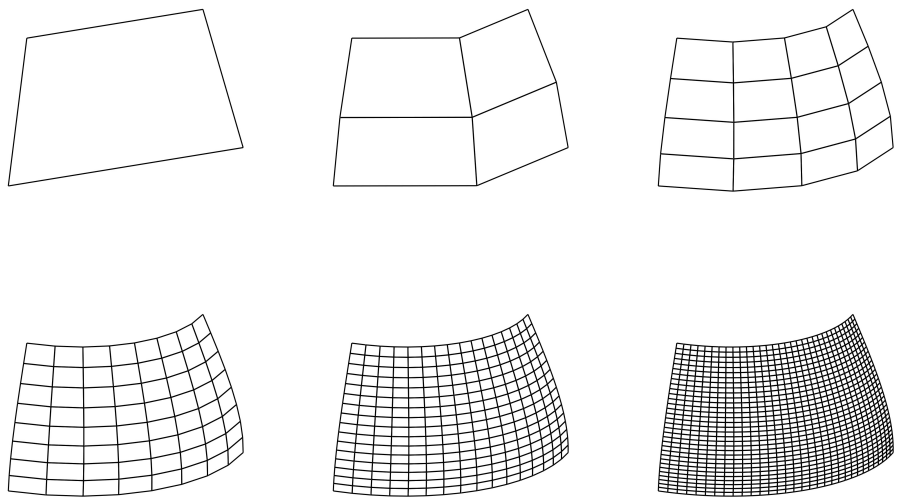


Figure 2.2: Bézier surface approximation using a hierarchy of quad meshes

main difference is in the internal nodes containing bounding volumes slightly thicker than those for a regular quad mesh. The extra thickness is needed for bounding the difference between a surface $S(u, v)$ and the quad mesh approximation (interpreted as a rectangular array of bilinear surface patches or pairs of triangles).

The surface approximation error can be bounded using a simple formula developed by Filip et al. [28]. (Krishnamurthy et al. [30, 20] also use this formula to construct various BVH trees for freeform surfaces.) Using a 2D array of size $(1 + 2^H) \times (1 + 2^H)$, where the x, y, z coordinates of $S\left(\frac{i}{2^H}, \frac{j}{2^H}\right)$ are stored in each element, we can encode the BVH

structure of the surface $S(u, v)$ in a compact way, where the thickness of the internal nodes at a level h ($\leq H$) can be estimated by the formula of Filip et al. [28]:

$$\|S_{ij}(u, v) - Q_{ij}(u, v)\| \leq \frac{\sqrt{3}\left(\|S_{uu}\|_\infty + 2\|S_{uv}\|_\infty + \|S_{vv}\|_\infty\right)}{8} = \hat{\epsilon}_{ij},$$

where $\|\mathbf{v}\|_\infty = \max\{|v_x|, |v_y|, |v_z|\}$ for a vector $\mathbf{v} = (v_x, v_y, v_z)$. (This is because the higher level h approximates the surface $S(u, v)$ using a coarser quad mesh with vertices taken at $S\left(\frac{i}{2^h}, \frac{j}{2^h}\right)$, where $i, j = 0, \cdots, 2^h$.

2.4 Torus

Torus is a surface created by rotating a circle of radius r on an axis of a straight line on the plane including the circle and at a distance $R > r$ away from the center of the circle [26]. The torus $T(u, v)$ parametrized by u and v in $[0, 2\pi) \times [0, 2\pi)$ is defined as follows:

$$T(u, v) = ((R + r\cos v)\cos u, (R + r\cos v)\sin u, r\sin v), \qquad (2.17)$$

where R is a major radius, and r is a minor radius of the torus. An implicit equation of the torus is

$$(\sqrt{x^2+y^2}-R)^2+z^2=r^2. \tag{2.18}$$

The partial derivatives and the second order partial derivatives of torus are defined as follows:

$$T_u(u,v)=(-r\cos u\sin v,-r\sin u\sin v,r\cos v) \tag{2.19}$$

$$T_v(u,v)=(-(R+r\cos v)\sin u,(R+r\cos v)\cos u,0) \tag{2.20}$$

$$T_{uu}(u,v)=(-r\cos u\cos v,-r\sin u\cos v,-r\sin v) \tag{2.21}$$

$$T_{uv}(u,v)=(r\sin u\sin v,-r\cos u\sin v,0) \tag{2.22}$$

$$T_{vv}(u,v)=(-(R+r\cos v)\cos u,-(R+r\cos v)\sin u,0). \tag{2.23}$$

From these, we can obtain the first fundamental form E,F,G, the second

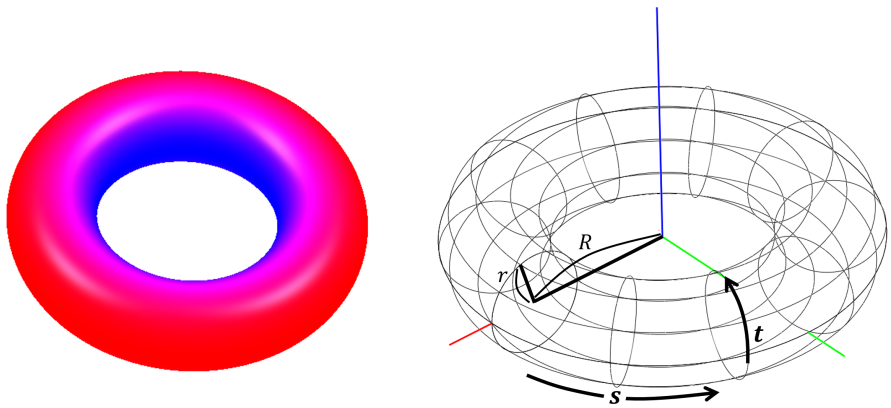


Figure 2.3: Torus with point $T(0,0)$ and point $T(0,\pi)$

fundamental form e, f, g, and the Gaussian curvature K of a torus [26]:

$$E = \langle T_u, T_u \rangle = r^2 \tag{2.24}$$

$$F = \langle T_u, T_v \rangle = 0 \tag{2.25}$$

$$G = \langle T_v, T_v \rangle = (R + r\cos v)^2 \tag{2.26}$$

$$e = \langle N, T_{uu} \rangle = r \tag{2.27}$$

$$f = \langle N, T_{uv} \rangle = 0 \tag{2.28}$$

$$g = \langle N, T_{vv} \rangle = \cos v(R + r\cos v) \tag{2.29}$$

$$K = (eg - f^2)/(EG - F^2) = \frac{\cos v}{r(R + r\cos v)} \tag{2.30}$$

where $\langle \cdot, \cdot \rangle$ is a dot product and $\mathbf{N}$ is the normal vector of the torus.

A Torus is a regular surface, and torus has a full range of Gaussian

curvature, not only positive and negative, but also zero curvature (along the parallels $v = \frac{\pi}{2}$ and $v = \frac{3\pi}{2}$). In Figure 2.3, the positive curvature of the torus is shown in red, the negative curvature in blue, and zero in purple.

2.5 Summary

In this chapter, we briefly introduce differential geometry of regular surfaces, Bèzier curve, and surface, which is used as an input of our algorithm. We also introduce the surface approximation method based on uniform sampling and the differential geometry of torus. With the surface approximation method and toroidal patches, we can perform geometric operations fast and robust. We introduce our new surface intersection algorithms with BVH and osculating toroidal patches on their leaf nodes in the following chapters, and show that it solves the problem previously considered impossible.

Chapter 3

Previous Work

3.1 Surface-Surface-Intersection

A comprehensive introduction to the problem of intersecting freeform curves and surfaces can be found in the Chapter 12 of Hoschek and Lasser [7], where early methods are explained in great details, including algebraic, subdivision, embedding, discretization, and tracing methods. In a survey article (published in 1993), Patrikalakis [31] reviews the SSI algorithms developed in the period of 1988–1992, which are then classified into four main categories: analytic, lattice evaluation, marching, and subdivision methods. Farin [32] compiled an SSI bibliography of 50 references, containing algorithms developed in the period of 1968–1990. On the other hand, Patrikalakis and Maekawa [8, 33] introduce an extensive

body of SSI literature (with more than 100 references), including new results developed in the late 90's, in particular, some techniques that can deal with the robustness issues using interval arithmetics. Since then, there have been relatively few publications in the SSI research. New results are often those for special types of surfaces such as torus [10, 11] or sweep surfaces [34, 22], or algorithms [35, 36] dealing with special cases that had been overlooked in the previous work.

The GPU-based SSI approach of Krishnamurthy et al. [30] generates the BVH structures for freeform surfaces (partially and dynamically on the fly). Because of the non-flexibility of GPU implementation, Krishnamurthy et al. [30] constructed only AABB (Axis-Aligned Bounding Box) trees; nevertheless, their basic approach can be applied to the generation of other BVH structures for the SSI problem (including CPU-based implementations). The BVH-based approach belongs to the category of the subdivision method. (The subdivisions are often made globally all over the surface in a preprocessing stage of the BVH construction.)

At the end of the BVH traversal for the SSI computation, a tracing method is then needed for the local construction of intersection curve segments. At the curve tracing stage, our approach is different from the conventional BVH-based algorithms for intersecting two mesh models, where the intersection of two triangles typically produces a line segment [37, 38, 39, 40]. (In some degenerate cases, two triangles may overlap

in a convex polygon only when they are contained in the same plane.) On the other hand, the intersection of two toroidal patches can be more complex (as shown in Figure 5.2).

For the resolution of the SSI curve topology in space, Sederberg et al. [41, 9] developed methods for detecting closed loops in the SSI curve by comparing the Gauss maps of the two surface patches to be intersected. When there is no overlap in their Gauss maps, the SSI curve has (at most) a single branch (not forming a closed loop). Otherwise, we can recursively subdivide the surfaces into smaller patches until the Gauss maps have no overlap. Nevertheless, in the degenerate case of (almost) tangential surface intersections, we need a long sequence of recursive subdivisions and at some point we have to deal with the SSI curve with multiple branches and/or closed loops. Even in these non-trivial cases, the osculating toroidal patches provide good approximate solutions to the tangential SSI problem, thanks to their geometric simplicity and high approximation order to the surface.

Regarding the robustness issue, the bounding volumes may be interpreted as a geometric version of the interval arithmetics (in the sense that intervals are one-dimensional AABBs) or a simplified version of other conservative techniques such as the convex hulls of control points for surface patches. The BVH-based SSI approach is thus a generalization of conventional SSI methods, where the main difference is in that the subdivisions

are done globally in a preprocessing stage, to a certain high resolution such as 512×512. The traversal in the hierarchy of surface subdivisions (and their bounding volumes) is about the same as the conventional SSI approach of subdivision methods.

When two regular surfaces intersect tangentially, Ye and Maekawa [42] determine the topology of SSI curve using the Dupin indicatrices of the two surfaces. The osculating toroidal patches in the leaf nodes can serve for the same purpose as they capture the same second order local surface properties such as curvature. The simplicity of toroidal patches also makes the implementation of primitive geometric operations easy and robust.

The G^1-continuous biarc approximation to planar freeform curves was shown to be very useful not only in the acceleration of many geometric algorithms but also in the improvement of their robustness [43, 44, 45]. Liu et al. [21] employed the second order osculating torus approximation to freeform surfaces for the acceleration of point-projection algorithm. In recent work, Son et al. [16] accelerated the minimum distance computation between two surfaces of revolution using the osculating toroidal patches, which are generated by rotating the G^1-biarc approximation to the profile curves of the rotational surfaces. In this thesis, we consider the SSI problem using the osculating toroidal patches. The main advantage of using toroidal patches is in the simplicity of finding all binormal lines

between two tori by computing the real roots of a polynomial equation of degree 8, which is also the main source of acceleration for the minimum distance algorithm of Son et al. [16].

3.2 Surface Self-Intersection

There is a rich body of literatures (including some extensive survey articles) on the surface-surface-intersection (SSI) problem [32, 7, 31, 8, 33, 46, 47, 48, 49, 50]; however, the majority of previous results are on the intersection of two different surfaces. The special case of intersecting an identical surface with itself has received much less research attention than the general SSI problem. Nevertheless, this does not necessarily mean that the surface self-intersection is less important. On the contrary, the sparsity of previous work is mainly due to technical difficulties handling the self-intersection case. For example, it is not easy to deal with trivial solutions: $(u, v) = (s, t)$, in the self-intersection problem: $S(u, v) = S(s, t)$, which should not be included in the solution set, because of the non-equality constraint: $(u, v) \neq (s, t)$. Consider the intersection test for two adjacent subpatches that share a common boundary curve along an iso-parametric curve: $u = s$ or $v = t$. A good solution to this problem can be used for the elimination of trivial solutions, and vice versa. The issue is how to accelerate the computing speed for a large number of overlap

tests among nearby subpatches, which is often considerably larger than those for intersecting two different surfaces.

As a method of choice for symbolically eliminating the redundant factors $(u - s)$ and $(v - t)$ from the constraint equations for $S(u, v) = S(s, t)$, many previous algorithms took an algebraic approach [51, 52]. In the univariate case of the curve self-intersection problem: $C(u) = C(s)$, Pekerman et al. [53] removed the redundant factor $(u - s)$ from each constraint equation for the solution set. Nevertheless, in the bivariate case of freeform surfaces, simple factors such as $(u-s)$, $(v-t)$, $[(u-s)^2 + (v-t)^2]$, do not always appear in the constraint equations directly drived from each coordinate of $S(u, v) = S(s, t)$. To deal with this problem, Elber et al. [52] formulated a different set of constraint equations by combining the coordinate functions of $F(u, v, s)$ and $G(v, s, t)$, where $S(u, v) - S(s, v) = (u - s)\ F(u, v, s)$, $S(s, v) - S(s, t) = (v - t)\ G(v, s, t)$, and thus $S(u, v) - S(s, t) = (u - s)\ F(u, v, s) + (v - t)\ G(v, s, t)$. Busé et al. [51] proposed a different algorithm based on resultant techniques. Regarding the main technical issue of the current work, the detection and representation of miter points, the elimination-based algebraic approach has a clear limitation. The miter points are characterized by the triviality condition: $(u, v) = (s, t)$, the detection of which becomes more difficult after the elimination of all trivial solutions. Thus we focus on geometric approaches that gradually reduce the given self-intersection problem to

relatively simple subproblems.

Volino and Thalmann [54, 55] presented a subdivision-based algorithm for computing the self-intersection of triangular meshes. The self-intersection-free condition developed in this approach is quite similar to the normal cone test traditionally used in the SSI algorithms [41, 9]. Ho and Cohen [24, 25] detected the miter point locations based on a necessary condition: $S_u \times S_v = \mathbf{0}$, where S_u and S_v are the partial derivatives of $S(u, v)$. They improved the solution curve, in particular, the locations of miter points, by carefully controlling the speed of curve tracing near miter points.

Galligo and Pavone [2] used a triangular mesh approximation and reported certain limitations of the mesh-based approach in the neighborhoods of miter points because the approximating triangles become almost coplanar. Hong et al. [56] considered the self-intersection problem for offset surfaces, where osculating toroidal patches are used for a stable curve tracing near the miter points. They have observed that the location of a miter point in the xyz-space is stable; however, the corresponding location in the parameter domain is numerically unstable to detect in a high precision. They also have observed that the self-intersection curve takes a local shape of line segment with an open endpoint at the miter point location.

The detection and elimination of self-intersections in an offset surface

has long been an important problem with applications in NC toolpath generation [57, 58, 59, 60, 61]. The offset self-intersection curves are often traced using numerical techniques (such as Runge-Kutta methods) applied to differential equations [57, 59, 61]. The offset trimming technique of Seong et al. [60] is based on the approximation of offset surfaces using rational freeform surfaces, in which subtle geometric details such as miter point locations can be changed to something else.

3.3 Summary

In this chapter, we briefly introduce previous studies on the surface intersection problem studied for a long time. Many surface intersection algorithms were introduced in the survey articles, and many papers are mentioned and cited. Since then, there have been relatively few publications in the research, and new results were shown for special types of surfaces. On the other hand, recent developments of BVH for freeform surfaces make geometry algorithm efficient, and second order osculating toroidal patches give a high precision of surface approximation [21]. This evidence is worth a revisit the surface intersection problem to solve some highly non-trivial surface intersection problems. We introduce a new data structure algorithm to solve the surface intersection problem in the following chapters.

Chapter 4

Bounding Volume Hierarchy for Surface Intersections

4.1 Binary Structure

In this section, we demonstrate how to construct a Bounding Volume Hierarchy(BVH) for a bicubic Bézier surface. For the sake of simplicity in the representation, we consider bicubic Bézier surfaces only. Our BVH structure can easily extend to any tensor product surfaces to an arbitrary degree. The input freeform surface $S(u, v)$ is first approximated by a rectangular array of bilinear surfaces $L_{ij}(u, v)$, $(u, v) \in [u_{i-1}, u_i] \times [v_{j-1}, v_j]$, for $i, j = 1, \cdots, 2^H$, where $u_i = \frac{i}{2^H}$ and $v_j = \frac{j}{2^H}$. Each bilinear surface $L_{ij}(u, v)$ is then approximated by a planar quadrangle $Q_{ij}(u, v)$. The approximation error between $Q_{ij}(u, v)$ and the subpatch $S_{ij}(u, v) = S(u, v)$,

on the subdomain $[u_{i-1}, u_i] \times [v_{j-1}, v_j]$, is bounded by $\epsilon_{ij} > 0$. Expanding the quadrangle Q_{ij} by the distance ϵ_{ij}, in all three dimensions, a bounding volume V_{ij} is generated for the rectangular subpatch S_{ij}. Starting from the bounding volumes V_{ij} at the leaf level nodes at the height H, we construct the BVH for the surface $S(u, v)$, in a bottom-up fashion.

4.1.1 Hierarchy of Bilinear Surfaces

For an efficient generation of the bicubic Bézier surface

$$S(u, v) = \sum_{\alpha=0}^{3} \sum_{\beta=0}^{3} \mathbf{b}_{\alpha\beta} B_\alpha^3(u) B_\beta^3(v),$$

for $0 \leq u, v \leq 1$, where $\mathbf{b}_{\alpha\beta}$ are the Bézier control points, we save the cubic Bézier basis function values precomputed at the uniform sample parameters $u_i = \frac{i}{2^H}$ and $v_j = \frac{j}{2^H}$ and reuse them repeatedly as needed in the rest of the SSI construction:

$$B_\alpha^3(u_i) = \frac{3!}{(3-\alpha)!\alpha!} (1 - u_i)^\alpha (u_i)^{3-\alpha},$$

for $\alpha = 0, 1, 2, 3$, and $i = 0, \cdots, 2^H$. These function values are reused for both $B_\alpha^3(u_i) = B_\alpha^3(i/2^H)$ and $B_\beta^3(v_j) = B_\beta^3(j/2^H)$.

The precomputation of Bézier basis functions accelerates the uniform sampling of the surface $S(u, v)$ at the parameters (u_i, v_j), for

$i, j = 0, \cdots, 2^H$. Each subpatch $S_{ij}(u, v) = S(u, v)$, on the subdomain $[u_{i-1}, u_i] \times [v_{j-1}, v_j]$, can be approximated by a bilinear surface $L_{ij}(u, v)$ that interpolates the four corner points of $S_{ij}(u, v)$, which are among the uniform sampling points of $S(u, v)$:

$$\begin{aligned} L_{ij}(u,v) &= \left(\frac{u-u_{i-1}}{2^H}\right)\left(\frac{v-v_{j-1}}{2^H}\right) S(u_{i-1}, v_{j-1}) \\ &+ \left(\frac{u_i-u}{2^H}\right)\left(\frac{v-v_{j-1}}{2^H}\right) S(u_i, v_{j-1}) \\ &+ \left(\frac{u-u_{i-1}}{2^H}\right)\left(\frac{v_j-v}{2^H}\right) S(u_{i-1}, v_j) \\ &+ \left(\frac{u_i-u}{2^H}\right)\left(\frac{v_j-v}{2^H}\right) S(u_i, v_j). \end{aligned} \tag{4.1}$$

The above uniform sampling points $S(u_i, v_j)$ also include the sampling points of $S(u, v)$ at lower resolutions. For example, when we replace the maximum height H by a smaller value h, the corresponding uniform sample parameters $\hat{u}_i = \frac{i}{2^h}$ and $\hat{v}_j = \frac{j}{2^h}$ also belong to the original $(1 + 2^H)$ sample parameters:

$$\hat{u}_i = \frac{i}{2^h} = \frac{i \cdot 2^{H-h}}{2^H} = u_{i \cdot 2^{H-h}},$$

$$\hat{v}_j = \frac{j}{2^h} = \frac{j \cdot 2^{H-h}}{2^H} = v_{j \cdot 2^{H-h}}.$$

Consequently, there is no need of further resampling of the surface points $S(\hat{u}_i, \hat{v}_j)$, for the approximation of $S(u, v)$ at lower resolutions.

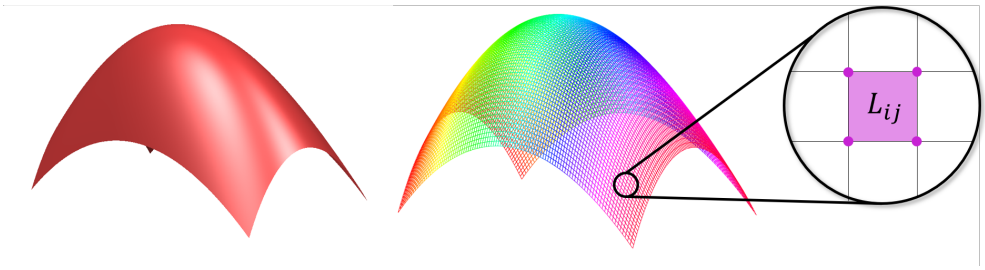


Figure 4.1: Example of bicubic Bézier surface and its bilinear surface

The Bézier surface $S(u, v)$ is approximated by a hierarchy of bilinear surfaces, for a sequence of heights $h = 0, \cdots, H$. Filip et al. [28] showed that the bilinear approximation has a quadratic convergence, which means that the approximation error is reduced by 4-times each time we increase the height h by one, to $h + 1$.

In a similar way, we can approximate the surface normal $N(u, v) = S_u \times S_v$ by a hierarchy of bilinear normal surfaces. The Gauss map of $S(u, v)$, on the subdomain $[u_{i-1}, u_i] \times [v_{j-1}, v_j]$, can be approximated by a spherical quadrangle with four corners $\hat{N}(u_{i-1}, v_{j-1})$, $\hat{N}(u_{i-1}, v_j)$, $\hat{N}(u_i, v_{j-1})$, $\hat{N}(u_i, v_j)$, on the unit sphere S^2, where

$$\hat{N}(u, v) = \frac{N(u, v)}{\|N(u, v)\|} \tag{4.2}$$

. Expanding the spherical quadrangle slightly by an amount estimated by Filip et al. [28], we can bound the Gauss map of $S(u, v)$ on the subdomain $[u_{i-1}, u_i] \times [v_{j-1}, v_j]$.

4.1.2 Hierarchy of Planar Quadrangles

The bilinear surfaces $L_{ij}(u, v)$ are quadrics in general. The interference test and the distance computation between two bilinear surfaces are more time-consuming than the case of two planar polygons in general position (i.e., two polygons not contained in the same plane). Moreover, the point-projection to bilinear surfaces (i.e., quadrics) is more difficult to compute than to planes. We approximate each quadratic surface patch $L_{ij}(u, v)$ by a planar quadrangle $Q_{ij}(u, v)$ and then by a rectangle $R_{ij}(u, v)$, so as to make the overlap test between two bounding volumes easier and thus faster. (The details of the overlap test will be discussed in Chapter 5.)

Given a bilinear surface $L_{ij}(u, v)$ of Equation (4.1) with four corners, the best plane approximation to $L_{ij}(u, v)$ is the tangent plane of $L_{ij}(u, v)$ at the midpoint:

$$L\left(\frac{u_{i-1}+u_i}{2}, \frac{v_{j-1}+v_j}{2}\right) = \frac{S(u_{i-1}, v_{j-1}) + S(u_i, v_{j-1}) + S(u_{i-1}, v_j) + S(u_i, v_j)}{4}. \quad (4.3)$$

Moving two corner points $S(u_{i-1}, v_{j-1})$ and $S(u_i, v_j)$ by

$$\mathbf{d} = \frac{-S(u_{i-1}, v_{j-1}) + S(u_i, v_{j-1}) + S(u_{i-1}, v_j) - S(u_i, v_j)}{4},$$

to

$$\frac{3S(u_{i-1}, v_{j-1}) + S(u_i, v_{j-1}) + S(u_{i-1}, v_j) - S(u_i, v_j)}{4},$$

and

$$\frac{-S(u_{i-1}, v_{j-1}) + S(u_i, v_{j-1}) + S(u_{i-1}, v_j) + 3S(u_i, v_j)}{4},$$

and similarly moving the other corner points $S(u_i, v_{j-1})$ and $S(u_{i-1}, v_j)$ by $-\mathbf{d}$, we construct the quadrangle $Q_{ij}(u, v)$ contained in the tangent plane at the midpoint of $L_{ij}(u, v)$.

Filip et al. [28] showed that the distance between the planar quadrangle $Q_{ij}(u, v)$ (which is also a parallelogram) and the subpatch $S_{ij}(u, v)$ can be bounded as follows:

$$\begin{aligned} &\|S_{ij}(u, v) - Q_{ij}(u, v)\| \\ &\qquad \leq \frac{\sqrt{3}\left(\|\hat{S}_{\hat{u}\hat{u}}\|_\infty + 2\|\hat{S}_{\hat{u}\hat{v}}\|_\infty + \|\hat{S}_{\hat{v}\hat{v}}\|_\infty\right)}{8} = \hat{\epsilon}_{ij}, \end{aligned} \tag{4.4}$$

where $\|\mathbf{v}\|_\infty = \max\{|v_x|, |v_y|, |v_z|\}$ for a vector $\mathbf{v} = (v_x, v_y, v_z)$, and $\hat{S}_{\hat{u}\hat{u}}, \hat{S}_{\hat{u}\hat{v}}, \hat{S}_{\hat{v}\hat{v}}$ are the second partial derivatives of a Bézier surface $\hat{S}_{ij}(\hat{u}, \hat{v})$, $(\hat{u}, \hat{v}) \in [0, 1] \times [0, 1]$, obtained by reparameterizing the sub-

patch $S_{ij}(u,v)$, $(u,v) \in [u_{i-1}, u_i] \times [v_{j-1}, v_j]$, as follows:

$$\hat{S}_{ij}(\hat{u},\hat{v}) = S\left(u_{i-1} + \hat{u} \cdot \frac{i}{2^H}, v_{j-1} + \hat{v} \cdot \frac{j}{2^H}\right), \tag{4.5}$$

for $0 \leq \hat{u}, \hat{v} \leq 1$. By expanding the planar quadrangle $Q_{ij}(u,v)$ by $\hat{\epsilon}_{ij} > 0$, we can construct a bounding volume for the subpatch $S_{ij}(u,v)$.

Figure 4.2(a) shows a sphere-swept volume over the tetrahedron made of the four corners of the subpatch $S_{ij}(u,v)$, where $\hat{\epsilon}_{ij}$ is the radius of the sweeping sphere. A similar sphere-swept volume over the quadrangle $Q_{ij}(u,v)$ is shown in Figure 4.2(b). As one can easily notice in Figure 4.2(b), the upper bound $\hat{\epsilon}_{ij}$ is often larger than the maximum deviation of the Bézier surface $\hat{S}_{ij}$ from the quadrangle Q_{ij}, which can be bounded by the maximum distance of the control points of $\hat{S}_{ij}$ to the planar quadrangle Q_{ij}. Moreover, we can construct even a simpler bounding volume by replacing the planar quadrangle Q_{ij} by a rectangle R_{ij} (as shown in Figure 4.2(c)). For this purpose, let P_{ij} denote the plane of Q_{ij}, and P_{ij}^+ and P_{ij}^- denote the parallel planes of P_{ij} that tightly bound the control points of $\hat{S}_{ij}$ from both sides of P_{ij}.

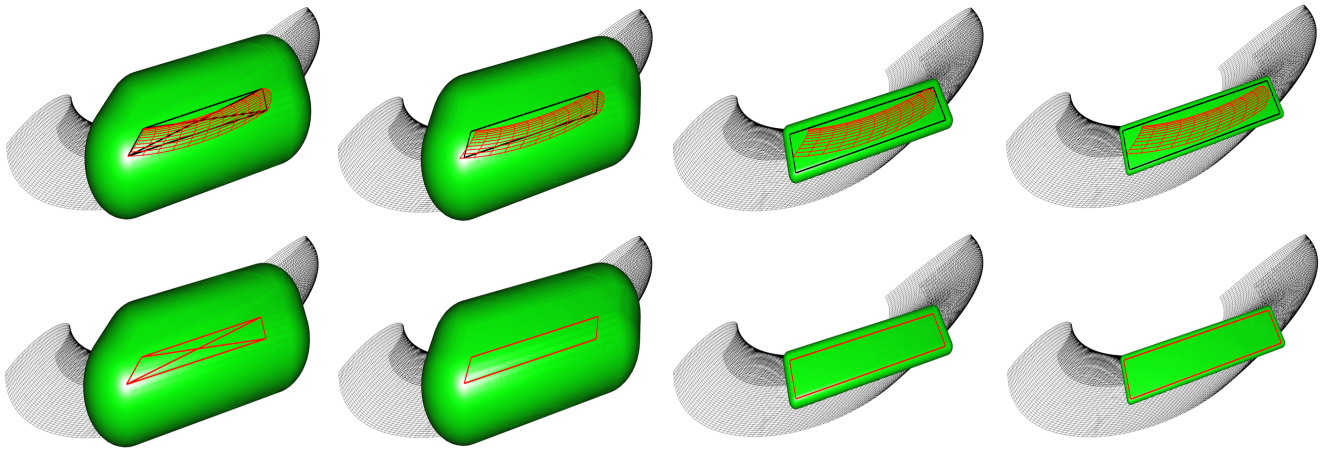


Figure 4.2: Sphere-swept bounding volumes for a Bézier surface patch $\hat{S}_{ij}$: (a) tetrahedron-swept sphere (TSS), (b) quadrangle-swept sphere (QSS), (c) rectangle-swept sphere (RSS), and (d) rectangle-swept sphere (RSS) made tigher; in the first row, the surface patch $\hat{S}_{ij}$ is shown together with each bounding volume, and in the second row, only the bounding volumes are shown for a better visual comparison of their tightness.

Now, let R_{ij}^o denote an optimal rectangle that bounds all projected control points of $\hat{S}_{ij}$ to the plane P_{ij}. By projecting R_{ij}^o to the parallel planes P_{ij}^+ and P_{ij}^-, we get two rectangles R_{ij}^+ and R_{ij}^-, which form two opposite faces of an OBB (Oriented Bounding Box) for the Bézier surface $\hat{S}_{ij}$. When we continuously shrink the OBB by moving all six faces inward simultaneously, the whole volume will collapse to a rectangle R_{ij}. Now, we measure the maximum distance of the control points of $\hat{S}_{ij}$ to the rectangle R_{ij}, which is often contained in the mid-plane between two parallel planes P_{ij}^+ and P_{ij}^-, but of a smaller size than the bounding rectangle R_{ij}^o. The rectangle-swept volume by the maximum distance is the bounding volume for the Bézier surface $\hat{S}_{ij}$, which is shown in Figure 4.2(d).

4.1.3 Construction of Leaf Nodes with Osculating Toroidal Patches

Each node of our BVH contains a bounding sphere and a rectangle-swept sphere for a surface patch that corresponds to the node. In addition to these bounding volumes, each leaf node also contains an osculating toroidal patch determined by the curvature of the Bézier surface $\hat{S}_{ij}$ at the midpoint $\hat{S}_{ij}\left(\frac{1}{2}, \frac{1}{2}\right)$. Depending on the surface curvature at the mid-point, we take either the outer part or the inner part of the torus for approximating the surface of positive or negative curvature [21].

The osculating toroidal patch is taken sufficiently large enough to cover both the positions and the normal directions of the Bézier surface $\hat{S}_{ij}(u, v)$, $(u, v) \in [0, 1] \times [0, 1]$. Moreover, we take the boundary of the toroidal patch as the meridian and parallel circular arcs, which greatly simplifies geometric operations on the toroidal patch (Figure 4.3). For example, the Gauss map of such a toroidal patch is bounded by four circular arcs on the Gaussian sphere. Taking larger toroidal patches bounded by circular arcs, it is easy to test the non-overlap condition for their Gauss maps, which also implies a similar condition for the two Bézier surface patches approximated by the toroidal patches.

We can trim the osculating toroidal patches T_{ij} using non-circular boundary curves so that the Hausdorff distance between the trimmed patch and the Bézier surface $\hat{S}_{ij}(u, v)$ is within a given error bound $\epsilon > 0$. This can be done by projecting the four corner points of $\hat{S}_{ij}(u, v)$ to the toroidal patch and connecting the projected points to a quadrangle in

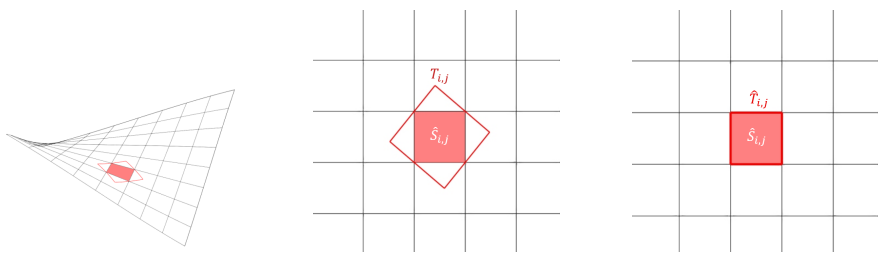


Figure 4.3: Construction of osculating toroidal patches and their trimming for surface matching.

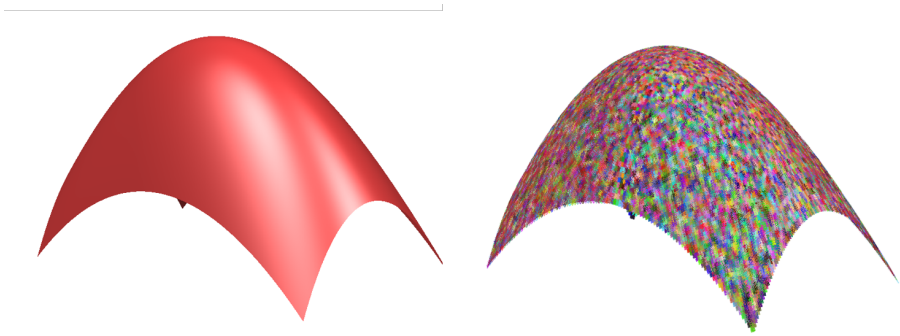


Figure 4.4: a bicubic Bézier surface and set of osculating toroidal patches.

the parameter space of the torus.

By reparameterizing the quadrangle bilinearly, we can represent the trimmed toroidal patch as $\hat{T}_{ij}(u, v)$. Based on this parameter matching, we can bound the Hausdorff distance between $\hat{S}_{ij}$ and $\hat{T}_{ij}$ by $\max_{(u,v)} \|\hat{S}_{ij}(u, v) - \hat{T}_{ij}(u, v)\|$. If the bound is larger than the given error bound $\epsilon > 0$, we can repeat the same procedure recursively by subdividing the Bézier surface $\hat{S}_{ij}(u, v)$ into four pieces. Finally, we get a union of toroidal patches $\hat{T}_{ij}$, which may not even be connected to each other, but whose Hausdorff distance with the given Bézier surface $S(u, v)$ is guaranteed to be within an error bound $\epsilon > 0$. By intersecting the toroidal patches, we approximate the SSI curve of the given freeform surfaces using the parameter matching between $\hat{S}_{ij}(u, v)$ and $\hat{T}_{ij}(u, v)$.

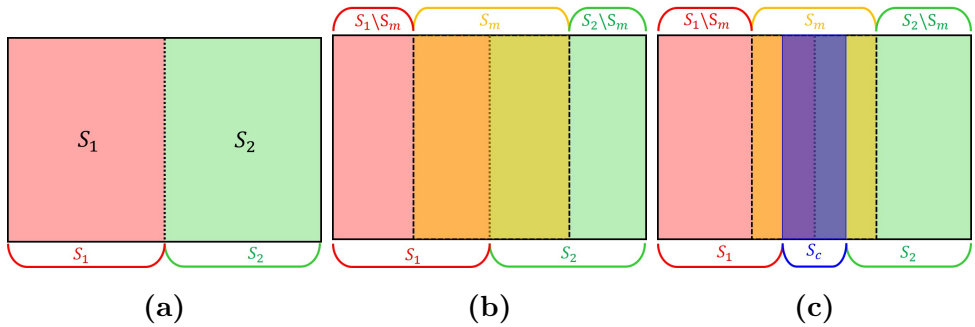


Figure 4.5: Recursive self-intersections: (a) in S_1 and S_2 sharing a common boundary curve, (b) in S_1, S_m, S_2 with S_m covering the common boundary, and (c) in S_1, S_m, S_2, followed by global intersections.

4.2 Ternary Structure

For surface self-intersection problem, the binary BVH structure is quite problematic. As shown in Figure 4.5(a), when we split a surface S into two smaller pieces $S = S_1 \cup S_2$, the self-intersection of S can be computed by the recursive self-intersections of S_1 and S_2, followed by a global intersection between S_1 and S_2. There is one serious problem in this simple approach – the global intersection $S_1 \cap S_2$ may include the common boundary of S_1 and S_2 in the solution, which is different from what we have intended to compute. It is computational expensive to decide whether the two surfaces S_1 and S_2 are smoothly connected or locally intersecting along their common boundary. The conventional subdivision-based intersection algorithms are usually slowed down by recursively sub-

dividing the surface into smaller and smaller pieces along the common boundary.

Therefore, we follow the basic guideline of binary BVH (designed for SSI problem), but make some changes to the BVH structure. A simple (but incomplete) remedy may be to include the self-intersection test for a marginal surface S_m that covers the common boundary curve (Figure 4.5(b)). And then we may replace the global intersection $S_1 \cap S_2$ by a new version of $(S_1 \setminus S_m) \cap (S_2 \setminus S_m)$. (But, this is incomplete as there are some parts missing from the solution – $(S_1 \setminus S_m) \cap (S_2 \cap S_m)$ and $(S_1 \cap S_m) \cap (S_2 \setminus S_m)$ should also be included.) A correct version (suitable for a BVH-based implementation) is a bit more complicated. As shown in Figure 4.5(c), we can split the surface S into three pieces $S = S_1 \cup S_c \cup S_2$, where the boundary curve is replaced by a surface patch S_c in the center. The marginal surface S_m now covers S_c, again with some overlaps with S_1 and S_2. Then we do recursive self-intersections on S_1, S_2, and S_m. The global intersections are then computed for $S_1 \cap S_2$, $(S_1 \setminus S_m) \cap S_c$, and $S_c \cap (S_2 \setminus S_m)$.

The recursive subdivision of a surface into three smaller pieces means that each internal node of our BVH structure would have three child nodes: the left, middle, and right nodes. The middle node should be responsible for the recursive self-intersection of S_m, and thus it is the root of the subtree for S_m. The BVH structure is ideal for handling the overlap

parts of S_m with S_1 and S_2. Instead of recursively constructing three other subtrees for smaller surfaces: $(S_1 \setminus S_m)$, S_c, and $(S_2 \setminus S_m)$, we can implicitly represent these subtrees by restricting the parameter domains of the subtrees for S_1, S_m, and S_2. For example, in the construction of three child nodes of S_m, their bounding volumes are generated for three surfaces $(S_m)_1$, $(S_m)_m$, and $(S_m)_2$. It is clear that the union of these three volumes also bounds the surface S_c, which is smaller than S_m. The BVH for S_m can also serve as a BVH for S_c, even though it is not an optimal one for S_c. Some internal nodes of the BVH for S_m may not bound any part of S_c when they represent some marginal areas in $(S_m \setminus S_c)$. We can treat these nodes as empty when the BVH is used for the smaller surface S_c.

In the global intersection steps, we need to process smaller surfaces $(S_1 \setminus S_m)$, S_c, and $(S_2 \setminus S_m)$ than those stored in the left, middle, and right subtrees of the BVH tree. By using the parameter domains for surfaces to be intersected, we can deal with smaller surfaces using the BVH for a larger surface. When the parameter domain of a child node is outside the domain of a surface, we can simply treat the node as an empty subtree and stop further recursive search down to that direction. One disadvantage of this approach is in the duplication of the overlap area $(S_m \setminus S_c)$ in different subtrees, which may cause the detection of some self-intersections in the overlap areas multiple times. Nevertheless,

by controlling the thickness of the strips, we can reduce the number of repeated detections. The self-intersection curve is constructed for the pairs of leaf nodes after all redundant duplications are filtered out. According to our experimental results, the computational advantage from the overlap region is far more beneficial than the duplication problem.

4.2.1 Miter Points

We start with a uniform subdivision of a surface $S(u, v)$ into subpatches S_{ij}, $(i, j = 1, \cdots, 128)$. (The following construction scheme also works for a grid structure of size $8M \times 8N$, as discussed in Section 4.2.3.) Each subpatch S_{ij} is then tightly approximated by an osculating toroidal patch T_{ij} based on the construction steps discussed in Section 4.1. When the deviation of normal directions at $S_{ij}(u, v)$ and $T_{ij}(u, v)$ is larger than a certain threshold, we consider this failure of approximation as a signal for the detection of a miter point in S_{ij}.

As shown in Figure 4.6, a non-isolated miter point is the junction point of two corresponding solution curves (represented as (u, v)- and (s, t)-curves in the parameter space) for the self-intersection curve (in the Euclidean space), given by the relation: $S(u, v) = S(s, t)$. We search the two branches of these solution curves in the 1-ring neighborhood of S_{ij}. In the example of Figure 4.6(b), the domains for $S_{i,j+1}$ and $S_{i,j-1}$ have the corresponding branches, which are computed by intersecting the two

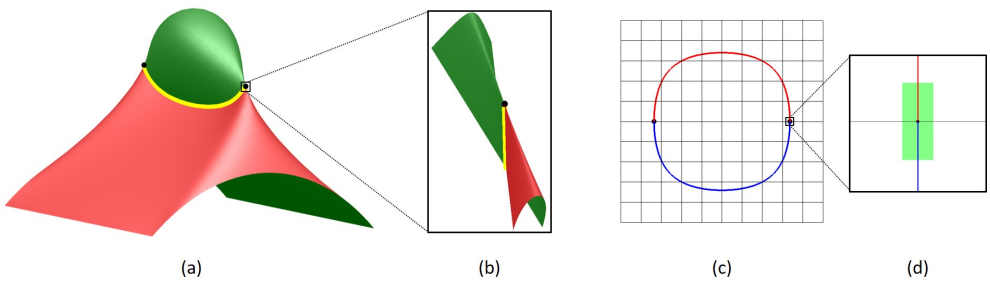
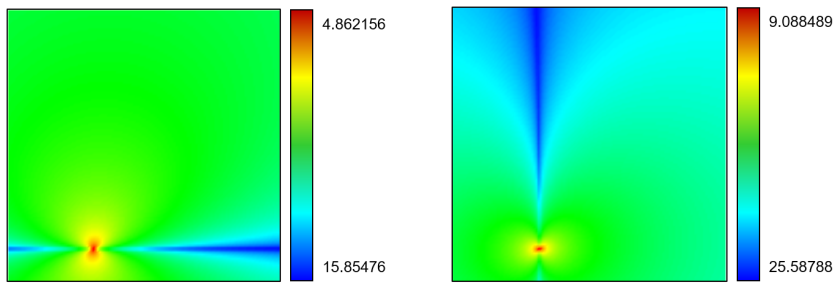


Figure 4.6: Miter points on a surface self-intersection curve: (a)–(b) in the Euclidean xyz-space, (c) in the (u, v)-parameter domain, and (d) bounding a miter point using a quadrangle.

osculating toroidal patches $T_{i,j+1}$ and $T_{i,j-1}$, which is more stable than intersecting the surface $S(u, v)$ with itself near a miter point. (In fact, we iteratively improve the intersection result by recomputing $T_{i,j+1}$ and $T_{i,j-1}$ at the corresponding solution points and incrementally walking along the self-intersection curve in small steps.)

We extend the (u, v)- and (s, t)-solution curves simultaneously from $S_{i,j+1}$ and $S_{i,j-1}$ so that they can meet at a miter point inside S_{ij}. The miter point location is computed by solving $S_u \times S_v = \mathbf{0}$ [34], or we can initial guess of location of the miter point with ratio of normal to position. In Figure 4.7, logarithm of ratio of normal to position is appeared in red to blue color scale. The numerical tracing along the solution curves works only up to a certain pair of solution points. Using the positions and tangent directions of the two solution curves at these points, we guess the remaining solution curves by interpolating a Bézier curve to



(a) **(b)**

Figure 4.7: Initial guess for miter quadrangles with ratio of normal to position in u-direction (a), and v-direction (b).

these data. The Bézier curve is then bounded by a quadrangle Q (in the parameter space of $S(u, v)$) as a regional representation for the miter point. The quadrangle Q is then represented as a bilinear surface $Q(\alpha, \beta)$, $(0 \leq \alpha, \beta \leq 1)$, in the uv-parameter domain. The composite surface $S(Q(\alpha, \beta))$ in the Euclidean xyz-space will be a very tiny surface patch containing the miter point of S_{ij}. When this small surface patch is totally contained in an ϵ-ball, we are done. Otherwise, we reduce the size of Q so as to improve the approximation result.

What if all these tests fail in the detection of Q or the bounding of $S(Q(\alpha, \beta))$ in an ϵ-ball? Then we increase the resolution of uniform subdivision in the 1-ring neighborhood of S_{ij}, and repeat the same procedure in this restricted domain. We also need to take this approach when there is no branch or only one branch in the 1-ring neighborhood of S_{ij}. We may have the same problem repeatedly even with higher and higher res-

olutions; then, we make the following decisions: (i) the no-branch case as an isolated miter point, and (ii) the one-branch case as a signal for the failure of our algorithm in separating two almost identical branches from each other. In the one-branch case, it would also be possible to have a small self-intersection loop or no self-intersection curve yet but about to start one if the surface shape is slightly changed. Thus it is reasonable to consider this case as an isolated miter point as well.

4.2.2 Leaf Nodes

The quadrangle Q is usually contained inside the domain of S_{ij}. In some cases, there may be overlap with two adjacent domains or three/four domains with a common corner. We slightly expand the domain (and the corresponding subpatch) with maximum overlap and shrink other domains. This works under the assumption that Q is much smaller than the domain of S_{ij}. Otherwise, Q may contain multiple miter points, and the construction of Q should be repeated more carefully with higher precision for possible detection of multiple miter points if there are some.

Now, as shown in Figure 4.8, we convert the leaf node for S_{ij} to an internal node by splitting it into three child nodes. The vertical bounding slab for Q is taken as the middle child node, from which we go one more step down the BVH tree by adding three child nodes. This time, the horizontal bounding slab for Q is the middle one. Let R denote the middle

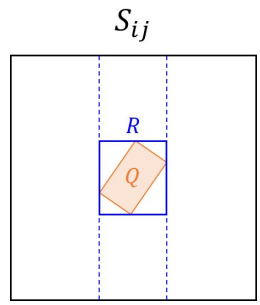


Figure 4.8: Converting S_{ij} to an internal node.

leaf node, which is a minimal AABB for Q. In case the composite surface $S(R(u, v))$ can be bounded in an ϵ-ball, we replace Q with the rectangle R. Otherwise, the middle leaf node contains Q and four triangles, one for each edge of Q.

4.2.3 Internal Nodes

The uv-parameter domain $[0, 1] \times [0, 1]$ of the surface $S(u, v)$ is first split vertically, then horizontally, and alternating the two directions in the rest of the BVH construction. In Figure 4.5(c), the three subpatches S_1, S_m, S_2 are made of the same size, having width 3/8 and height 1. (The width of S_c is 2/8, and each of the overlaps $S_1 \cap S_m$ and $S_2 \cap S_m$ has width 1/16.) They become the three child nodes for the root of the BVH tree. Each child then becomes an internal node by splitting horizontally in the same ratio. Starting with a uniform grid of 128×128 subpatches S_{ij}, the first row of Figure 4.10 shows the grid sizes of these intermediate subpatches

in the vertical and horizontal splits. Repeating the same procedure one more time to each uniform grid of 48×48, each internal node of the BVH at this level will cover a grid of size 18×18. After that, we have to change the ratio of splits so that the split lines always go through the boundary curves of some S_{ij} but never through the interiors of some subpatches. (Otherwise, the width or height of an overlap region $S_1 \cap S_m$ at this level will be 18/16 of the size of S_{ij}, and the boundary of S_m will go through the interior of some subpatch S_{ij} .)

In the grid of 18×18, we set the widths of S_1, S_c, S_2 to 8, 2, 8, and that of S_m to 4. The horizontal splits are also made in the same way. (The second row of Figure 4.10 shows the grid sizes of the corresponding intermediate subpatches.) In the remaining lower levels of the BVH construction, we leave no overlap region by making $S_m = S_c$. Thus, the size 8 will be split to $8 = 3 + 2 + 3$, and the size 4 will be $4 = 2 + 0 + 2$. At this stage, the self-intersection test results are explicitly recorded in the BVH nodes by doing the test in a preprocessing stage for small windows of 3×3 or 2×2 subpatches. In Section 6.1, we have more details of the preprocessing computations. (In fact, we do this test for each 3×3 windows of the total grid of size 128×128, in a preprocessing step, and store the results in the grid structure. The tests for 2×2 windows can be done as a part of the tests for 3×3 windows containing them.) If the recorded test result is no self-intersection, we can save computing time by

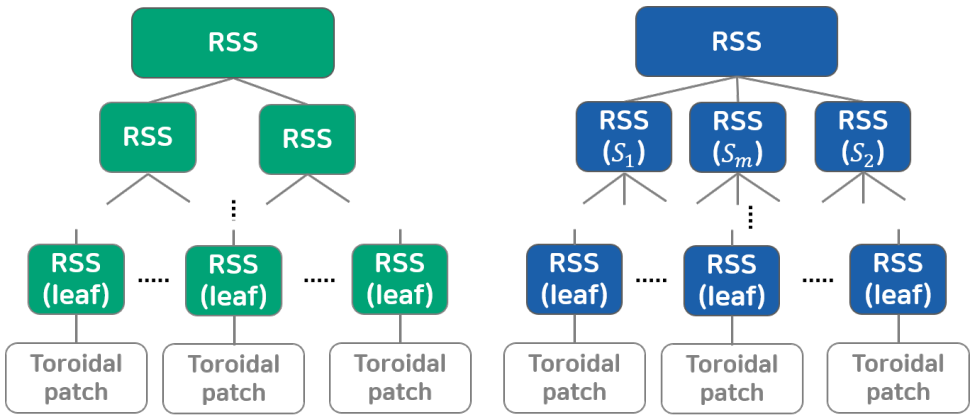


Figure 4.9: Binary and ternary BVH structure.

skipping the self-intersection test. Otherwise, we should be ready for the detection of miter points and the construction of solution curves for the surface self-intersection. This is done in the BVH subtree, with its root at the corresponding internal node. From the internal nodes of size 3×3 or 2×2, we go down to the lower levels (in two more steps) using the split ratios of $3 = 1 + 1 + 1$ and $2 = 1 + 0 + 1$, where the number 0 in the middle means that there is no middle child from the node of size 2×2. At this stage, all leaf nodes are at the same level 10. Some leaf nodes are converted to an internal node as discussed above when they contain miter points and their regional representation as a small quadrangle Q.

Remark: We can also construct a ternary BVH tree for a grid of $8N \times 8N$ subpatches S_{ij}. For example, when the grid size is 256×48, the vertical splits are made twice to generate nodes of size 96×48, and

then of size 36×48. After that, the horizontal splits produce nodes of size 36×18. Now, 36 and 18 are not multiples of 8, and we need to use different ratios for the splits. The basic rule is to build a balanced ternary BVH tree at high levels and at the same time to provide some gaps between two subpatches to be intersected for global intersections. The split ratio $8 = 3 + 2 + 3$ is intended for this purpose. At low levels of the BVH tree, we have to compromise. The grids of small sizes make the split business more difficult, often leaving no more overlap regions. On the other hand, the local self-intersections become relatively easier to check for small windows of adjacent subpatches. The self-intersection test results can be pre-computed and recorded in the grid structure for later usage. We discuss more details in Chapter 6.

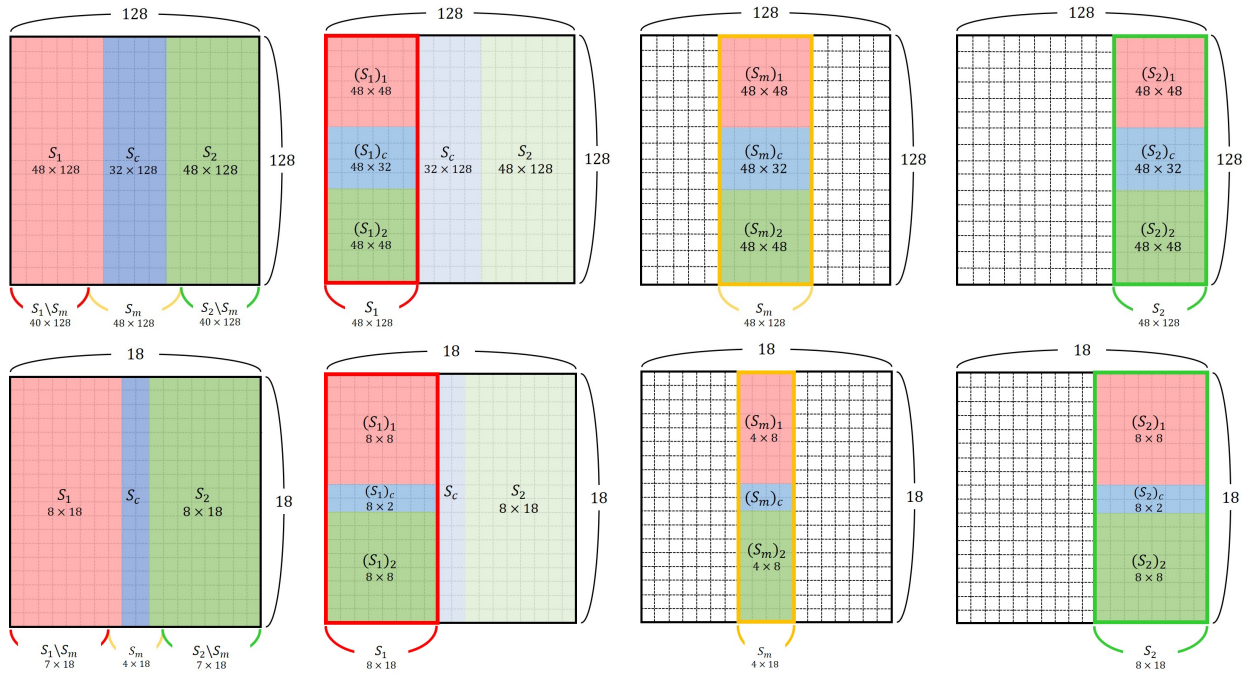


Figure 4.10: Grid sizes for the subpatches generated by the vertical and horizontal splits.

4.3 Summary

In this chapter, we introduce a new BVH data structure for surface intersections. First, the binary structure for surface-surface-intersection problem is the same as the typical BVH structure that has been used for a long time. However, the BVs are consist of rectangle-swept-sphere(RSS), and the leaf nodes additionally have a second-order contact osculating toroidal patches. The binary structure becomes a problem for surface self-intersection problems because the child nodes necessarily have a common boundary. The surface intersection algorithm recursively subdivides the surface into smaller and smaller pieces along the common boundary, it takes considerably more computing time, and the advantage of using BVH is lost. Therefore, we introduce a ternary structure, accelerating the decision procedure and avoiding the common boundary problem.

Chapter 5

Surface-Surface-Intersection

Our surface-surface-intersection algorithm is based on traversing the binary BVH tree demonstrated in Section 4.1. The BVH-based SSI algorithm proceeds by traversing the BVHs for the two freeform surfaces and detecting all pairs of leaf nodes whose bounding volumes overlap. For each pair of overlapping leaf nodes, we first intersect their osculating toroidal patches. When the two toroidal patches have no overlap in their Gauss maps, their SSI curve may have at most one branch with no bifurcation. Otherwise, the two surfaces may have tangential intersections, and we need further steps to determine the topological type of the SSI curve segments. In the final stage, we connect all the SSI curve segments thus constructed, in a correct topology.

5.1 BVH Traversal

Given two bicubic Bézier surfaces $S_A(u, v)$ and $S_B(s, t)$, let A and B denote their respective BVHs. The traversal of A and B is about the same as the sequence of subdivisions of S_A and S_B in the conventional SSI algorithms (that belong to the category of subdivision method). Instead of testing the overlap between the convex hulls of control points for the two surface patches to be intersected, we test the overlap between the bounding volumes stored in the interior nodes of A and B that correspond to the two surface patches.

When there is no overlap between the two bounding volumes, there is no intersection between the two surface patches under consideration. In the case of overlap, when A's bounding volume is larger than B's, we go down to the child nodes of A and compare them with the current node of B; otherwise, we switch the roles of A and B. At a leaf node of A or B, we also switch the roles of A and B. When we are at the leaf nodes of both A and B, we move to the next stage of constructing SSI curve segments using the osculating toroidal patches stored in the leaf nodes of A and B.

5.2 Construction of SSI Curve Segments

Given two leaf nodes of A and B and their osculating toroidal patches, it is easy to construct their Gauss maps and to test their overlap. When the Gauss maps have no overlap, the toroidal patches may have at most one branch in their SSI curve. In the case of having one branch, each endpoint of the SSI curve segment is located on the boundary of one toroidal patch [41, 9]. As discussed in Section 4.1.3, the osculating toroidal patches have their boundaries as circular arcs. By intersecting each of the boundary circular arcs against the other toroidal patch, we can find the two endpoints of the SSI curve branch. The SSI curve interior can then be approximated by a numerical tracing technique [62, 63, 64] or a specialized algorithm [34, 65, 11] for torus.

The intersection curve segments between two toroidal patches are then projected to the uv-parameter domain of the surface $S_A(u, v)$, using the parameter matching between $\hat{S}_{ij}(u, v)$ and $\hat{T}_{ij}(u, v)$ discussed at the end of Section 4.2. (The projection to the st-domain of the other surface $S_B(s, t)$ can be done in the same way.) There is one technical problem to consider here. Two adjacent trimmed toroidal patches $\hat{T}_{i-1,j}(u, v)$ and $\hat{T}_{ij}(u, v)$ are not connected and they share no common boundary. (On the other hand, their counterparts $\hat{S}_{i-1,j}(u, v)$ and $\hat{S}_{ij}(u, v)$ are smoothly connected on the same Bézier surface $S_A(u, v)$.) This means that the

projections of the SSI curve segments from $\hat{T}_{i-1,j}(u,v)$ and $\hat{T}_{ij}(u,v)$ to the uv-domain may not be connected across their common boundary line $u = u_i$. In the next subsection, we discuss how to deal with this problem by constructing a connected sequence of G^1-continuous biarcs in the uv-domain. The projected curve segments are approximated by G^1-biarcs within a given error bound $\epsilon > 0$.

5.2.1 Merging SSI Curve Segments with G^1-Biarcs

Each SSI curve segment in the uv-domain is usually given as a sequence of points $\mathbf{p}_k$, $(k = 0, \cdots, N)$, representing a polyline. Figure 5.1 represents three typical forms of SSI curve segment. For the sake of simplicity, we consider the construction of a G^1-continuous u-monotone curve segment on a subdomain $[u_{i-1}, u_i] \times [v_{j-1}, v_j]$. We assume that the first point $\mathbf{p}_0$ is located on the left boundary $u = u_{i-1}$ and the last point $\mathbf{p}_N$ is on the line $u = u_i$. There may be a slightly different point $\mathbf{q}_0$ (on the same line $u = u_i$) as a starting point for the next SSI curve segment on the subdomain $[u_i, u_{i+1}] \times [v_{j-1}, v_j]$ on the right. Thus we need to relocate $\mathbf{p}_N$ to $\mathbf{p}_N^*$, simply by averaging $\mathbf{p}_N^* = (\mathbf{p}_N + \mathbf{q}_0)/2$, or for better, by numerically solving $S_1(u_i, v) = S_2(s, t)$. Then, we set $\mathbf{q}_0^* = \mathbf{p}_N^*$. The relocation $\mathbf{p}_0^*$ is computed in a similar way.

When the SSI curves are generated, the tangent directions $\mathbf{v}_k$ at the SSI points $\mathbf{p}_k$ are also computed in the solution process. Using these $\mathbf{v}_k$,

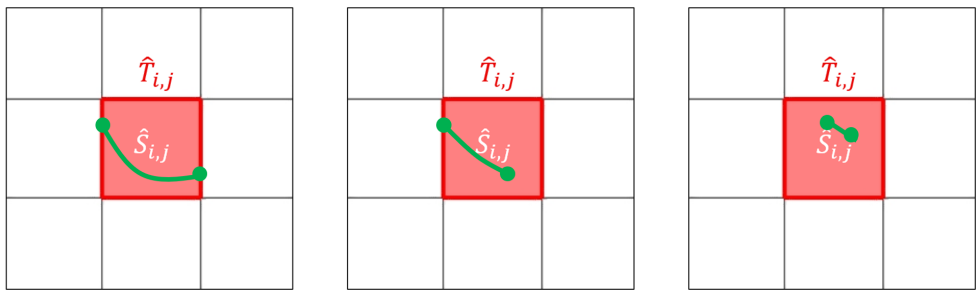


Figure 5.1: Three types of SSI curve segment.

we compute the tangent direction $\mathbf{v}_N^*$ at the new location $\mathbf{p}_N^*$, simply by averaging the directions for $\mathbf{p}_N$ and $\mathbf{q}_0$, or as a byproduct of the solution for $\mathbf{p}_N^*$. In a similar way, we get a new direction $\mathbf{v}_0^*$ at the new starting point $\mathbf{p}_0^*$.

We construct a G^1-biarc that interpolates the two boundary conditions: $(\mathbf{p}_0^*, \mathbf{v}_0^*)$ and $(\mathbf{p}_N^*, \mathbf{v}_N^*)$ [66, 67], and measure the maximum deviation of the intermediate points $\mathbf{p}_k$, $(k = 1, \cdots, N-1)$. When the maximum deviation is larger than a given tolerance $\epsilon > 0$, we repeat the approximation using two G^1-biarcs by interpolating an additional point $\mathbf{p}_m$ and its tangent vector $\mathbf{v}_m$. The point $\mathbf{p}_m$ is taken, simply as the middle point, or for better, as the point of maximum deviation from the G^1-biarc approximation. We relocate $\mathbf{p}_m$ to a more accurate position $\mathbf{p}_m^*$ and the corresponding tangent direction $\mathbf{v}_m^*$. After that, we repeat the G^1-biarc constructions recursively in two subproblems.

In general, we need to consider the G^1-biarc construction for various

different cases such as: (i) from a boundary point $\mathbf{p}_0^*$ to an X-junction point $\mathbf{p}_N$, or the other way around, (ii) from the leftmost point $\mathbf{p}_0^*$ of a closed loop to the rightmost point $\mathbf{p}_N^*$ of the same loop, and then back to $\mathbf{p}_0^*$ around the other side of the loop, and so on. The one branch condition (on two transversally intersecting surface patches) greatly simplifies the ordering of the SSI points $\mathbf{p}_k$, and thus the G^1-biarc approximation of these ordered points. Nevertheless, the ordering problem becomes more difficult when the surfaces intersect (almost) tangentially.

Even under the one branch condition, there is one serious technical problem in our approach – the osculating toroidal patches may not intersect even though their counterparts $\hat{S}_{ij}(u, v)$ have a branch of intersection curve. (This is because the one branch condition does not necessarily guarantee the existence of one branch.) For example, the sequence of SSI points $\mathbf{p}_k$, $(k = 0, \cdots, N)$, may not be connected to another sequence of points $\mathbf{q}_l$, $(l = 0, \cdots)$, in the subdomain on the right. Nevertheless, starting with the information $(\mathbf{p}_N^*, \mathbf{v}_N^*)$, we can do the conventional numerical curve tracing [62, 63, 64] on the SSI curve: $S_1(u, v) = S_2(s, t)$. But then, what if the whole loop is missing, by bad luck, without leaving any information $(\mathbf{p}^*, \mathbf{v}^*)$ to start with? Though the chance of missing a whole loop is extremely low, we may have (almost) tangential intersections of two surfaces when their Gauss maps overlap. We have not discussed this non-trivial case, yet.

The bottom line is that, regarding the issues of robustness and topological guarantees, we can do as much as other conventional methods do. The BVH-based approach can work with any other subdivision-based SSI methods, while incorporating their apparatus (such as ensuring at least one point on each branch or loop), and at the same time, improving the performance by using the pre-built subdivision structures for freeform surfaces. Thus, in the rest of this work, instead of trying to fill small gaps for the completeness of a typical SSI work, we focus on the main computational features that can be made possible using the precomputation of osculating toroidal patches and the nice geometric properties of G^1-biarc spline curves.

5.2.2 Measuring the SSI Approximation Error Using G^1-Biarcs

The G^1-biarc spline curve $(u(\theta), v(\theta))$ is only an approximate solution to the SSI curve, where the parameter θ means an arc-length parameterization of the biarc spline curve. To test if the approximation error is within a given error bound $\epsilon > 0$, we need to bound the maximum deviation of the curve-on-surface $S_A(u(\theta), v(\theta))$ from the other surface $S_B(s, t)$. For this purpose, using a similar G^1-biarc approximation of the SSI curve in the st-domain of $S_B(s, t)$, we match pairs of circular arcs from both the uv and st-domains and estimate the upper bound for the squared

distance function:

$$d^2(\theta) = \|S_A(u(\theta), v(\theta)) - S_B(s(\theta), t(\theta))\|^2,$$

by sampling on the angles θ_k of the matching arcs and adding an error term $\eta > 0$ estimated from Filip et al. [28] to the maximum of the sampled function values, $\max d^2(\theta_k)$.

When the bounding condition: $\max d^2(\theta_k) + \eta < \epsilon^2$, is met, each of the matching SSI curves, $S_A(u(\theta), v(\theta))$ and $S_B(s(\theta), t(\theta))$, is guaranteed to be within a given tolerance $\epsilon > 0$ from the other surface. The matching pair $(S_A(u(\theta), v(\theta)), S_B(s(\theta), t(\theta)))$ is considered to be an acceptable approximate solution to the SSI computation problem. Otherwise, we need to refine the G^1-biarc SSI curves in the uv and st-parameter domains by sampling some more points in the SSI curves if necessary.

5.3 Tangential Intersection

When two surfaces intersect tangentially at a point, they share the same normal line (i.e., their *binormal line*) at the point. The location of an X-junction or a near X-junction can be detected by computing the binormal lines to the two surfaces and checking the *signed* distance between the corresponding binormal-surface intersection points. Depending on the sign, we can decide which type of (almost) tangential intersection curve

the two surfaces have. For example, in Figure 5.2, when measured along the normal direction of the blue torus, the binormal-surface intersection point of the red torus is located at a signed distance of zero, negative, and positive, respectively, in each of the three rows. Osculating toroidal patches greatly simplify this test by converting the distance computation to a much simpler problem of solving a polynomial equation of degree 8 [68, 16]. The polynomial equation can be solved considerably faster than the general case of computing binormal lines for two bivariate surfaces, where we need to deal with a system of four equations in four variables.

5.4 Summary

In this chapter, we have presented a new approach to the SSI problem for freeform surfaces, which is based on a pre-built rectangle-swept sphere(RSS) tree with osculating toroidal patches stored in the leaf nodes. RSS tree accelerates the geometric search for the potential pairs of surface patches that may generate some curve segments in the surface-surface-intersection. The high approximation order of osculating torus was shown to be an effective geometric tool for handling non-trivial cases of two freeform surfaces tangentially intersecting almost everywhere.

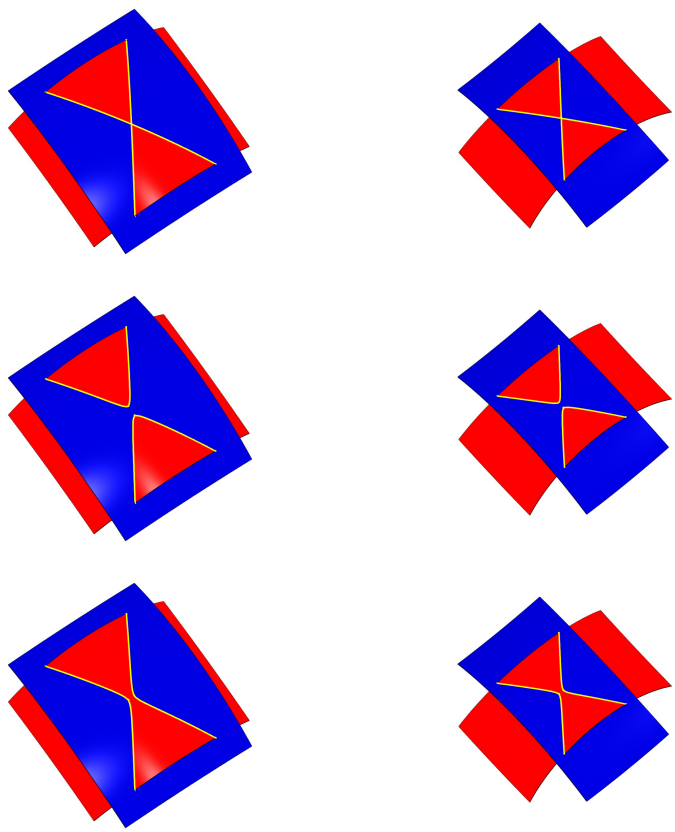


Figure 5.2: Intersection curves of toroidal patches which may have X-junctions or multiple branches; the example on the left is a pair of (convex, convex) patches and on the right is a pair of (concave, concave) patches.

Chapter 6

Surface Self-Intersection

Our surface self-intersection algorithm is based on traversing the ternary BVH tree demonstrated in Section 4.2. In a hash table, we store the pairs of leaf nodes whose bounding volumes overlap. In the insertion process to the hash table, we can filter out duplicated copies of the same pair already stored. For each pair of leaf nodes stored in the hash table, we compute their intersection curves only when the corresponding subpatches intersect each other. The (u, v) and (s, t) solution curve segments are recorded in the parameter domain of the surface. In the final step, they are connected in a correct topology.

6.1 Preprocessing

It is convenient to store surface local details in a simple grid structure of uniformly subdivided subpatches S_{ij}, $(i, j = 1, \cdots, 128)$. T_{ij} is the osculating toroidal patch of S_{ij} computed at the mid-parameter point of S_{ij}, properly parameterized and trimmed so that $\|S_{ij}(u, v) - T_{ij}(u, v)\| < \epsilon_T$, for all (u, v) in the domain of S_{ij}, and some error bound $\epsilon_T > 0$ [28, 21]. Their surface normals also satisfy a similar condition by increasing the number of subpatches in the uniform grid structure if necessary. However, for the subpatches containing miter points, the normal bounding fails repeatedly even if we reduce the size of S_{ij}, we should handle differently.

For non-miter subpatches, the Gauss map of S_{ij} can be bounded by adding some margin to that of T_{ij}, based on which we can tell that S_{ij} is self-intersection-free if the Gauss map of T_{ij} is sufficiently small and totally contained in a unit hemisphere. Using a higher resolution for the grid structure if necessary, we assume that each non-miter subpatch S_{ij} has no self-intersection. Next, we check if 2×2 and 3×3 windows of these non-miter subpatches are self-intersection-free. When the Gauss maps of T_{ij} in the window cover an area than a unit hemisphere, we record this information to the upper-left corner of a 2×2 window and to the center of a 3×3 window. The windows near a miter subpatch have some chances of being classified as this class.

6.2 BVH Traversal

Starting from the BVH root for a freeform surface $S(u, v)$, we recursively compute the self-intersection curve for each of the three child nodes: S_1, S_m, and S_2. (When the child node is empty or a leaf node, we stop the recursion.) After that, we recursively compute the global intersections for three pairs: (S_1, S_2), $(S_1 \setminus S_m, S_c)$, and $(S_2 \setminus S_m, S_c)$, where $S_c = S_m \setminus (S_1 \cup S_2)$. In the global intersection for a pair (S_A, S_B), we swap the two nodes, (i) when S_B is larger than S_A in the domain size, or (ii) when S_A is a leaf node and S_B is an internal node.

Algorithm 1: Self-Intersection of a Freeform Surface

Result: Surface Self-Intersection Curve
input: S: the root of a BVH tree for a freeform surface patch
if *S is either empty or a leaf node* **then**
  return
end
S_1, S_m, S_2: the left, middle, right child nodes for S;
Self-Intersect(S_1); Self-Intersect(S_m); Self-Intersect(S_2);
$S_c = S_m \setminus (S_1 \cup S_2)$;
Intersect(S_1, S_2); Intersect($S_1 \setminus S_m, S_c$); Intersect($S_2 \setminus S_m, S_c$);

We stop the recursion when either node is empty or both are leaf nodes. The pair of leaf nodes is then inserted to a hash table for all pairs to be intersected. In the insertion process, duplications of the same pair are filtered out. Moreover, we also stop the recursion when the two bounding volumes for S_A and S_B have no overlap. In principle, we may

use any bounding volumes for freeform surfaces. In the current work, we also use Rectangle Swept Spheres(RSS) for the overlap test as same as surface-surface-intersection problem. One exception is an ϵ-ball, for bounding the composite surface $S(Q(\alpha, \beta))$ for a miter-quadrangle Q.

Algorithm 2: Intersection of Two Surfaces

Result: Intersection of Two Surfaces
input: S_A, S_B: Two surfaces to be intersected
if *S_A is empty or S_B is empty* **then**
  return
else if *S_B is larger than S_A* **then**
  Swap S_A and S_B;
end
if *S_A is a leaf node* **then**
  Compute-SSI-Curve(S_A,S_B);
else if *S_A and S_B have no overlap in their bounding volumes* **then**
  return
else
  S_1, S_m, S_2: the left, middle, right child nodes for S_A;
  Intersect(S_1, S_B); Intersect(S_m, S_B); Intersect(S_2, S_B);
end

6.3 Construction of Intersection Curve Segments

The intersection of two surface patches S_{ij} and S_{kl} from non-miter leaf nodes can be constructed using a conventional SSI algorithm. Thus we focus on the pair of nodes, where one contains a miter point. (The chance of both being miter nodes is extremely low; thus we skip this highly

unusual case in the following discussions.) The center of an ϵ-ball for $S(Q(\alpha, \beta))$ can be projected to the subpatch S_{kl} of the other node [21]. Based on the result, we can decide on which side of S_{kl} the miter point (approximated by the ϵ-ball center) is located, including the case of lying on the subpatch.

The self-intersection curve (in the xyz-space) near the miter point is an almost linear segment with the miter point at its open end. By intersecting the line segment against the subpatch S_{kl}, we can decide the planar lution curves in a neighborhood of the miter-quadrangle Q. Note that the solution curves for a local self-intersection (meeting at the miter point as shown in Figure 4.6) are constructed in a preprocessing step and stored in the grid structure of subpatches. To complete the construction of self-intersection curves, we need to add an additional solution curve from a global self-intersection of S_{kl} against the neighborhood of Q.

When the line segment has no intersection with S_{kl}, the solution curves (meeting at the miter point) for a local self-intersection and the solution curve for a global self-intersection also have no intersection (see Figure 6.1(a)). On the other hand, if there is an intersection with S_{kl}, the intersection point should be of the form $S(u_*, v_*) = S(s_*, t_*)$, where the two locations (u_*, v_*) and (s_*, t_*) are on their respective (u, v) and (s, t) solution curves. In this case, as shown in Figure 6.1(b), the solution curve for a global self-intersection with S_{kl} will intersect the (u, v) and

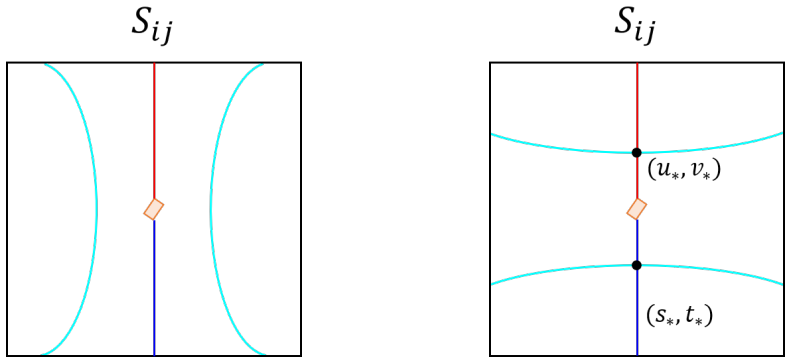


Figure 6.1: Arrangement of solution curves: (a) when the local self-intersection curve has no intersection with S_{kl}, and (b) when the local self-intersectio n curve has a global intersection with S_{kl}.

(s, t) solution curves for the local self-intersection.

The illustration in Figure 6.1 corresponds to the example of Figure 8.8(a), with a hyperbolic type solution curve. In Figure 8.8(b), an elliptic type solution curve produces an 8-figured self-intersection curve in the Euclidean space. In a similar way, a parabolic type solution curve produces a self-intersection curve of $\propto$ shape. The type of each solution curve can be determined by the number of branches in the 1-ring neighborhood of S_{ij}. We skip the details of the case analysis.

6.4 Summary

Surface self-intersection is the collection of all points where two different surface parameters map to the same location in the Euclidean space. The

miter points make the surface self-intersection problem more difficult because there is an almost tangential self-intersection and the surface normal directions change dramatically around each miter point, located at the open endpoints of the self-intersection curve.

We have presented a new approach to the self-intersection problem for freeform surfaces, using a regional representation of miter points in the parameter space. Nevertheless, the self-intersection curve near a miter point has an almost linear shape in the Euclidean space. The geometric uncertainty can be confined to the miter quadrangles in the parameter space. Based on this observation, we can decide a local topology of the self-intersecting surfaces at miter points.

Chapter 7

Trimming Offset Surfaces with Self-Intersection Curves

In order to use offset surfaces meaningfully in CAD/CAM applications, we must trim off the self-intersection and redundancies of offset surfaces. Detecting the self-intersection of an offset surface is an important research issue, but it has long been considered one of the most challenging problems because the offset of a rational surface is non-rational in general [58, 69]. Furthermore, global and local singularities such as isolated points, cusps, ridges, and self-intersection curves make the offset surface trimming problem considerably more difficult [70], because they introduce serious numerical instability in associated geometric computations. As the result, many conventional methods [71, 70, 72] are

based on approximation techniques and they computed the offset self-intersection curves for $O(u,v) = O(s,t)$ in the (u,v,s,t)-space by applying Runge–Kutta iterations to certain differential equations derived from the self-intersecting offset surfaces. Unfortunately, they often include large error in the solution in the vicinity of singularities. In this chapter, we introduce how we overcome these computational difficulties by using ternary hybrid BVH tree and osculating toroidal patches.

7.1 Offset Surface and Ternary Hybrid BVH

Given a regular freeform surface $S(u,v)$, the offset surface $O(u,v)$ is defined as follows:

$$O(u,v) = S(u,v) + d \cdot N(u,v), \tag{7.1}$$

where d is a positive or negative offset distance value and $N(u,v)$ is a unit normal vector of $S(u,v)$:

$$N(u,v) = \frac{S_u(u,v) \times S_v(u,v)}{\|S_u(u,v) \times S_v(u,v)\|}. \tag{7.2}$$

In Figure 7.1, the blue progenitor surface makes the left red offset surface with a positive offset radius and makes the right red offset surface with a negative offset radius. As our surface self-intersection algorithm

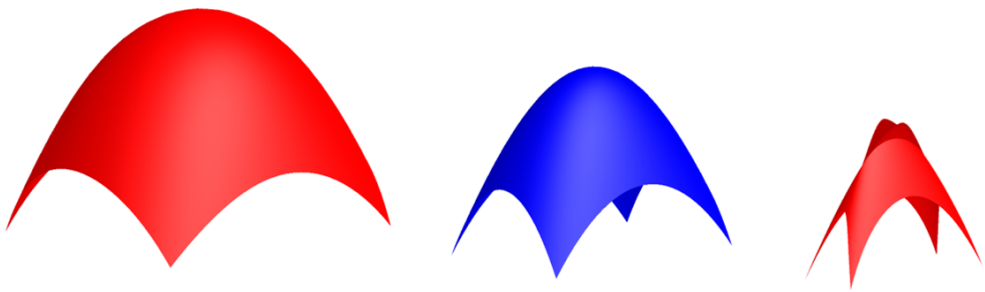


Figure 7.1: Blue surface is a progenitor surface, and red surfaces are offset surfaces.

with the hybrid BVH with ternary structure (for $S(u, v)$) works reasonably well, we can expect it would also work in an efficient and robust way as well in the offset surface. We start with the bounding volume technique for a hybrid BVH. From the hierarchy of bilinear surfaces, we approximate $S(u, v)$, mainly using the four corner points of each surface subpatch. Given u and v, we calculate four corner points of a subpatch of $O(u, v)$ ($O_{i,j}(u, v)$) by Equation 7.1, and also build a ternary hybrid BVH tree for $O(u, v)$. As discussed in Section 4.2, there are osculating toroidal patches on the leaf nodes of BVH. Simply saying, the osculating toroidal patch for a leaf node can be constructed using the principal curvatures of $O_{i,j}(u, v)$. However, this is non-trivial in case of singularity. Using the geometric properties of torus, we compute a toroidal patch that approximates $O_{i,j}(u, v)$ while avoiding this problem. The offset of a torus can be computed simply by adding offset radius d to the minor radius r of the

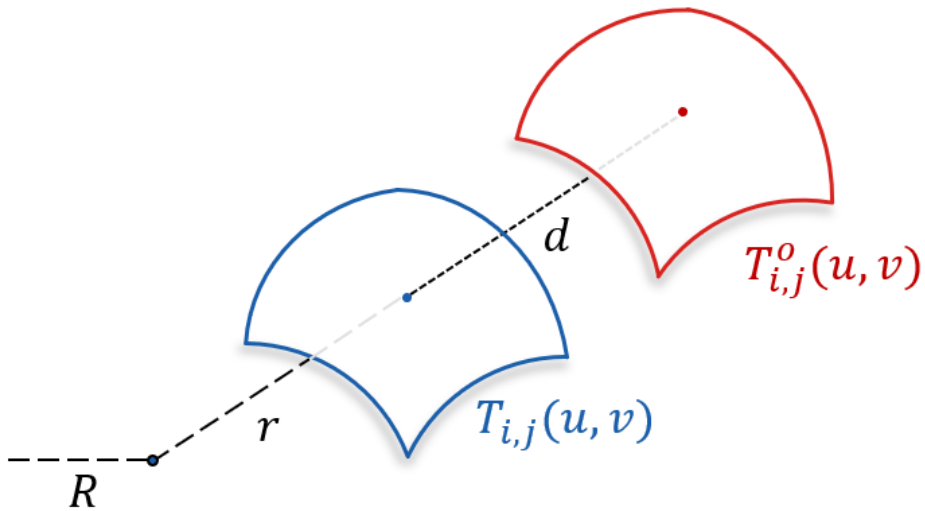


Figure 7.2: Toroidal patch and their offsets.

torus. Furthermore, since we build an osculating toroidal patch using the principal curvatures of $S_{i,j}(u,v)$, the surface normal at (u,v) is the same as that of the toroidal patch. This means that the offsets of the osculating toroidal patch for $S_{i,j}(u,v)$ approximates $O_{i,j}(u,v)$. In Figure 7.2, a red toroidal patch $T^o_{i,j}(u,v)$ is the offset of a blue toroidal patch $T_{i,j}(u,v)$.

7.2 Preprocessing

To build a ternary hybrid BVH tree, we first make uniform subdivision of an offset surface $O(u,v)$ into subpatches $O_{ij}, (i,j = 1, \cdots, 144)$. Note that we take 144 as our $8N \times 8N$ rule in Section 4.2, which can be extended to 1024, using split ratios $1024 = 384 + 256 + 384$ and $384 =$

144 + 96 + 144. After the subdivision for leaf nodes, we need to check all leaf nodes to deal with the singularities with two conditions, *normal flips* and *near singular*. Hong et al. [56] simplified the equation of determining the normal flipping of the offset surface as follows:

$$f(u,v) = \left(\kappa_1 - \frac{1}{d}\right)\left(\kappa_2 - \frac{1}{d}\right), \tag{7.3}$$

where κ_1 and κ_2 are the principal curvatures of $S(u,v)$. When $f(u,v)$ is negative, the normal vector of $O(u,v)$ is opposite to the normal of $S(u,v)$. With this geometric condition, we can categorize leaf nodes to three types:

- *Fully flipped* : $f(u,v) < 0$ for the whole region of a leaf node; the normal of leaf node flips from the progenitor surface.
- *Partially flipped* : $f(u,v)$ has both positive and negative values; some regions of leaf node is flipped but the others are not.
- *Not flipped* : $f(u,v) > 0$ for the whole region of a leaf node; the normal of the leaf node is the same as the progenitor surface.

Fully flipped leaf nodes must be excluded and not included in the trimmed offset surface [23], so we can ignore them in the offset surface self-intersection, which often appear in a large area on the surface (u,v)-domain. Moreover, when an internal node of the BVH tree contains all

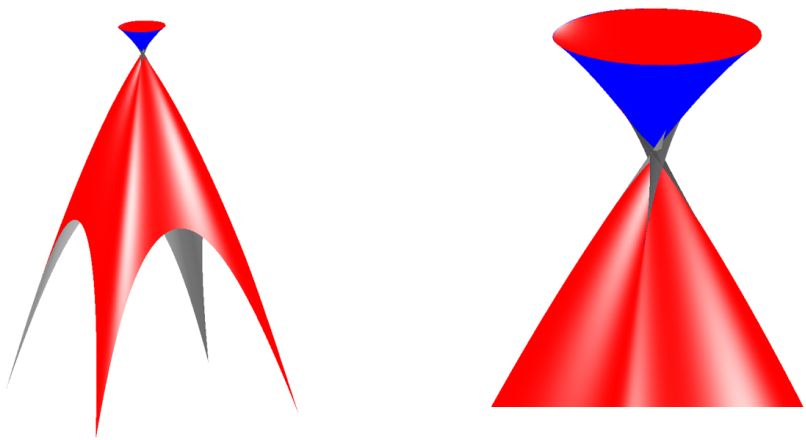
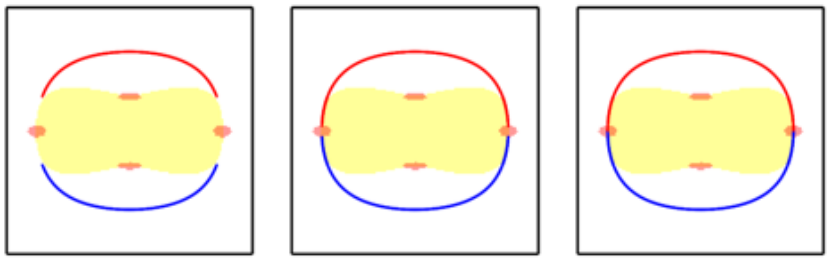


Figure 7.3: Offset surface with normal flipping.

fully flipped regions, we do not need to generate children nodes and stop recursive subdivision, which makes the self-intersection algorithm faster. *Not flipped* leaf nodes are ordinary nodes, so they are ready to execute an intersection algorithm. In Figure 7.3, there is an offset surface without trimming redundancies. *Fully flipped* region is shown in blue, and *not flipped* region is shown in red. The problem is *partially flipped* leaf nodes. They are located near the boundary of *fully flipped* leaf nodes. Not flipped regions in the *partially flipped* leaf nodes can be used for an tracing an intersection curve. However, prebuilt osculating toroidal patches can not bound its normal correctly and maybe invalid, because the angle of the normal vector of *partially flipped* leaf node increases by more than 180°. Even if the approximation is done reasonably well, the



(a) **(b)** **(c)**

Figure 7.4: (a) Solution curve with transversal intersections; (b) expand solutions with instant osculating toroidal patch matching; (c) final solution curve with contouring constant curvature.

parameter matching between the toroidal patch and the surface subpatch may go awry, so we store *partially flipped* leaf nodes separately.

Unit normal vector $N(u, v)$ is well-defined for a regular surface $S(u, v)$ as the u and v-partial derivatives $S_u(u, v)$ and $S_v(u, v)$ are non-parallel, and the vector $S_u(u, v) \times S_v(u, v)$ never vanishes. Even if the surface $S(u, v)$ is given as a regular surface, the offset surface $O(u, v)$ may have certain areas where the partial derivatives $O_u(u, v)$ and $O_v(u, v)$ become almost parallel. This singularity occurs generically around the area where the surface $S(u, v)$ has curvature close to $\frac{1}{d}$, and the offset is taken to the concave side of the surface [70]. Therefore, we need to check all leaf nodes, where the corresponding progenitor surface has curvature close to $\frac{1}{d}$. We store them separately as *near-singular*.

7.3 Merging Intersection Curve Segments

After collecting *not flipped* leaf node pairs whose bounding volumes are colliding, we can compute self-intersection solution curve(s) by doing torus-torus-intersection on the osculating toroidal patches. Since all intersections that occur in this process are transversal intersections, our algorithm for surface self-intersection can be used. Intersection results are usually given as a set of open curve segments in the (u, v)-domain. As we separate *partially flipped* and *near-singular* leaf nodes, the solution curves are constructed only partially. In Figure 7.4(a), the solution curves are shown as a red (u, v)-solution curve and a blue (s, t)-solution curve and their construction is terminated at the boundary of a *fully flipped* region, shown in yellow. In general, the solution curves encounter a *partially flipped* nodes before *near-singular*. In this case, we make osculating toroidal patches for $O(u, v)$ and $O(s, t)$ on the fly, instead of using prebuilt osculating toroidal patches. Since the osculating toroidal patch is made with the curvature information at the endpoint of an intersection curve segment, it approximates the surface locally better than the prebuilt toroidal patches in the limited area of Figure 7.5(a). After computing their intersection in the xyz-space for a short curve segment, we project the result back to the corresponding segments in the (u, v) and (s, t)-domains. When the parameter matching $(x, y, z) - (u, v) - (s, t)$ is correct

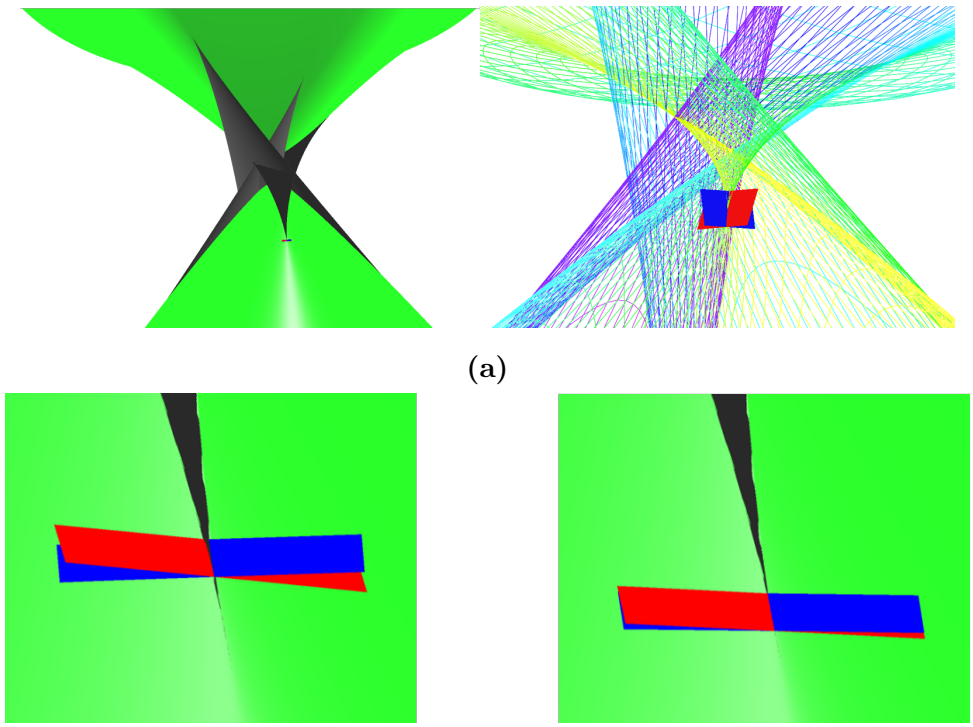


Figure 7.5: (a) Build toroidal patch instantly with endpoint of solution curve; (b) Project the result back to the parameters and repeat the procedure.

with the osculating toroidal patches on the fly, we accept the result to the solution curve and continue the same process (in Figure 7.5(b)). Note that the current endpoint's principal curvature is sufficiently close to $\frac{1}{d}$.

We finally reach to the *near-singular* region (shown in red in Figure 7.4(b)) where local self-intersections occur almost everywhere and the parameter matching between (u, v) and (s, t) becomes highly unreliable,

and the toroidal patch matching method becomes not useful anymore. The *near-singular* region in the xyz-space is very tiny (usually $< 10^{-5}$ in our examples), so we can consider this region as a miter quadrangle (or a miter tube). In such a small neighborhood, we need to connect solution curve segments in a correct topology.

To trace the intersection curve through the singularities, we contour the constant principal curvature of $\frac{1}{d}$[8]. We first do super-sampling in the region between the parameters of endpoints, (u, v) to (s, t), and perform a curvature analysis of progenitor surface. When the principal curvatures of sample points are close enough to $\frac{1}{d}$ ($\epsilon < 10^{-8}$), we trace these sampled points. In most cases, however, the principal curvatures at sample points perpendicular to the direction of the solution curve is positive on one side and negative on the other. Then we pick two sample points where the sign changes between them and perform a binary search until we get $\epsilon < 10^{-8}$. After finishing the contouring of the constant principal curvature, we obtain a set of parameters, but the $(u, v) - (s, t)$ matching has not been made yet. From the perspective of the minimum Hausdorff distance computation, we match the $(u, v) - (s, t)$ parameters and decide a miter point (in Figure 7.4(c)). Finally, we get all self-intersection points. At the end of the intersection algorithm, we approximate these segments using G^1-biarcs and measure the approximation error.

7.4 Summary

The self-intersection computation for offset surfaces is essential for many important geometric applications. However, it is more complex to deal with than the progenitor surface. Even when the progenitor surface is regular, the offset surface can be singular. We overcome this difficulty by categorizing the *near-singular* region and *normal flipped* region and handling them separately. By separating the singularity regions in a preprocessing stage, we have eliminated the step to create a bounding volume and build a hierarchy structure for the redundant areas. It makes our algorithm more efficient and robust. We construct a solution curve not only using prebuilt toroidal patches with ternary structure, but also constructing toroidal patches on the fly at the endpoint of solution curves. Even when local self-intersections occur almost everywhere, we contour the constant curvature line and decide a correct local topology in a reliable way.

Chapter 8

Experimental Results

8.1 Surface-Surface-Intersection

We have implemented the proposed SSI construction algorithm in C++, on an Intel Core i9-9900KF 3.6GHz Windows PC with a 64GB main memory. To demonstrate the effectiveness of the BVH-based approach, we have tested the SSI algorithm on a large set of examples, including those of intersecting two almost identical surfaces.

The BVH construction on a bicubic Bézier surface typically takes less than one second in the preprocessing step, when the resolution of 64×64 is used for sampling the uv-parameter domain of the surface. The construction time increases quadratically as we take finer resolutions along both u, v-directions. Thus the BVH construction would take less

Table 8.1: BVH storage for a bicubic Bézier surface (in MB).

Height	Convex	AABB	RSS	Tori
4	1.599	0.112	0.142	0.157
5	6.374	0.447	0.493	0.605
6	25.414	1.792	2.217	2.334
7	101.899	7.137	9.123	9.395
8	413.189	28.724	36.548	37.773

than one minute even for a high resolution of 512×512. Nevertheless, there is a certain limitation on the maximum resolution, mainly due to the fixed memory space of a hardware system. On the other hand, the SSI computation takes far less time than the BVH construction.

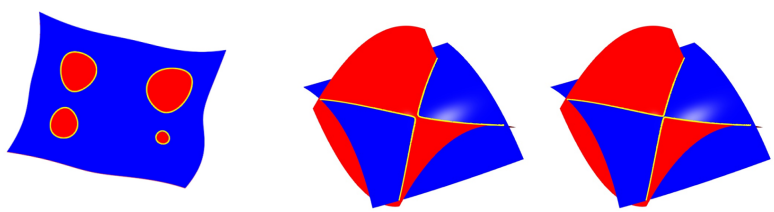
Table 8.1 reports the memory size for storing BVH trees for a bicubic Bézier surface, where the leaf level contains a total of $2^H \times 2^H$ nodes, for the tree heights $H = 4, \cdots, 8$. The convex hull tree contains, in each node, the convex hull boundary for the Bézier control points, which is stored as a triangular mesh. The exact number of triangles in each convex hull changes depending on the shape of a Bézier subpatch. On the other hand, the sizes of AABB and RSS trees are fixed, depending only on the height of each BVH tree. The convex hull tree takes much more memory space than others. In principle, the osculating toroidal patches can be used for any type of BVH tree for freeform surfaces. Their sizes are reported separately in the rightmost column of Table 8.1, providing information on additional memory costs in case they are adapted to a BVH structure.

Table 8.2: Construction time for complete BVH trees of $H = 6$ (in seconds).

Surfaces	# Béziers	Convex	AABB	RSS	Tori
GK 1	578	227.7	50.9	142.3	31.4
GK 2	578	212.7	50.6	140.6	35.0
GK 3	578	258.0	51.2	136.6	38.1
GK 4	2	0.6	0.2	0.5	0.1
GK 5	305	101.7	27.0	71.8	21.2
GK 6	273	106.8	23.9	64.5	18.0
Overlap	8	3.4	0.9	2.0	0.5
Saddle 1	2	0.7	0.2	0.5	0.1
Saddle 2	2	0.7	0.2	0.5	0.1

Table 8.2 reports the BVH construction time for the surfaces in the test examples of Figures 8.2–8.1. The convex hull tree is also the most expensive in the construction time as well. We have measured the computing time using only one CPU, which seems to produce more stable timing results than using multi-core CPUs.

Figure 8.2 shows the results from testing our BVH-based algorithm on six examples of Grandine and Klein [3]. (We have not included the



(a) **(b)** **(c)**

Figure 8.1: (a) Two almost overlapping surfaces, and (b)–(c) two saddle surfaces.

Table 8.3: Tree traversal time using BVH trees of $H = 6$ (in ms), and pairs of overlapping leaf nodes in complete BVH traversal with $H = 4$.

Surfaces	# Béz Pairs	Traversal time			Overlapping pairs		
		Convex	AABB	RSS	Convex	AABB	RSS
GK 1	(289,289)	134.07	39.14	44.35	12,008	71,533	16,367
GK 2	(289,289)	107.99	27.31	32.18	6,834	20,284	7,941
GK 3	(289,289)	102.45	26.31	29.25	3,547	6,276	5,034
GK 4	(1,1)	0.06	0.01	0.02	17	34	19
GK 5	(16,289)	6.85	1.89	1.77	1,228	4,009	1,748
GK 6	(81,192)	16.58	4.91	4.74	572	2,042	618
Overlap	(4,4)	2.64	0.48	0.80	522	1,049	572
Saddle 1	(1,1)	0.57	0.07	0.20	111	284	251
Saddle 2	(1,1)	0.89	0.08	0.20	125	286	250

other four examples of [3], where one surface is given as a simple plane.) The almost overlapping example of Figure 8.1(a) is taken from Choi [73], which is generated by gradually scaling down two intersecting surfaces along the vertical z-direction until they become almost (visually) indistinguishable. In this scaling process, all points on the SSI curves preserve the same xy-coordinates and the SSI topology is thus invariant. Two examples of intersecting almost tangential saddle surfaces are shown in two other examples of Figure 8.1, where the last example of Figure 8.1(c) is the case of missing an X-junction only slightly and thus forming two separate branches in an orthogonal direction to those of the second example in Figure 8.1(b).

Table 8.3 reports the time taken in the BVH tree traversal for each example of Figures 8.2–8.1. The convex hull tree is again the worst in the

traversal time. Nevertheless, we have yet to compare the total tracing time for constructing all SSI curve segments. Table 8.3 also may give some idea on the relative tracing time of each method by comparing the number of pairs of overlapping leaf nodes when the BVH tree traversal is made all the way to the leaf level (i.e., without checking the one branch condition in the internal nodes). This way of complete traversal may produce a larger number of pairs than necessary, which is the reason why we check only with the height $H = 4$, until then not many early exits would be made at the higher levels $h = 1, 2, 3$. Though the AABB tree is often faster in the tree traversal (mostly for simple test examples) than the RSS tree, it is mainly due to the simple AABB overlap test, which takes less time than comparing two rectangles in general orientations in the xyz-space.

According to Table 8.3, the convex hull tree generates the tightest overlap tests, which saves much time wasted for unnecessarily checking the potential overlaps among surface subpatches. Though the RSS tree generates more redundant pairs than the convex hull tree, the overlap test for toroidal patches is simpler than for freeform surface patches. Considering all these, the RSS tree (combined with osculating toroidal patches) provides a good compromise for both space and time efficiency in the SSI computation for freeform surfaces.

Now, we consider a more serious issue, the robustness of our method,

by generating some highly non-trivial test examples of freeform surfaces which intersect tangentially almost everywhere. Heo et al. [1] report some interesting results from such a test, where two almost coaxial cylinders were intersected, with the angle between the two cylinders being $\theta = 10°, 1°, 0.1°$, and $0.01°$. In Figure 9 of Heo et al. [1], the X-junctions are slightly missed at a small angle $\theta = 0.1°$, and then completely missed at a smaller angle $\theta = 0.01°$. We repeat the same test using our method. Figure 8.3 reports the test results, including some more challenging tests with $\theta = 0.0001°$, and $0.00002°$. The first row of Figure 8.3 shows a red cylinder approximated (within an error bound $\epsilon = 10^{-10}$) by four bicubic polynomial Bézier surfaces, where the parallel lines are degree-elevated to cubic Bézier curves. Rotating the cylinder axis by angle θ, we have generated blue cylinders for more and more difficult test examples. Two cylinders of the same radius intersect while sharing two exact tangential intersection points. In these test results, the tangential intersection curves are shown as yellow patches. The second row shows only the boundary curves of the yellow patches in the tangential intersection regions, and the zoomed views on their X-junctions are given in the third row. In Figure 8.4, a tiny shape change is made to the blue cylinders by rotating their bicubic Bézier surfaces about the axis of the cylinder by angle $45°$. (Note that the maximum error in the Bézier surface approximation occurs at the almost tangential intersection points.) After that, we repeat the

same tests, but the results are shown for a different set of angles θ.

Table 8.4 reports the number of overlap pairs of leaf nodes in each of the cylinder examples shown in Figure 8.3. (The performance of AABB is highly dependent on the orientation of cylinder, which may explain the unusual result for the case of angle $\theta = 0.00002°$. In this experiment, we mitigate the influence of orientation by taking the average of three tests using the axis directions along $(1, 0, 0), (1, 1, 0), (1, 1, 1)$.) The AABB tree seems impractical in most of these test cases. On the other hand, the RSS tree looks impractical only after the angle gets smaller than $\theta = 0.0001°$, where the overlap occurs almost everywhere in the two cylinders. Nevertheless, even in the case of $\theta = 0.00002°$, using the osculating toroidal patches, we were able to detect the branching structure of the intersection curve reliably as shown in the rightmost example of Figure 8.3.

In theory, the convex hull tree might have generated a smaller number of overlap pairs; nevertheless, due to the large memory space required for the construction of intermediate convex hulls, we have experienced difficulty generating the convex hull tree for $H = 9$. As we have made some success in the challenging case of $\theta = 0.00002°$, with more than 5M overlap pairs, any BVH tree combined with the osculating toroidal patches would also work for this test.

The yellow patches in Figure 8.3 are ideally the surface subpatches

Table 8.4: Pairs of overlapping leaf nodes for two cylinders with $H = 9$.

Angles	AABB	RSS
0.01°	2,667,043	35,572
0.001°	2,806,193	242,521
0.0001°	3,062,665	2,005,799
0.00002°	4,376,814	5,483,794
0.00001°	5,807,194	5,510,081

which are within a certain one-sided Hausdorff distance $\delta > 0$ to the other surface, the exact computation of which would require the intersection of a surface with the $\pm\delta$-offsets of the other surface. This problem seems to be more difficult than the problem of intersecting two regular surfaces. In the current work, we take uniform samples from the osculating toroidal patch of one node to the osculating torus of the other overlap node, and collect those within a tolerance $\delta = 10^{-8}$ to the torus. The cylinder example is somewhat artificial as the cylinder is a simple surface by itself. In Figure 8.5, we repeat the same robustness test to a saddle surface taken from the examples of Figure 8.1.

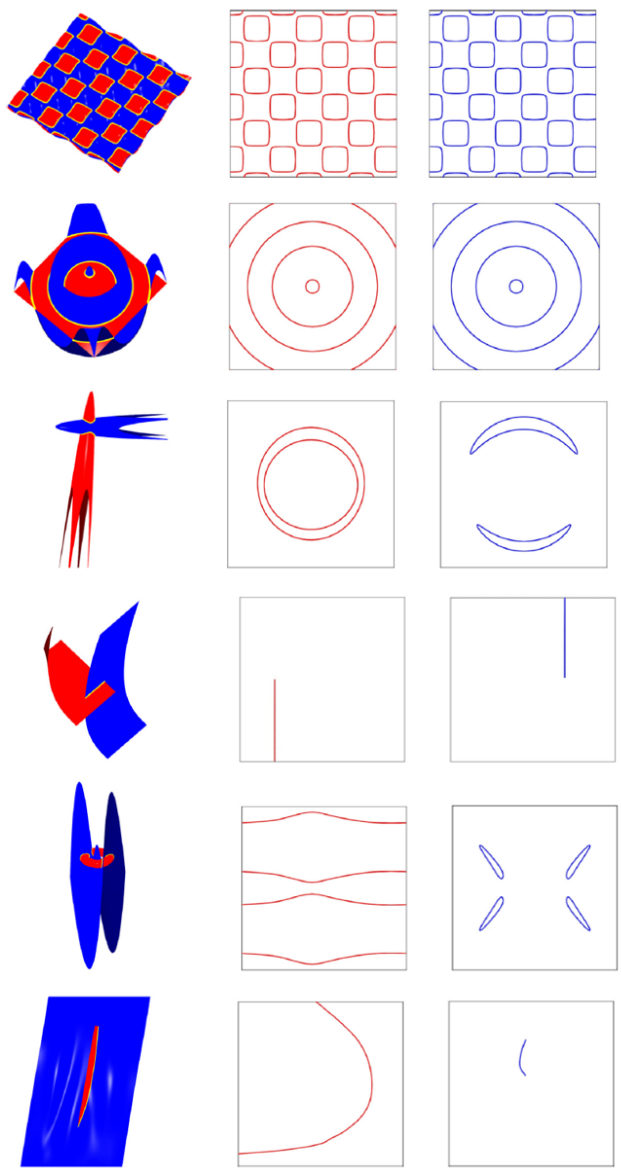


Figure 8.2: Examples from Grandine and Klein [3]; the leftmost column shows the results of intersecting two freeform surfaces and the rightmost two columns show the intersection curves in the parameter domains of the red and blue surfaces, respectively.

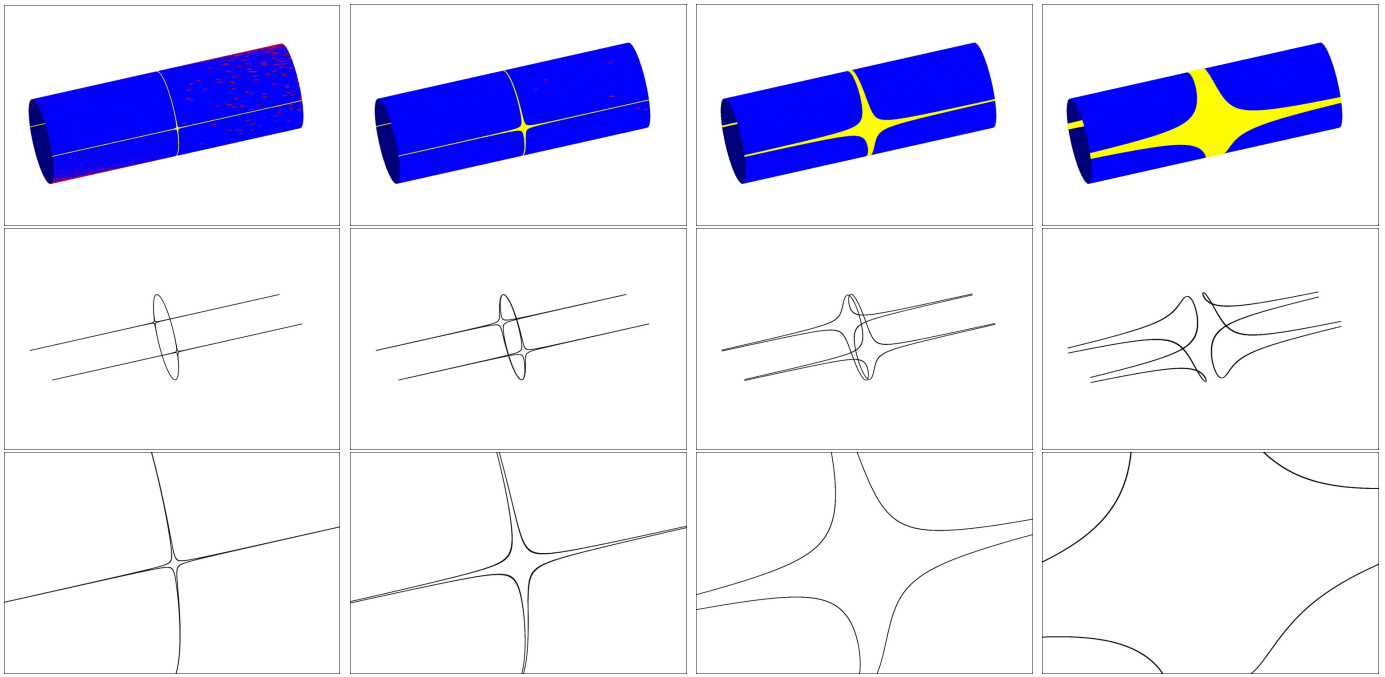


Figure 8.3: Tangential intersection of two almost identical cylinders, where their two axes make a small angle $\theta = 0.01°, 0.001°, 0.0001°$, and $0.00002°$, from left to right; the same tolerance $\delta = 10^{-8}$ was used for all test examples.

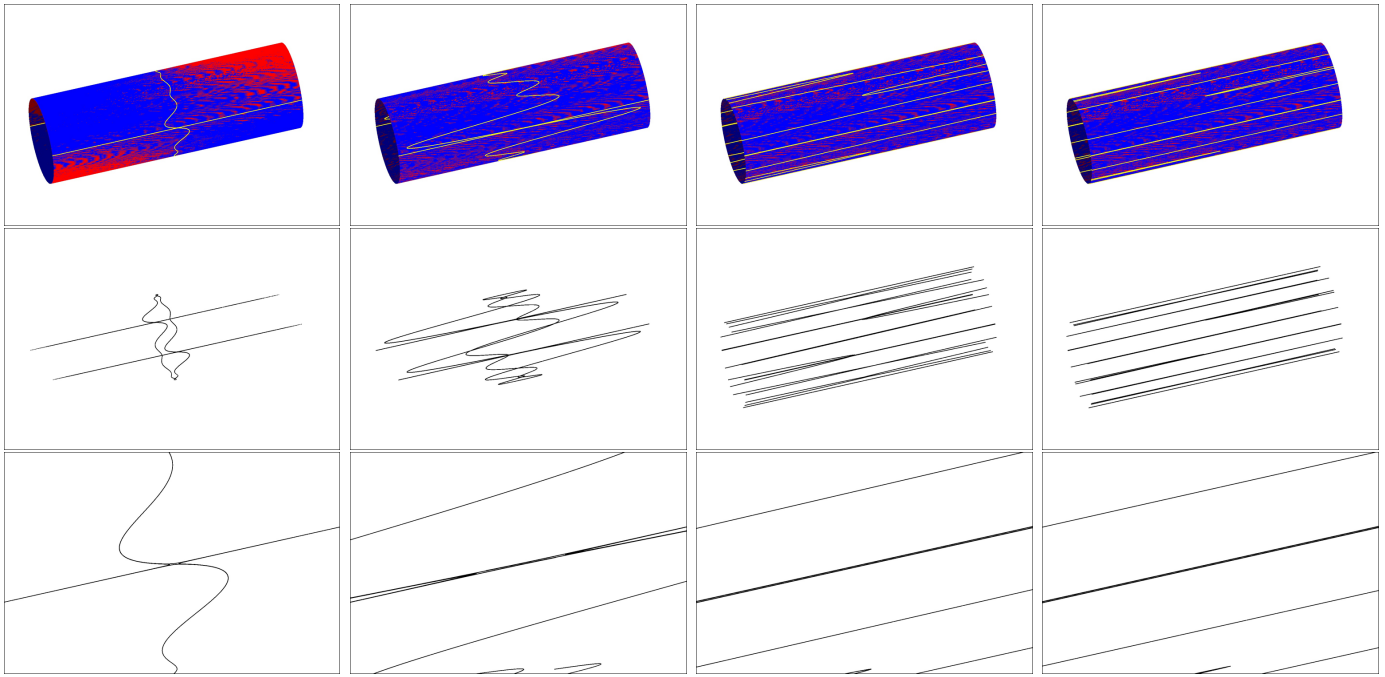


Figure 8.4: Rotating bicubic polynomial Bézier surfaces approximating (within a maximum error bound $\epsilon = 10^{-10}$) the red cylinder by angle 45° about the axis of the cylinder and then intersecting with the blue cylinder, using the angles $\theta = 0.1°, 0.01°, 0.001°, and 0.0001°$.

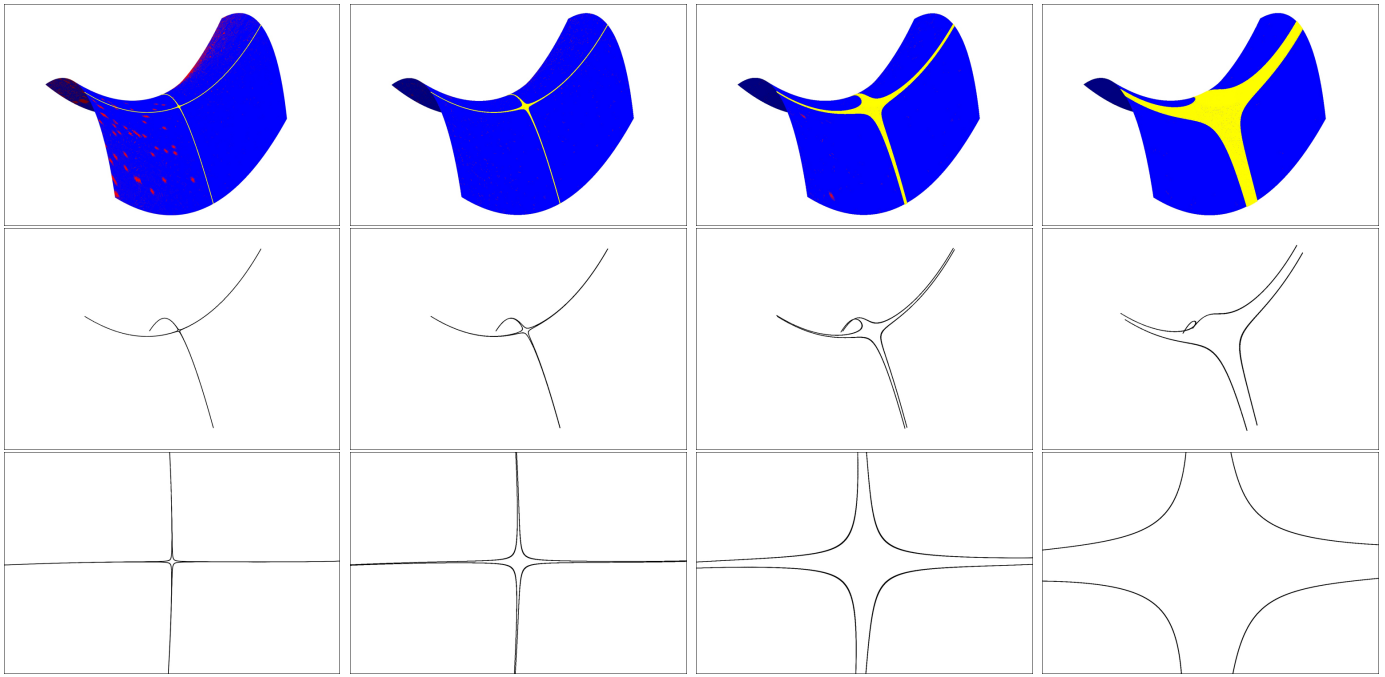


Figure 8.5: Tangential intersection of a saddle surface with its rotation about a normal line by a small angle $\theta = 0.01°$, $0.001°$, $0.0001°$, and $0.00002°$.

Table 8.5: BVH construction, traversal, and surface intersection time (in ms) and the number of pairs of overlapping leaf nodes.

Examples	Construction	Traversal	Intersection	# Overlap pairs
(A)	3,635	392	1,084	599
(B)	1,902	260	1,133	773
(C)	1,872	257	1,251	662
(D)	11,113	1,111	6,960	3,629
(E)	10,980	1,238	5,133	4,858
(Miter-A)	3,616	451	45,610	1,730
(Miter-B)	1,986	257	11,400	774
(Miter-C)	1,828	296	28,522	1,374

8.2 Surface Self-Intersection

8.2.1 Regular Surfaces

We have implemented the proposed self-intersection algorithm in C++, on an Intel Core i7-10700K 3.8GHz Windows PC with a 128GB main memory. To demonstrate the effectiveness of the proposed approach, we have tested the algorithm on several test examples, including three freeform surfaces with miter points on their self-intersection curves.

Figure 8.6 shows five freeform surfaces, each with the self-intersection curve (in yellow line) in the Euclidean xyz-space and the corresponding (u, v) and (s, t)-solution curves (in red and blue lines) in the parameter domain. These five surfaces contain no miter point. Consequently, the construction of their self-intersection curves is essentially reduced to the problem of computing global self-intersections.

More challenging test examples are given in Figure 8.7, where three surfaces are shown with miter point(s) on their self-intersection curves. The miter points are shown as black dots located at the tips of the self-intersection curves in the Euclidean space. In the parameter domain, the miter points (also represented as black dots) appear as the junction points where the corresponding red and blue solution curves meet. The regional representations of some miter points are also shown as green rectangles.

Table 8.5 reports some statistics on the self-intersection curve construction for the eight example surfaces in Figures 8.6–8.7. In the middle three columns, the computing times are given in milliseconds, for the BVH construction, the BVH tree traversal for overlap tests, and the handling of overlapping leaf nodes for the construction of self-intersection curves. The rightmost column reports the number of pairs of overlapping leaf nodes, actually processed in the final intersection stage. Compared with the surfaces with no miter points, the self-intersection with miter points takes considerably (approximately 10–40 times) more computing time. The extra computational effort will eventually pay off when we deal with non-trivial geometric decision problems near the miter points, an example of which we briefly discuss below.

In Figure 8.8, two local surface patches are extracted from small neighborhoods of the miter points, which are on the (Miter-B) and (Miter-C) surfaces of Figure 8.7. Each patch is intersected with three par-

allel planes as shown in the left, middle, and right columns of Figure 8.8. The top surface of Figure 8.8(a) intersects in two separate branches (as shown on the left and right), or in an X-shaped self-intersecting curve (in the middle) as the result of a tangential intersection at the miter point (of the (Miter-B) surface in Figure 8.7). Regarding the three topological types of the plane-surface intersection curve, as discussed in Section 6.3, a correct classification can be made based on the result of intersecting the plane against a short line segment (with the miter point at an endpoint).

The plane intersections with the (Miter-C) surface of Figure 8.7 produce even more interesting results. The plane-surface intersection curves are 8-figures, the miter point, or empty, depending on whether the plane intersects the short line segment (approximating the surface self-intersection curve near the miter point) at an interior point, tangentially at the miter point, or at no point. Note that the plane-surface intersection appears as a closed loop in the parameter space, which turns into an 8-figured self-intersecting space curve in the Euclidean space, as the result of gluing the two locations (u_*, v_*) and (s_*, t_*) to the same point $S(u_*, v_*) = S(s_*, t_*)$ in the xyz-space.

Though we have considered the intersection of a small surface patch (containing a miter point) against parallel planes (which may not be from the same surface), the basic principle works for the self-intersection with some other parts of the surface. Because of the tiny size of the ϵ-balls, it

Table 8.6: BVH construction, traversal, and surface intersection time of offset surface examples (in seconds).

Examples	Construction	Traversal	Intersection
(A)	7.97	0.87	3.52
(B)	5.87	0.56	1.38
(C)	5.73	0.64	1.68

is quite cumbersome to generate an example of generic surface that has a global intersection of a miter neighborhood with some other parts of the same surface. Nevertheless, in some degenerate cases, we need to deal with this special case, a reliable solution with deciding a correct local topology based on the techniques we have introduced in this thesis.

8.2.2 Offset Surfaces

We have implemented the proposed offset surface self-intersection algorithm in the same environment as regular surfaces. To demonstrate the effectiveness of the proposed approach, we have tested the algorithm on three test examples. Figure 8.9 shows three offset surfaces with the parameter domain, and the (u, v) and (s, t)-solution curves (in red and blue lines). Example (A) has no miter point, and the other two have miter points. The miter points are shown as black dots. In the parameter domain, the miter points appear as the junction points in the same way as the miter points for the regular self-intersection surfaces. Table 8.6 reports the computing times for computing offset surface self-intersection

curves. Note that the Example (A) is a B-spline surface, which needs to be converted to a set of Bézier surfaces and thus requires more computing time than the other two examples. Compared with the result of Hong et al. [23], where multivariate equation solver took about 30 minutes to compute the self-intersection curves for the same examples, our new approach completed the whole computation within 10 seconds.

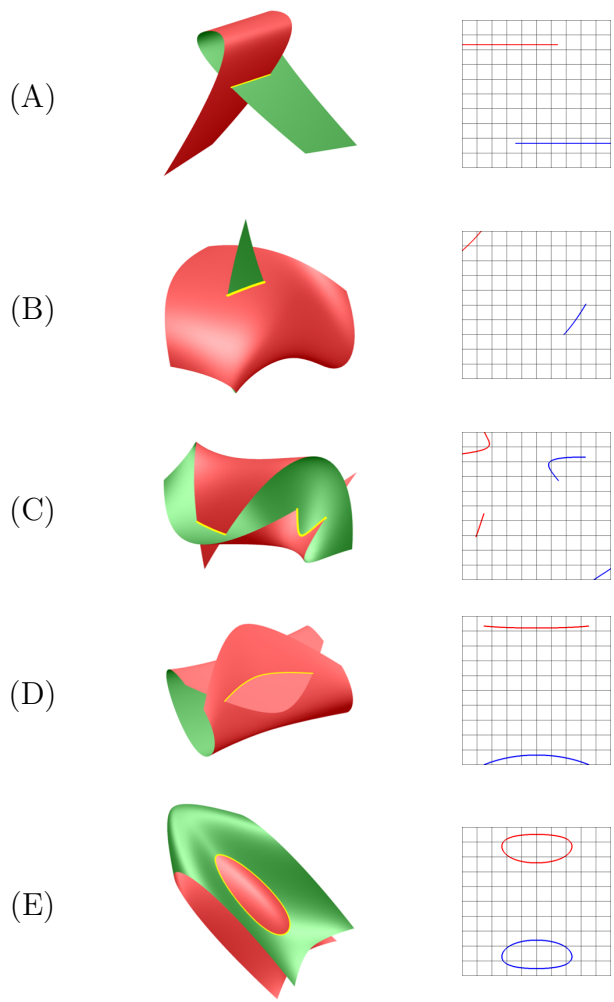


Figure 8.6: Examples of self-intersecting surfaces; the left column shows the results of self-intersecting a freeform surface and the right column shows the intersection curves in the parameter domains of the surfaces, respectively.

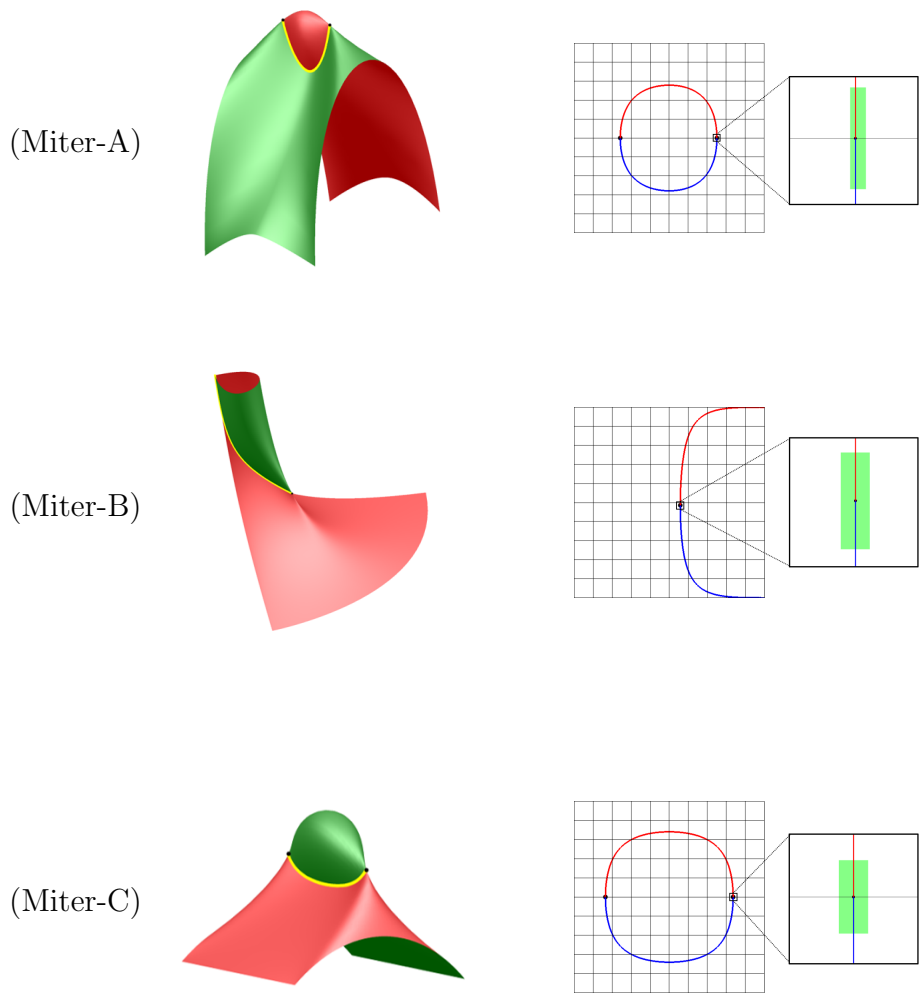
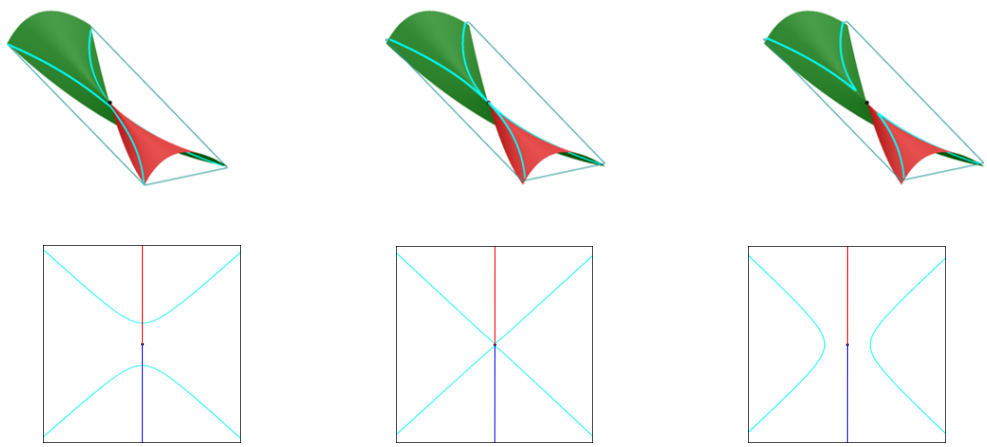
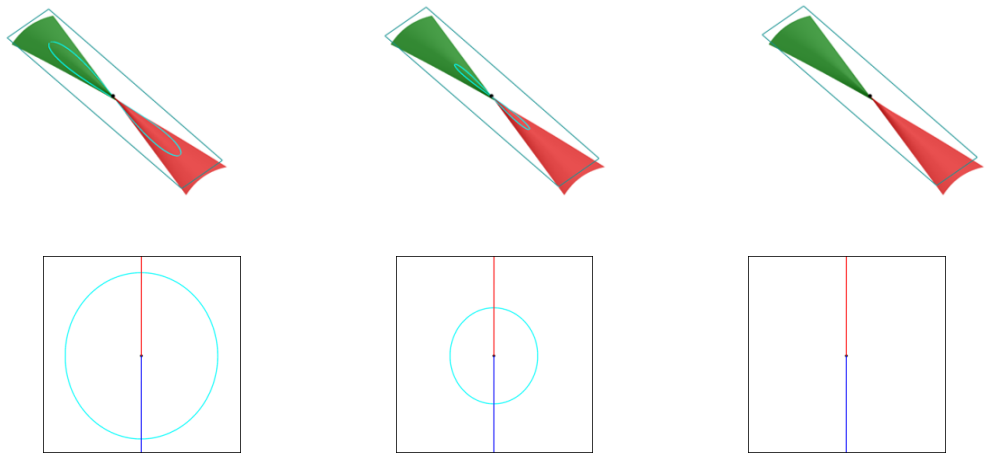


Figure 8.7: Examples of self-intersecting surfaces with miter point(s) on their intersection curves; the left column shows the results of self-intersecting a freeform surface and the right column shows the intersection curves in the parameter domains of the surfaces and zoom-in areas around the miter points, respectively.



(a)



(b)

Figure 8.8: Examples of local self-intersection curve near the miter point.

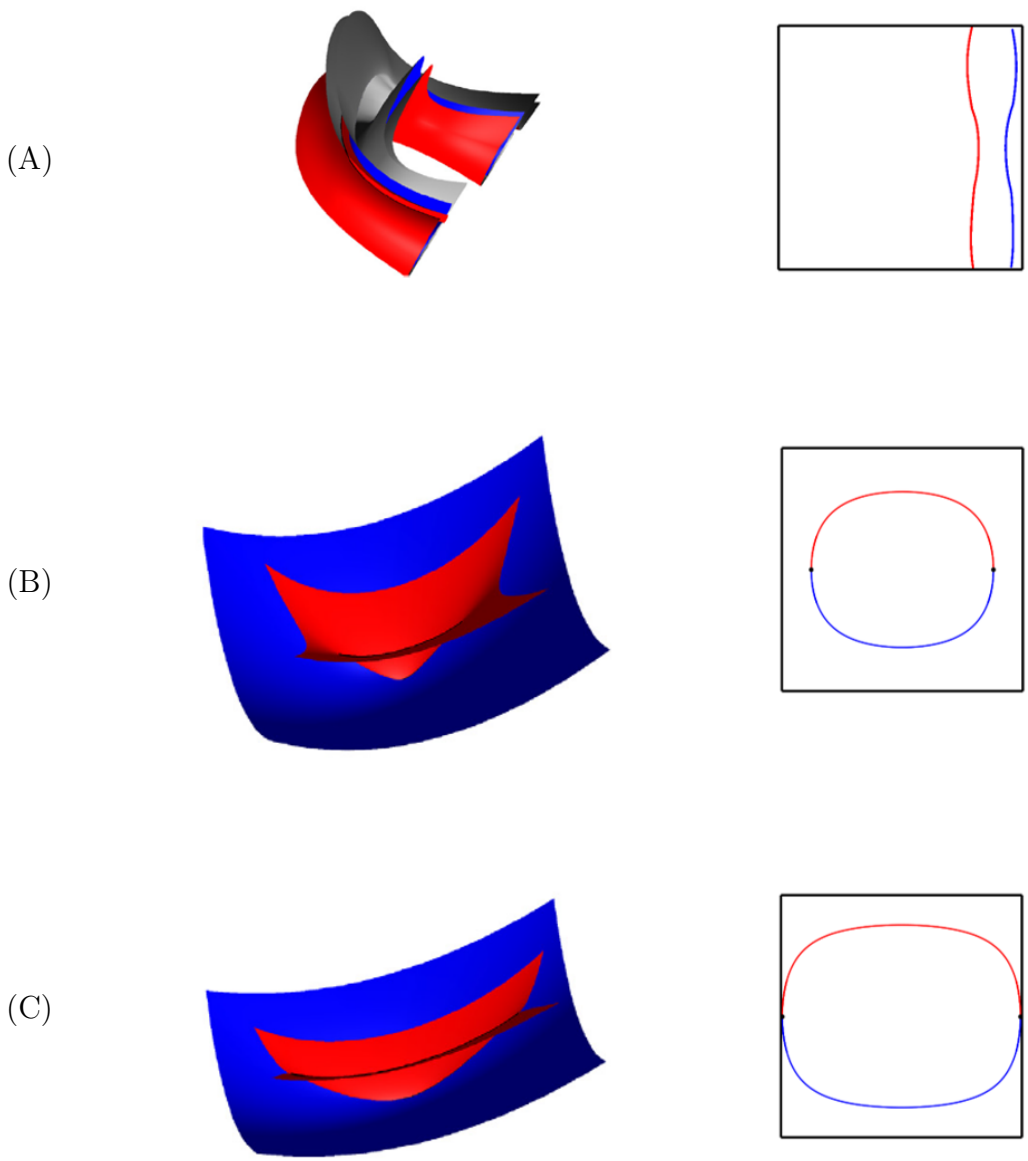


Figure 8.9: Self-intersection curves in the xyz-space and in the uv-domain.

Chapter 9

Conclusion

We have presented a new approach to the intersection of freeform surfaces, including surface-surface-intersection, surface self-intersection, and offset surface self-intersection. Using a hybrid BVH, where the internal nodes contain rectangle-swept spheres(RSS) and the leaf nodes contain osculating toroidal patches, we developed an efficient and robust algorithm for computing the SSI curves. The bounding volumes of internal nodes accelerate the geometric search for the potential pairs of local surface patches that may intersect. Even in highly non-trivial cases, we have shown that the osculating toroidal patches provide stable solutions to the SSI problem.

We have also proposed a new approach to the surface self-intersection problem using a ternary hybrid BVH structure and a regional representation of miter points. We bound miter points with small quadrangles in the parameter domain of the surface. Moreover, the exact location of each miter point is bounded by a tiny(10^{-5} or 10^{-6}) ball in the Euclidean space. The self-intersection curve near a miter point often has an almost linear shape in the Euclidean space, and the geometric uncertainty can be confined to the miter quadrangle in the parameter space. Based on this observation, we can decide a local geometry of the self-intersecting surfaces at miter points.

We have demonstrated that our new approach, using a hybrid BVH and osculating toroidal patches, is quite effective in handling the surface intersection problems. In fact, the proposed approach has great potential in solving other geometric problems that are more general than the surface intersection problems, such as 3D Voronoi diagram construction, convex hull computation, real-time ray tracing, and so on. In the future work, we plan to extend the proposed approach to other geometric operations, solving some non-trivial geometric challenges previously considered extremely difficult.

Bibliography

[1] H.-S. Heo, M.-S. Kim, and G. Elber, "Ruled/ruled surface intersection," *Computer-Aided Design*, vol. 31, pp. 33–50, 1999.

[2] A. Galligo and J. P. Pavone, "Self intersections of a Bézier bicubic surface," in *Proc. of the Int'l Symp. on Symbolic and Algebraic Computation*, 2005, pp. 148–155.

[3] T. Grandine and F. Klein, "A new approach to the surface intersection problem," *Computer Aided Geometric Design*, vol. 14, p. 111–134, 1997.

[4] A. Requicha and H. Voelcker, "Boolean operations in solid modeling: Boundary evaluation and merging algorithms," *Proceedings of the IEEE*, vol. 73, no. 1, pp. 30–44, 1985.

[5] C. M. Hoffmann, *Geometric and Solid Modeling.* San Mateo, CA: Morgan Kaufmann, 1989.

[6] H. Chiyokura, *Solid modeling with DESIGNBASE: theory and implementation*. Boston, MA: Addison-Wesley Longman Publishing Co. Inc., 1988.

[7] J. Hoschek and D. Lasser, *Fundamentals of Computer Aided Geometric Design*. Wellesley, MA: AK Peters, 1993.

[8] N. Patrikalakis and T. Maekawa, *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer, 2002.

[9] T. Sederberg, H. Christiansen, and S. Katz, "An improved test for closed loops in surface intersections," *Computer-Aided Design*, vol. 21, pp. 505–508, 1989.

[10] K.-J. Kim, "Circles in torus-torus intersections," *J. of Computational and Applied Mathematics*, vol. 236, pp. 2387–2397, 2012.

[11] X.-M. Liu, C.-Y. Liu, J.-H. Yong, and J.-C. Paul, "Torus/torus intersection," *Computer-Aided Design & Applications*, vol. 8, pp. 465–477, 2011.

[12] T. Sederberg, D. Anderson, and R. Goldman, "Implicit representation of parametric curves and surfaces," *Computer Vision, Graphics, and Image Processing*, vol. 28, no. 1, pp. 72–84, 1984.

[13] M. Barton and G. Elber, "Spiral fat arcs–bounding regions with cubic convergence," *Graphical Models*, vol. 73, no. 2, pp. 50–57, 2011.

[14] G. Elber and M.-S. Kim, "Geometric constraint solver using multivariate rational spline functions," in *Proc. of the 6th ACM Symposium on Solid Modeling and Applications*, ser. SMA '01. New York, NY: Association for Computing Machinery, 2001, p. 1–10.

[15] I. Hanniel and G. Elber, "Subdivision termination criteria in subdivision multivariate solvers," *Computer-Aided Design*, vol. 39, pp. 369–378, 2007.

[16] M.-S. K. S. Son, S.-H. Yoon and G. Elber, "Efficient minimum distance computation for solids of revolution," *Computer Graphics Forum*, vol. 39, pp. 535–544, 2020.

[17] A. K. I. Hanniel and S. McMains, "Computing the Hausdorff distance between NURBS surfaces using numerical iteration on the gpu." *Graphical Models*, vol. 74, pp. 255–264, 2012.

[18] Y.-J. Kim, Y.-T. Oh, S.-H. Yoon, M.-S. Kim, and G. Elber, "Precise Hausdorff distance computation for planar freeform curves using biarcs and depth buffer," *The Visual Computer*, vol. 26, pp. 1007–1016, 2010.

[19] ——, "Efficient Hausdorff distance computation for freeform geometric models in close proximity," *Computer-Aided Design*, vol. 45, pp. 270–276, 2013.

[20] A. Krishnamurthy, S. McMains, and I. Hanniel, "Gpu-accelerated Hausdorff distance computation between dynamically deformable NURBS surfaces," *Computer-Aided Design*, vol. 43, pp. 1370–1379, 2011.

[21] X.-M. Liu, L. Yang, J.-H. Yong, H.-J. Gu, and J.-G. Sun, "A torus patch approximation approach for point projection on surfaces," *Computer Aided Geometric Design*, vol. 26, pp. 593–598, 2009.

[22] J.-K. Seong, K.-J. Kim, M.-S. Kim, G. Elber, and R. Martin, "Intersecting a freeform surface with a general swept surface," *Computer-Aided Design*, vol. 37, pp. 473–483, 2005.

[23] Q. Hong, "Trimming self-intersections of offset curves and surfaces," Ph.D. dissertation, Dept. of Computer Science and Engineering, Seoul National University, 2020.

[24] C.-C. Ho., "Feature-based process planning and automatic numerical control part programming," Ph.D. dissertation, Dept. of Computer Science, Univ. of Utah, 1997.

[25] C.-C. Ho and E. Cohen, "Surface self-intersection," in *Mathematical Methods for Curves and Surfaces*, T. Lyche and L. L. Schumaker, Eds., 2001, pp. 183–194.

[26] M. do Carmo, *Differential Geometry of Curves and Surfaces*. Englewood Cliffs, NJ: Prentice-Hall, 1976.

[27] G. E. Farin and D. Hansford, *The Essentials of CAGD*, 1st ed. USA: A. K. Peters, Ltd., 2000.

[28] D. Filip, R. Magedson, and R. Markot, "Surface approximations using bounds on derivatives," *Computer Aided Geometric Design*, vol. 3, pp. 295–311, 1986.

[29] J. M. Beck, R. T. Farouki, and J. K. Hinds, "Surface Analysis Methods," *IEEE Computer Graphics and Applications*, vol. 6, no. 12, pp. 18–36, 1986.

[30] A. Krishnamurthy, R. Khardekar, S. McMains, K. Haller, and G. Elber, "Performing efficient NURBS modeling operations on the gpu," *IEEE Trans. on Visualization and Computer Graphics*, vol. 15, pp. 530–543, 2009.

[31] N. Patrikalakis, "Surface-to-surface intersections," *IEEE Computer Graphics and Applications*, vol. 13, pp. 89–95, 1993.

[32] G. Farin, "An ssi bibliography," in *Geometry Processing for Design and Manufacturing*, R. E. Barnhill, Ed. SIAM, Philadelphia, PA: Chapter 10, 1992, pp. 205–207.

[33] N. Patrikalakis and T. Maekawa, "Intersection problems," in *Handbook of Computer Aided Geometric Design*, G. Farin, J. Hoschek, and M.-S. Kim, Eds. Amsterdam: Elsevier, 2002.

[34] H.-S. Heo, S. J. Hong, J.-K. Seong, M.-S. Kim, and G. Elber, "The intersection of two ringed surfaces and some related problems," *Graphical Models*, vol. 63, pp. 228–244, 2001.

[35] S. Hur, M. Oh, and T. Kim, "Approximation of surface-to-surface intersection curves within a prescribed error bound satisfying G^2 continuity," *Computer-Aided Design*, vol. 41, pp. 37–46, 2009.

[36] ——, "Classification and resolution of critical cases in Grandine and Klein's topology determination using a perturbation method," *Computer Aided Geometric Design*, vol. 26, pp. 243–258, 2009.

[37] T. Akenine-Möller, E. Hains, and N. Hoffman, *Real-Time Rendering*. Natick, MA: A.K. Peters, 2008.

[38] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast proximity queries using swept sphere volumes," Dept. of Computer Science, UNC, Tech. Rep., 1999.

[39] M. C. Lin and S. Gottschalk, "Collision detection between geometric models: A survey," in *Proc. of IMA Conference on Mathematics of Surfaces*, 1998, pp. 37–56.

[40] M. C. Lin and D. Manocha, *Handbook of Discrete and Computational Geometry*, 2nd ed., J. Goodman and J. O'Rourke, Eds. Chapman & Hall/CRC, 2004.

[41] T. Sederberg and R. Meyers, "Loop detection in surface patch intersections," *Computer Aided Geometric Design*, vol. 5, pp. 161–171, 1988.

[42] X. Ye and T. Maekawa, "Differential geometry of intersection curves of two surfaces," *Computer Aided Geometric Design*, vol. 16, pp. 767–788, 1999.

[43] Y.-J. Kim, J. Lee, M.-S. Kim, and G. Elber, "Efficient offset trimming for planar rational curves using biarc trees," *Computer Aided Geometric Design*, vol. 29, no. 7, pp. 555–564, 2012.

[44] J. Lee, Y.-J. Kim, M.-S. Kim, and G. Elber, "Efficient offset trimming for deformable planar curves using a dynamic hierarchy of bounding circular arcs," *Computer-Aided Design*, vol. 58, pp. 248–255, 2015.

[45] ——, "Efficient voronoi diagram construction for planar freeform spiral curves," *Computer Aided Geometric Design*, vol. 43, pp. 131–142, 2016.

[46] G. Müllenheim, "On determining start points for a surface/surface intersection algorithm," *Computer Aided Geometric Design*, vol. 8, no. 5, pp. 401–408, 1991.

[47] N. M. Aziz, R. Bata, and S. Bhat, "Bézier surface/surface intersection," *IEEE computer graphics and applications*, vol. 10, no. 1, pp. 50–58, 1990.

[48] R. E. Barnhill, "Geometry processing: Curvature analysis and surface-surface intersection," in *Mathematical Methods in Computer Aided Geometric Design*, T. Lyche and L. L. Schumaker, Eds. Academic Press, 1989, pp. 51–60.

[49] R. E. Barnhill, G. Farin, M. Jordan, and B. R. Piper, "Surface/surface intersection," *Computer Aided Geometric Design*, vol. 4, no. 1, pp. 3–16, 1987.

[50] R. E. Barnhill and S. N. Kersey, "A marching method for parametric surface/surface intersection," *Computer aided geometric design*, vol. 7, no. 1-4, pp. 257–280, 1990.

[51] L. Busé, M. Elkadi, and A. Galligo, "Intersection and self-intersection of surfaces by means of bezoutian matrices," *Computer Aided Geometric Design*, vol. 25, pp. 53–68, 2008.

[52] G. Elber, T. Grandine, and M.-S. Kim, "Surface self-intersection computation via algebraic decomposition," *Computer-Aided Design*, vol. 41, pp. 1060–1069, 2009.

[53] D. Pekerman, G. Elber, and M.-S. Kim, "Self-intersection detection and elimination in freeform curves and surfaces," *Computer-Aided Design*, vol. 40, pp. 150–159, 2008.

[54] P. Volino and N. M. Thalmann, "Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity," in *Computer Graphics Forum*, vol. 13, no. 3. Wiley Online Library, 1994, pp. 155–166.

[55] ——, "Collision and self-collision detection: Efficient and robust solutions for highly deformable surfaces," in *Computer Animation and Simulation '95*. Springer, 1995, pp. 55–65.

[56] Q. Hong, Y. Park, M.-S. Kim, and G. Elber, "Trimming offset surface self-intersections around near-singular regions," *Computers & Graphics*, vol. 82, pp. 84–94, 2019.

[57] S. Aomura and T. Uehara, "Self-intersection of an offset surface," *Computer-Aided Design*, vol. 22, pp. 417–421, 1990.

[58] R. Barnhill, T. Frost, and S. Kersey, "Self-intersections and offset surfaces," in *Geometry Processing for Design and Manufacturing*, R. Barnhill, Ed. SIAM, 1992.

[59] T. Maekawa, W. Cho, and N. M. Patrikalakis, "Computation of self-intersections of offsets of Bézier surface patches," *Journal of Mechanical Design: ASME Transactions*, vol. 119, pp. 275–283, 1997.

[60] J.-K. Seong, G. Elber, and M.-S. Kim, "Trimming local and global self-intersections in offset curves/surfaces using distance maps," *Computer-Aided Design*, vol. 38, pp. 183–193, 2006.

[61] Y. Wang, "Intersection of offsets of parametric surfaces," *Computer Aided Geometric Design*, vol. 13, pp. 453–465, 1996.

[62] C. Bajaj, C. Hoffmann, R. Lynch, and J. Hopcroft, "Tracing surface intersections," *Computer Aided Geometric Design*, vol. 5, pp. 285–307, 1988.

[63] C. Bajaj and G. Xu, "NURBS approximation of surface-surface intersection curves," *Advances in Computational Mathematics*, vol. 2, pp. 1–21, 1994.

[64] G. Elber, J.-J. Choi, and M.-S. Kim, "Ruled tracing," *The Visual Computer*, vol. 13, pp. 78–94, 1997.

[65] K.-J. Kim, "Torus and simple surface intersection based on a configuration space approach," Ph.D. dissertation, Dept. of Computer Science, POSTECH, 1998.

[66] D. Meek and D. Walton, "Approximating smooth planar curves by arc splines," *J. of Computational and Applied Mathematics*, vol. 59, pp. 221–231, 1995.

[67] ——, "Spiral arc spline approximation to a planar spiral," *J. of Computational and Applied Mathematics*, vol. 107, pp. 21–30, 1999.

[68] C. A. Neff, "Finding the distance between two circles in three-dimensional space," *IBM J. of Research and Development*, vol. 34, pp. 770–775, 1990.

[69] G. Elber, "Free form surface analysis using a hybrid of symbolic and numerical computation," Ph.D. dissertation, Dept. of Computer Science, The University of Utah, 1992.

[70] T. Maekawa, W. Cho, and N. M. Patrikalakis, "Computation of self-intersections of offsets of be´ zier surface patches," *Journal of Mechanical Design*, 1997.

[71] S. Aomura and T. Uehara, "Self-intersection of an offset surface," *Computer-Aided Design*, vol. 22, no. 7, pp. 417–421, 1990.

[72] Y. Wang, "Intersection of offsets of parametric surfaces," *Computer Aided Geometric Design*, vol. 13, no. 5, pp. 453–465, 1996.

[73] J.-J. Choi, "Local canonical cubic curve tracing along surface/surface intersections," Ph.D. dissertation, Dept. of Computer Science, POSTECH, 1997.

초 록

두 변수를 가지는 B-스플라인 자유곡면의 곡면간 교차곡선과 자가 교차곡선, 그리고 오프셋 곡면의 자가 교차곡선을 구하는 효율적이고 안정적인 알고리즘을 개발하는 새로운 접근 방법을 제시한다. 새로운 방법은 최하단 노드에 최대 접촉 토러스를 가지는 복합 바운딩 볼륨 구조에 기반을 두고 있다. 이 바운딩 볼륨 구조는 곡면간 교차나 자가 교차가 발생할 가능성이 있는 작은 곡면 조각 쌍들의 기하학적 검색을 가속화한다. 최대 접촉 토러스는 자기가 근사한 C^2-연속 자유곡면과 2차 접촉을 가지므로 주어진 곡면에서 다양한 기하 연산의 정밀도를 향상시키는데 필수적인 역할을 한다.

효율적인 곡면간 교차곡선 계산을 지원하기 위해, 미리 만들어진, 최하단 노드에 최대 접촉 토러스가 있으며 구형구면 트리를 가지는 복합 이항 바운딩 볼륨 구조를 설계하였다. 최대 접촉 토러스는 거의 모든 곳에서 접선교차가 발생하는, 자명하지 않은 곡면간 교차곡선 계산 문제에서도 효율적이고 안정적인 결과를 제공한다.

곡면의 자가 교차 곡선을 구하는 문제는 주로 마이터 점 때문에 곡면간 교차곡선을 계산하는 것 보다 훨씬 더 어렵다. 자가 교차 곡면은 마이터 점 부근에서 법선 방향이 급격히 변하며, 마이터 점은 자가 교차 곡선의 끝점에 위치한다. 따라서 마이터 점은 자가 교차 곡면의 기하 연산 안정성에

큰 문제를 일으킨다. 마이터 점을 안정적으로 감지하여 자가 교차 곡선의 계산을 용이하게 하기 위해, 자유곡면을 위한 복합 바운딩 볼륨 구조에 적용할 수 있는 삼항 트리 구조를 제시한다. 특히, 두 변수를 가지는 곡면의 매개변수영역에서 마이터 점을 충분히 작은 사각형으로 감싸는 특별한 표현 방법을 제시한다.

접선교차와 마이터 점을 가지는, 아주 자명하지 않은 자유곡면 예제를 사용하여 새 방법이 효과적임을 입증한다. 모든 실험 예제에서, 기하요소들의 정확도는 하우스도르프 거리의 상한보다 낮음을 측정하였다.

주요어: 곡면 교차, 곡면 자가 교차, 바운딩 볼륨 구조, 구형구면, 최대 접촉 토러스, 마이터 점
학번: 2016-21202