Self-Intersection Computation for Freeform Surfaces Based on a Regional Representation Scheme for Miter Points

Youngjin Park[a], Q Youn Hong[b], Myung-Soo Kim[a], Gershon Elber[b]

[a] *Dept. of Computer Science and Engineering, Seoul National University, Seoul 08826, South Korea*
[b] *Computer Science Department, Technion, Haifa 32000, Israel*

Abstract

We present an efficient and robust algorithm for computing the self-intersection of a freeform surface, based on a special representation of *miter points*, using sufficiently small quadrangles in the parameter domain. A self-intersecting surface changes its normal direction quite 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 the surface. To facilitate a stable detection of miter points, we employ osculating toroidal patches and their intersections, and consider a gradual change to degenerate intersections as a signal for the detection of miter points. The exact location of each miter point is bounded by a tiny ball in the Euclidean space and is also represented as a small quadrangle in the parameter space. The surface self-intersection curve is then constructed, using a hybrid Bounding Volume Hierarchy (BVH), where the leaf nodes contain osculating toroidal patches and miter quadrangles. We demonstrate the effectiveness of our approach by using test examples of computing the self-intersection of freeform surfaces.

Keywords: Surface self-intersection, miter point, Bounding Volume Hierarchy (BVH), osculating toroidal patches, miter quadrangles

1. Introduction

In geometric and solid modeling, Boolean operations support the construction of complex solid models from simple primitives. To convert the solid models to their boundary representations, we need to use an intersection algorithm for freeform surfaces. In the majority of previous algorithms for intersecting two surfaces, 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, in particular, when the surfaces are given as offset or sweep surfaces. Compared with the previous work on the surface-surface-intersection (SSI) problem [6; 12; 19; 20; 21], the self-intersection problem has received considerably less attention [3; 5; 8; 9; 10]. Moreover, the self-intersection problem for offset or sweep surfaces usually solves a simplified subproblem [11; 25], where the most complicated self-intersections are trimmed away. In this paper, we consider the self-intersection problem without assuming any prior knowledge on the applications of this unary geometric operation on freeform surfaces. The main focus of this work is on the local changes needed to the spatial data structures (such as Bounding Volume Hierarchy) for freeform surfaces.

The self-intersection of a freeform surface $S(u,v)$ is formally defined as the following set:

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

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

*Corresponding author
Email address: `mskim@snu.ac.kr` (Myung-Soo Kim)

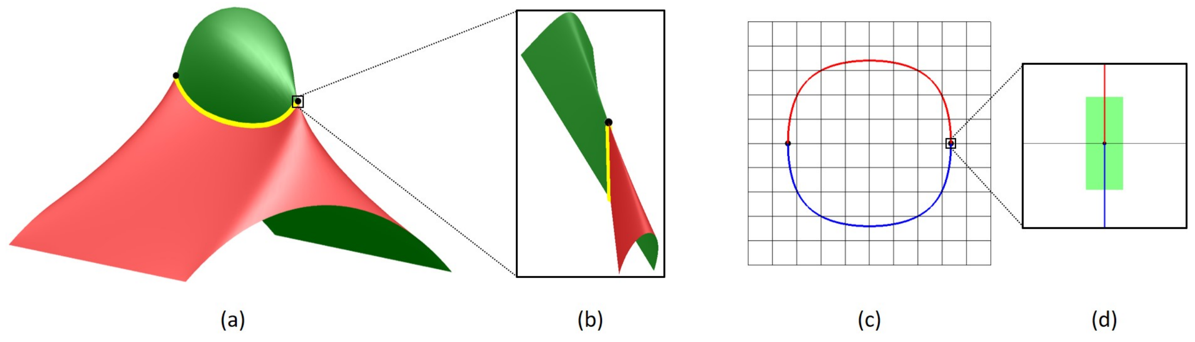


Figure 1: Miter points on a surface self-intersection curve: (a)–(b) in the Euclidean xyz-space, (c) in the (u, v)-parameter domain, and (d) bounding a miter point using a quadrangle.

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 [8; 9; 10].

Figure 1(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(c), 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 [4]. In this paper, we address the representational issue for the miter points, as a crucial step towards a stable solution for the surface self-intersection problem.

The miter points play an important role in the determination of the correct topology for a self-intersection curve, in particular, when we deal with the trimming problem for self-intersecting offset surfaces [9; 10; 11]. Once all the proper neighborhoods of miter points are successfully detected and separated from other (u, v)-subdomains, the self-intersection problem is essentially reduced 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. Ho and Cohen [9; 10] 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 $S(u, v)$. Galligo and Pavone [8] constructed the self-intersection curve using a triangular mesh approximation to the freeform surface. As shown in an example of Galligo and Pavone [8], the curve approximation near a miter point is highly unstable, producing a curve approaching to a miter point in a zigzag fashion. Near a miter point, the approximating triangles become almost coplanar and their intersections will inevitably produce unstable results.

In a recent work on the offset self-intersection trimming problem, Hong et al. [11] approximated the offset self-intersection curve near a miter point (on an offset surface) by intersecting an osculating toroidal patch (which is almost identical to other osculating toroidal patches nearby) with a normal plane in the direction of the curve tracing. They have observed that the location of a miter point in the xyz-space is stable; however, the corresponding location in the (u, v) or (s, t)-parameter domain is numerically unstable to detect in a high precision. In the current work, we take a practical approach to the detection of miter point locations, by representing the location in the (u, v)-parameter domain as a bounding quadrangle $Q(\alpha, \beta)$, $0 \leq \alpha, \beta \leq 1$, that contains the exact miter point somewhere inside the quadrangle Q. The reparameterized surface subpatch $\hat{S}(\alpha, \beta) = S(Q(\alpha, \beta))$ is guaranteed to contain the exact location of the miter point in the xyz-space. We bound the subpatch $\hat{S}$ in a tiny ball of radius $\epsilon > 0$. As shown in Figure 1(c), the size of Q is considerably larger than the ϵ-ball, in particular, along the tangent direction of the red and blue solution curves at their junction point in the parameter domain.

For the acceleration of computing speed, recent algorithms [13; 18] for surface intersection often employ the spatial data structure of Bounding Volume Hierarchy (BVH), specially designed for freeform surfaces, where each leaf node represents a small surface patch that can be approximated by simple primitives such as triangles, rectangles, toroidal patches, etc. It has been implicitly assumed that the surface patches in the leaf nodes have no local self-intersections. Nevertheless, the existence of miter points means that this assumption is no longer valid and we need to fix the structure of leaf nodes to a more general form so that the local surface geometry can be represented more precisely. There are certain technical issues that we need to address for different cases of miter point locations, the details of which we discuss in Section 3. Another technical issue is in the geometric computations with the leaf nodes containing miter points. This is a more difficult problem than the representational issue for leaf nodes – we need to deal with an unusual geometric behavior of the surface normal directions near miter points. (Note that the surface normal direction turns around more than 180° in the neighborhoods of miter point, which makes the normal cone tests fail and subdivision-based SSI algorithms never ending [23; 24].) Using the ϵ-ball that bounds the surface patch itself in the Euclidean xyz-space, we can control the influence of miter points within certain error bounds of the geometric operations needed for an application. We discuss the details of these special treatments of the miter points in Section 4, regarding the surface self-intersection problem. Other geometric operations (mainly dependent on the surface normals) would require considerably more sophisticated handling of the special geometric features of miter points even though we may also expect reasonably good solutions to these problems thanks to the geometric nature of ϵ-bounding for the miter points.

The algorithmic flow for computing a surface self-intersection is related to, but somewhat different from, that of intersecting two different surfaces. As shown in Figure 2(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 computationally 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 subdividing the surface into smaller and smaller pieces along the common boundary. 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 2(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 2(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, $(S_2 \setminus S_m)$, we can implicitly represent these subtrees by restricting the parameter domains of the subtrees for S_1, S_m, S_2. For example, in the construction of three child nodes of S_m, their bounding volumes are generated for three surfaces $(S_m)_1, (S_m)_m, (S_m)_2$. 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

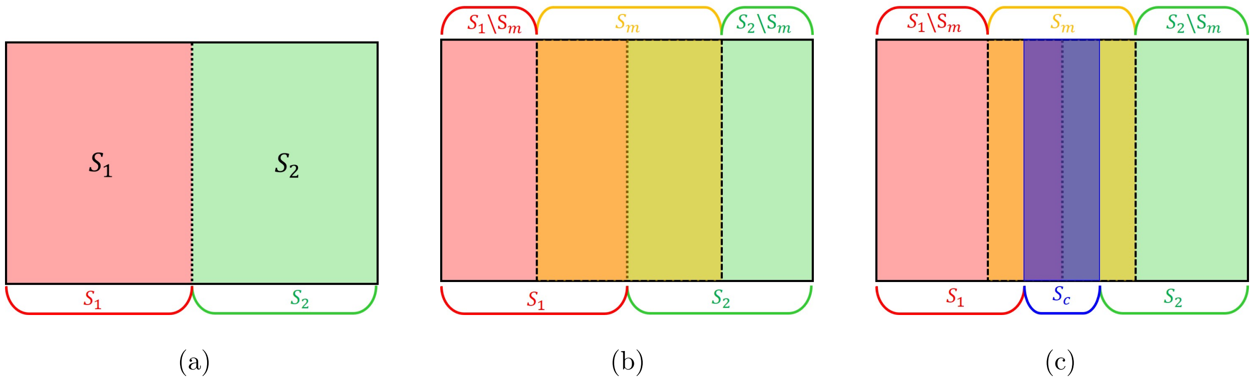


Figure 2: Recursive self-intersections: (a) in S_1 and S_2 sharing a common boundary curve, (b) in S_1, S_m, S_2 with S_m covering the common boundary, and (c) in S_1, S_m, S_2, followed by global intersections.

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, $(S_2 \setminus S_m)$ 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 relative size of the overlap area, 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 experience, the computational advantage from the overlap region is far more beneficial than the duplication problem.

The main contributions of this work 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. Note that the SSI curves are often approximated within an error bound of similar size (as a sequence of cubic curve segments).

- Using osculating toroidal patches [11; 16; 18], 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 (developed for freeform surfaces with no self-intersections) 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.

- We present an efficient algorithm for computing the self-intersection of a freeform surface, based on recursive self-intersections and global intersections on smaller subpatches of the input surface S. Due to the overlap of S_m with S_1 and S_2, there are duplications of computation. We suggest a simple way of filtering out the redundancies in the most expensive stage of constructing the self-intersection curve.

- Using only one BVH structure precomputed for the input surface S, we introduce a new technique for supporting the effects of different BVH trees for smaller surfaces $S_1 \setminus S_m$, S_c, $S_2 \setminus S_m$ by comparing the parameter domains of the surfaces and the BVH nodes.

- In Section 4, 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 often has a linear shape near miter points.
- We provide a general guideline for the extension of geometric algorithms so that they can properly handle input surfaces that may self-intersect. Some duplications of computation may be inevitable in a geometric approach based on recursive subdivisions. We need to develop different strategies for filtering out the duplication of computations, depending on the requirements for each application.

2. Related work

There is a rich body of literatures (including some extensive survey articles) on the surface-surface-intersection (SSI) problem [6; 12; 19; 20; 21]; 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 [3; 5]. In the univariate case of the curve self-intersection problem: $C(u) = C(s)$, Pekerman et al. [22] 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. [5] 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. [3] 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 [26; 27] 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 [23; 24]. Starting with an initial solution from a polygonal approximation to the given freeform surface, Ho and Cohen [9; 10] 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 [8] also used a triangular mesh approximation and reported certain limitations of the mesh-based approach in the neighborhoods of miter points. Hong et al. [11] 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 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 [1; 2; 17; 25; 28]. The offset self-intersection curves are often traced using numerical techniques (such as Runge-Kutta methods) applied to differential equations [1; 17; 28]. The offset trimming technique of Seong et al. [25] 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.

For the intersection of two polygonal meshes, the BVH-based algorithms are now widely accepted as the method of choice for efficiency reasons, where the detection of all possible pairs of intersecting triangles is important [14; 15]. The BVH-based approach is also commonly used for intersecting two freeform surfaces. The GPU-based SSI algorithm of Krishnamurthy et al. [13] used the AABB (Axis-Aligned Bounding Box) tree, which is the most suitable for GPU implementations. Based on the surface approximation techniques of Filip et al. [7] and Liu et al. [16], Park et al. [18] developed a hybrid BVH using rectangle swept spheres (RSS) for internal nodes and osculating toroidal patches for leaf nodes. In the current work, we follow the basic guideline of Park et al. [18] and make some changes to the BVH structure.

3. BVH Construction

The consideration of miter points and the ternary structure for the BVH tree necessitates certain changes to the conventional BVH structures for freeform surfaces. The changes are mainly in the leaf nodes containing miter points. For the sake of simplicity, we assume that the uv-parameter domain of a surface patch is explicitly stored in the corresponding node, and a leaf node may contain at most one miter point.

3.1. 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 3.3.) Each subpatch S_{ij} is then tightly approximated by an osculating toroidal patch T_{ij} based on the construction steps discussed in Section 3.3 of Park et al. [18]. 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 1, 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 1(b), the domains for $S_{i,j+1}$ and $S_{i,j-1}$ have the corresponding branches, which are computed by intersecting the two 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 $S_u \times S_v = \mathbf{0}$. 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 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 $Q(\alpha,\beta)$, $(0 \le \alpha,\beta \le 1)$, 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 resolutions; 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.

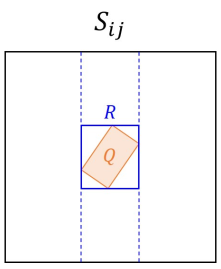


Figure 3: Converting S_{ij} to an internal node.

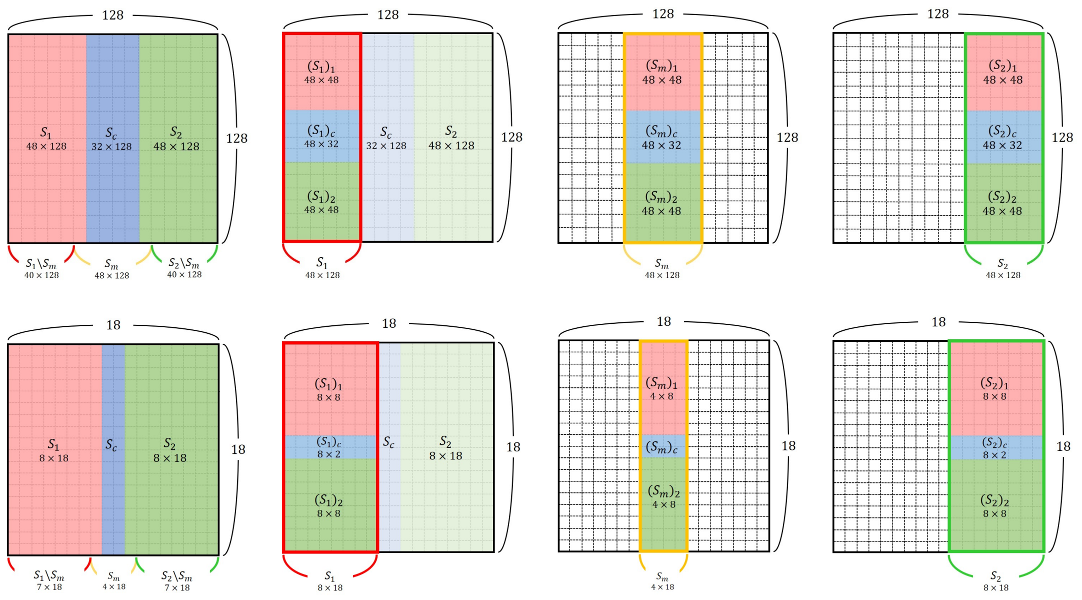


Figure 4: Grid sizes for the subpatches generated by the vertical and horizontal splits.

3.2. 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 3, 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 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.

3.3. 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 2(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 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 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 4.1, 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 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 8. 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 $8M \times 8N$ 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 below.

4. Surface Self-Intersection Algorithm

Our surface self-intersection algorithm is based on traversing the ternary BVH tree constructed in the previous section. 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.

4.1. 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 $\epsilon_T > 0$. (We use the surface approximation techniques of Liu et al. [16] and Park et al. [18].) 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}. These subpatches are handled 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 map of T_{ij} in the window covers an area larger 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.

4.2. 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, 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), $(S_1 \setminus S_m, S_c)$, $(S_2 \setminus S_m, S_c)$, 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, and 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 use Rectangle Swept Spheres (RSS) for the overlap test [18]. One exception is an ϵ-ball, for bounding the composite surface $S(Q(\alpha, \beta))$ for a miter-quadrangle Q.

4.3. 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 [16]. 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 Euclidean xyz-space) near the miter point is usually an almost linear segment with the miter point at its open end. By intersecting the short curve (often approximated with a line segment) against the subpatch S_{kl}, we can decide the planar arrangement of solution 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 1) are constructed in a preprocessing step and stored in the

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, and right child nodes for $S_A$;
    Intersect($S_1, S_B$); Intersect($S_m, S_B$); Intersect($S_2, S_B$);
end
```

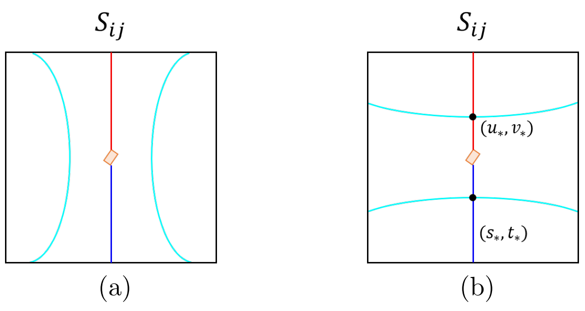


Figure 5: Arrangement of solution curves: (a) when the local self-intersection curve has no intersection with S_{kl}, and (b) when the local self-intersection curve has a global intersection with S_{kl}.

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 5(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 5(b), the solution curve for a global self-intersection with S_{kl} will intersect the (u, v) and (s, t) solution curves for the local self-intersection.

The illustration in Figure 5 corresponds to the example of Figure 8(a), with a hyperbolic type solution curve. In Figure 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.

5. Experimental Results

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.

Table 1: 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 (418)	599
(B)	1,902	260	1,133 (321)	773
(C)	1,872	257	1,251 (316)	662
(D)	11,113	1,111	6,960 (1,482)	3,629
(E)	10,980	1,238	5,133 (1,393)	4,858
(Miter-A)	3,616	451	45,610 (11,712)	1,730
(Miter-B)	1,986	257	11,400 (2,113)	774
(Miter-C)	1,828	296	28,522 (4,793)	1,374

Figure 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 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 1 reports some statistics on the self-intersection curve construction for the eight example surfaces in Figures 6–7. 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 (shown together with the parallel computing time using all 8 cores of the CPU). 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, 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 7. Each patch is intersected with three parallel planes as shown in the left, middle, and right columns of Figure 8. The top surface of Figure 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 7). Regarding the three topological types of the plane-surface intersection curve, as discussed in Section 4.3, 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 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 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 of which can be based on the techniques we have introduced in this paper.

6. Conclusions

In this paper, 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. The geometric behavior of freeform surfaces at miter points may look highly unwieldy. Nevertheless, 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 quadrangles in the parameter space. Based on this observation, we have made reliable decisions on the local geometry of the self-intersecting surfaces at miter points. In future work, we would like to extend the coverage of geometric operations on self-intersecting freeform surfaces, including the self-bisector computation for self-intersecting surfaces, among many others.

Acknowledgments

We would like to thank anonymous reviewers for their invaluable comments. This research was supported in part by the European Union Horizon 2020 research and innovation programme, under grant agreement No 862025, and in part by the National Research Foundation of Korea (No. NRF-2019R1A2C1003490, NRF-2019K1A3A1A78112596).

References

[1] S. Aomura and T. Uehara. Self-intersection of an offset surface. *Computer-Aided Design*, **22**(7):417–421, 1990.
[2] R. Barnhill, T. Frost, S. Kersey. Self-intersections and offset surfaces. In *Geometry Processing for Design and Manufacturing*, Robert Barnhill (Ed.), SIAM, 1992.
[3] L. Busé, M. Elkadi, A. Galligo. Intersection and self-intersection of surfaces by means of Bezoutian matrices. *Computer Aided Geometric Design*, **25**(2):53–68, 2008.
[4] M. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, NJ, 1976.
[5] G. Elber, T.. Grandine, M.-S. Kim, Surface self-intersection computation via algebraic decomposition. *Computer-Aided Design*, **41**(12):1060–1069, 2009.
[6] G. Farin. An SSI bibliography. *Geometry Processing for Design and Manufacturing*, R.E. Barnhill (Ed.), Chapter 10, pp. 205–207, SIAM, Philadelphia, PA, 1992.
[7] D. Filip, R. Magedson, R. Markot. Surface approximations using bounds on derivatives. *Computer Aided Geometric Design*, **3**(4):295–311, 1986.
[8] A. Galligo and J.P. Pavone. Self intersections of a Bézier bicubic surface. In *Proc. of the 2005 Int'l Symp. on Symbolic and Algebraic Computation (ISSAC)*, pp. 148–155, 2005.
[9] C.-C. Ho. *Feature-Based Process Planning and Automatic Numerical Control Part Programming*. Ph.D. Thesis, Dept. of Computer Science, Univ. of Utah, 1997.
[10] C.-C. Ho and E. Cohen. Surface self-intersection. In *Mathematical Methods for Curves and Surfaces*, Lyche, T., and Schumaker, L.L. (Eds.), Proc. of Oslo 2000, Vanderbilt Univ. Press, Nashville, Tennessee, 183–194, 2000.
[11] Q Hong, Y. Park, M.-S. Kim, G. Elber. Trimming offset surface self-intersections around near-singular regions. *Computers & Graphics*, **82**:84–94, 2019.
[12] J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design*. AK Peters, Wellesley, MA, 1993.
[13] A. Krishnamurthy, R. Khardekar, S. McMains, K. Haller, G. Elber. Performing efficient NURBS modeling operations on the GPU. *IEEE Trans. on Visualization and Computer Graphics*, **15**(4):530–543, 2009.
[14] M.C. Lin and S. Gottschalk. Collision detection between geometric models: A survey. *Proc. of IMA Conference on Mathematics of Surfaces*, pp. 37–56, 1998.
[15] M.C. Lin and D. Manocha. Collision and proximity queries. *Handbook of Discrete and Computational Geometry*, 2nd Ed., J.E. Goodman and J. O'Rourke, Eds., Chapman & Hall/CRC, pp. 787–807, 2004.
[16] X.-M. Liu, L. Yang, J.-H. Yong, H.-J. Gu, J.-G. Sun. A torus patch approximation approach for point projection on surfaces. *Computer Aided Geometric Design*, **26**(5):593–598, 2009
[17] T. Maekawa, W. Cho, N.M. Patrikalakis. Computation of self-intersections of offsets of Bézier surface patches. *Journal of Mechanical Design: ASME Transactions*, **119**(2):275–283, 1997.
[18] Y. Park, S.-H. Son, M.-S. Kim, G. Elber. Surface-surface-intersection computation using a bounding volume hierarchy with osculating toroidal patches in the leaf nodes. *Computer-Aided Design*, **127**, Article 102866, 2020.
[19] N. Patrikalakis. Surface-to-surface intersections. *IEEE Computer Graphics and Applications*, **13**(1):89–95,1993.
[20] N. Patrikalakis and T. Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer, 2002.
[21] N. Patrikalakis and T. Maekawa. Intersection problems. *Handbook of Computer Aided Geometric Design*, G. Farin, J. Hoschek, and M.-S. Kim (Eds), Elsevier, Amsterdam, 2002.

[22] D. Pekerman, G. Elber, M.-S. Kim. Self-intersection detection and elimination in freeform curves and surfaces. *Computer-Aided Design*, **40**(2):150–159, 2008.

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

[24] T. Sederberg, H. Christiansen, S. Katz. An improved test for closed loops in surface intersections. *Computer-Aided Design*, **21**(8):505–508, 1989.

[25] J.-K. Seong, G. Elber, M.-S. Kim. Trimming local and global self-Intersections in offset curves/surfaces using distance maps. *Computer-Aided Design*, **38**(3):183–193, 2006.

[26] P. Volino and N.M. Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum*, Eurographics '94, M. Daehlen and L. Kjelldahl (Eds), **13**(3):C-155–166, 1994.

[27] P. Volino and N.M. Thalmann. Collision and self-collision detection: Efficient and robust solutions for highly deformable surfaces. In *Computer Animation and Simulation '95*, D. Terzopoulos and D. Thalmann (Eds), pp. 55—65. Springer-Verlag, 1995.

[28] Y. Wang. Intersection of offsets of parametric surfaces. *Computer Aided Geometric Design*, **13**(5):453–465, 1996.

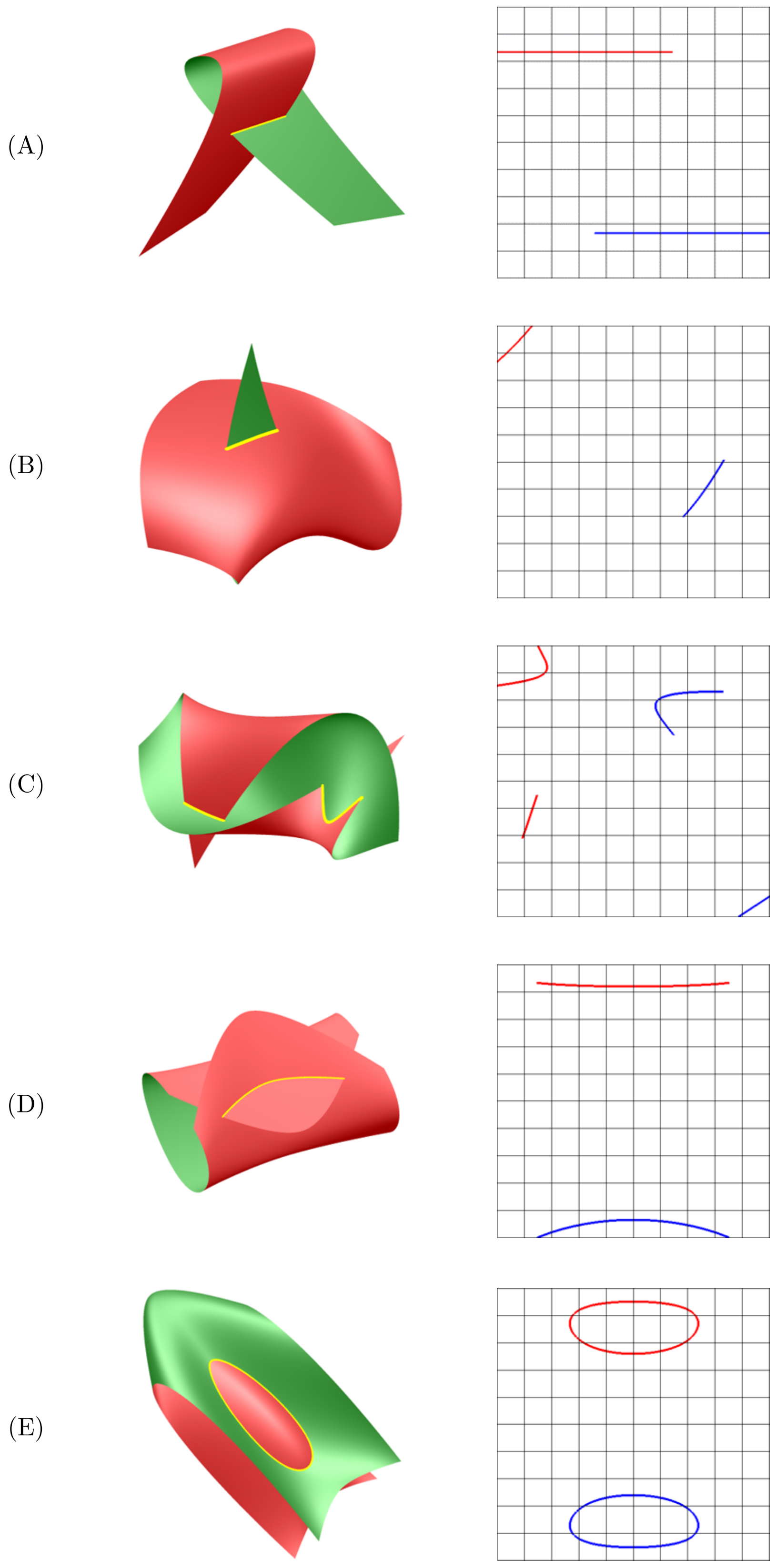


Figure 6: Examples of self-intersecting surfaces; the left column shows the results of self-intersecting a freeform surface and the right column shows the intersection curves in the parameter domains of the surfaces, respectively.

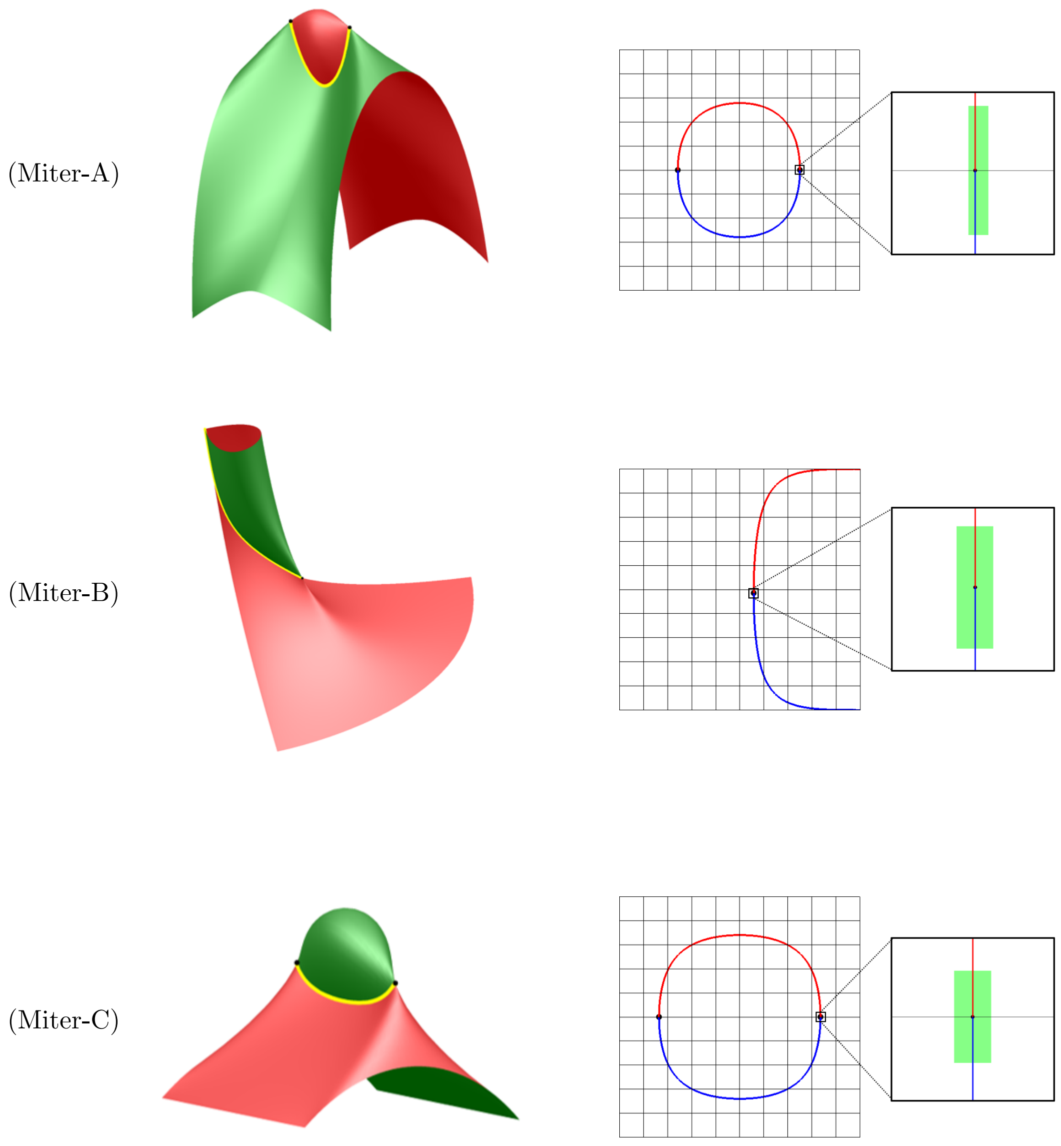
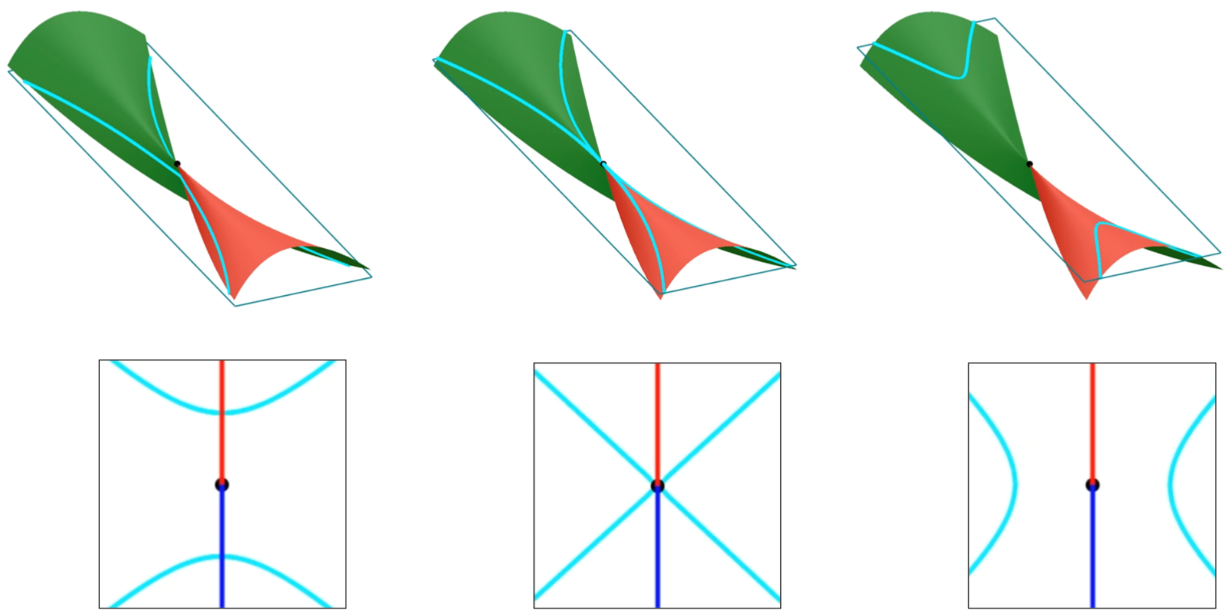
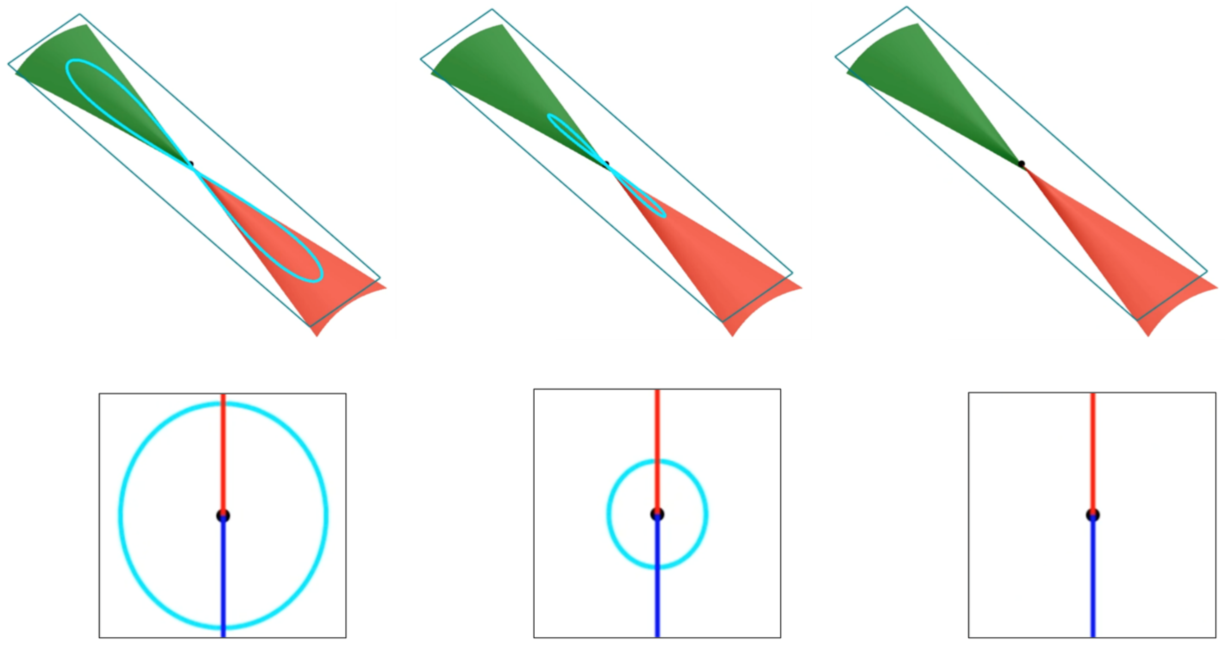


Figure 7: Examples of self-intersecting surfaces with miter point(s) on their intersection curves; the left column shows the results of self-intersecting a freeform surface and the right column shows the intersection curves in the parameter domains of the surfaces and zoom-in areas around the miter points, respectively.



(a)



(b)

Figure 8: Examples of local self-intersection curve near the miter point.