A Line/Trimmed NURBS Surface Intersection Algorithm Using Matrix Representations

Jingjing Shen, Laurent Busé, Pierre Alliez, Neil Dodgson

A Line/Trimmed NURBS Surface Intersection Algorithm Using Matrix Representations

Jingjing Shen[a], Laurent Busé[b], Pierre Alliez[b], Neil Dodgson[a]

[a]*The Computer Laboratory, University of Cambridge, UK*
[b]*Inria Sophia Antipolis - Méditerranée, France*

Abstract

We contribute a reliable line/surface intersection method for trimmed NURBS surfaces, based on a novel matrix-based implicit representation and numerical methods in linear algebra such as singular value decomposition and the computation of generalized eigenvalues and eigenvectors. A careful treatment of degenerate cases makes our approach robust to intersection points with multiple pre-images. We then apply our intersection algorithm to mesh NURBS surfaces through Delaunay refinement. We demonstrate the added value of our approach in terms of accuracy and treatment of degenerate cases, by providing comparisons with other intersection approaches as well as a variety of meshing experiments.

1. Introduction

NURBS (Non Uniform Rational Basis Spline) is a mathematical model that has become the standard in the Computer-Aided Design (CAD) community for generating and representing curves and surfaces with high quality and precision [27]. Among the many possible geometric operations applicable to NURBS models, computing the intersection between a line and a NURBS surface is of particular importance. Indeed, such an intersection algorithm is central for high quality visualization through ray tracing. It is also central for isotropic triangle meshing of parametric surfaces through Delaunay refinement [11, 35, 23], which can also be used for repairing and meshing imperfect CAD models [8]. From the theoretical point of view, intersection problems can be translated into polynomial systems solving, which is itself an active area of research where sophisticated methods based on algebraic computations have been proposed. From the practical point of view, a good intersection algorithm must return accurate approximated solutions, but it must also be reliable and robust in the sense that all intersections must be correctly identified, even in degenerated or nearly degenerated geometric configurations. Common degenerated intersections include cases where the intersection point has multiple pre-images in parameter space, where the ray is near tangential to the surface, or even more extreme, where the ray intersects the surface in infinitely many points as the ray is partially included in the surface. These degenerated cases are not rare in practice (see for instance Table 3 in the context of Delaunay-based meshing of NURBS surfaces) and hence intersection algorithms must deal with them reliably.

The focus of this paper is a novel intersection algorithm for computing the intersection between a line and a (trimmed) NURBS surface. Based on matrix-based implicit representation of rational Bézier surfaces, our approach is both reliable and robust to the above-mentioned degenerated cases. In addition, we turn this intersection algorithm into an intersection oracle for the Delaunay-based meshing framework of the CGAL library. We obtain a reliable Delaunay-based NURBS meshing approach, oblivious to the patch layouts and parameterizations of the input NURBS surfaces. Delaunay-based meshing is useful for several classes of applications in computational engineering such as structural and FEM simulation where it is critical to control the shape and size of the mesh elements in order to guarantee the accuracy and convergence of the simulation.

Related work. Most related work on the line/NURBS intersection problem has mainly focused on ray-tracing [1, 25] and, as far as we know, all practical algorithms rely on two different approaches: The *Newton-Raphson iteration* [21, 24] and recursive *subdivision* approach [13, 14, 31, 16]. The former relies upon point and derivative evaluations, while the latter is based on subdivision of NURBS patches and the convex hull property. Although all these operations can

be performed on NURBS surfaces, they are substantially slower than their equivalent for rational Bézier surfaces. Therefore, in order to improve performance, NURBS surfaces are split into a collection of joint Bézier patches during a preprocessing step, without loss of precision. Intersection problems are then dealt with at the level of Bézier patches.

Line/Bézier patch intersection algorithms based on the *Newton-Raphson method* first compute the equations of two planes defining the line and then substitute the parameterization of the Bézier patch into these equations to solve a system of 2 polynomial equations in the 2-dimensional parameter space of the Bézier patch. Such algorithms are usually very fast [1] when a good initial guess is provided, and when its convergence can be controlled within the parameter space of the patch. However, these two requirements are difficult to meet in practice. In general, a flattening process is applied to split each Bézier patch into many near-planar sub-patches and their centers are then used as initial guesses.

Subdivision methods for solving line/Bézier patch intersection consist in turning the intersection problem into the one of finding the zeroes of a function in the 2-dimensional parameter space of the Bézier patch. The latter problem is then solved by using recursive subdivision techniques and exploiting the convex hull property of Bézier patches. Such approaches have been widely studied with several variations. Two major contributions are the "recursive subdivision" approach by Dokken et al. [13, 14], whose implementation is available in the SISL Library [34], and the more recent "Bézier clipping" approach introduced by Sederberg et al. [31]. Contrary to Newton-Raphson based methods, subdivision methods do not require any initial guess but they are not as fast. Moreover, subdivision methods can converge to wrong intersections and may fail in the presence of multiple intersections. Although several heuristics have been devised to address these failure cases (e.g. [10, 16]), these methods are challenged by multiple intersections (e.g. [5]). Another issue with subdivision methods is the insufficient control over the geometric precision in 3D. This issue stems from the fact that the intersection problem is solved in the 2D parameter space of the Bézier patch. As the parameterization may have strong distortion it may translate into an insufficient precision in 3D. Some heuristics have been proposed to circumvent this issue [16, §5.2-3] but they are not a final answer in addition to being often compute-intensive.

Recently, Busé [9] introduced an approach for solving line/Bézier patch intersection problems, through a novel matrix-based implicit representation of Bézier patches (referred to as "MRep"). Note that the Bézier patch is defined over a unit parameter domain, while its MRep provides an implicit representation of the algebraic surface that contains the patch, referred to as the *supporting surface*, its parameters being not limited to the unit parameter square, see Figure 1. Departing from previous work, this approach relies upon an MRep of the Bézier patch and solves the intersection problem in the 1-dimensional parameter space of the line (Ba et al. [4]). More specifically, after substituting the line parameterization in the MRep of the patch, all intersection points between the line and the supporting surface are computed by means of eigencomputations, without relying on a flattening step or on a subdivision scheme. Then, the localization of intersection points in the parameter space of the Bézier patch is performed by solving a point inversion problem from the MRep. However, this last step is limited to intersection points that have a single pre-image (counting multiplicity) on the supporting surface of the Bézier patch. Therefore, the initial contribution [9] does not cover all possible geometric configurations of intersections, in particular the aforementioned degenerated cases.

Positioning and contribution. In this paper we contribute a reliable ray/trimmed NURBS surface intersection algorithm by improving the MRep-based approach. More specifically, we introduce several modifications and improvements over a recent work [9] in order to deal with all degenerated cases. Our choice to investigate further this approach is motivated by the following advantages of MRep compared to the two other methods. First, there is a proliferation of small intervals that are refined and split during several iterative steps in subdivision methods, whereas our approach is a global method that computes all intersections in a single step. Regarding the Newton iteration method it is necessary to provide an initial guess, which can sometimes leads to difficulties. Second, several heuristics and parameters are required to address robustness issues with subdivision methods. Our MRep-based approach is made robust by leveraging standard and well-established tools from numerical linear algebra that have been specifically designed for that purpose, namely the singular value decomposition and the computation of generalized eigenvalues and eigenvectors, that are also reliably implemented in modern software libraries and well suited to deal with limited precision. Another advantage of using MRep is that the numerical precision can be controlled directly in 3D space, whereas for subdivision methods control is only possible in the 2D parameter space of the Bézier patch and hence suffers from the distortion of its parameterization. Furthermore, in the Newton iteration and the subdivision methods an intersection point with multiple pre-images is found as many times as its number of pre-images, and all these pre-images must be

identified. In comparison, such an intersection point is found only once by using MRep and then all its pre-images are detected at once by our new inversion algorithm (Section 2.4). We notice that points which are located on seams of closed patches have (at least) two pre-images.

Another contribution of this paper is a new method for isotropic triangle meshing of NURBS surfaces. Based on Delaunay filtering and refinement, our approach departs from parametric approaches by meshing in embedding space instead of parametric space. The meshing engine requires a reliable line/NURBS intersection method (referred to as an oracle) derived from our new intersection algorithm. In addition to ensure both approximation accuracy and mesh quality, our approach is seamless as it does not depend on the initial decomposition into NURBS patches, and is oblivious to the parameterization of the patches. Removing such dependencies provides us with a means to reliably mesh across patches with greater control over mesh sizing and shape of the elements.

Content. The paper is organized as follows. In Section 2 we first briefly review matrix-based representation of Bézier patches and then provide description of our line/Bézier patch intersection algorithm. Our main contributions are (1) a reliable intersection algorithm that is able to recover the intersections in near degenerate cases (where the line is tangential to or contained into the surface), where the numerical computation of regular pencil extraction, proposed by Ba et al. [4], fails; (2) a reliable inversion algorithm that is able to extract all pre-images of a surface point with multiple pre-images, while Busé's algorithm [9] is limited to the inversion of the surface point with a unique pre-image. As our inversion algorithm covers all configurations it may be used for solving other geometric problems involving parameterized objects. We evaluate and compare our method with others in Section 3.

Section 4 applies our intersection algorithm to seamlessly mesh NURBS surfaces using the Delaunay-based approach from the CGAL library. Finally, some illustrative examples, details of implementations and comparisons are provided in Section 4.3.

2. Line/Bézier Patch Intersection

In this section, we describe in detail a new algorithm for intersecting a ray and a Bézier patch. As already mentioned, we recall that ray/NURBS intersections are always split into several such ray/Bézier patch intersections for computational performance. We begin with a quick review of the concept of MRep (Section 2.1) and then proceed with the overall description of the algorithm (Section 2.2). We then provide details on two important steps of the algorithm, namely the pencil reduction (Section 2.3) and the inversion (Section 2.4) steps. This latter step that allows the inversion of parameterizations by means of MRep is a crucial point that was not treated in [8] for degenerate geometric situations (intersection points with multiple pre-images, or line nearly tangent or contained to a Bézier patch). Thanks to this improved inversion algorithm, we mention that the MRep approach which is described below can now be transposed for solving other geometric problems involving parameterized objects. For instance, it can be used to compute the euclidian distance of a given point to a NURBS surface by inversion of the parameterization of the two-dimensional family of the normal lines to the NURBS surface (often called normal congruence).

2.1. Matrix Representation (MRep)

Suppose given a Bézier patch of (bi-)degree (d_1, d_2):

$$\begin{array}{rcl} \phi : [0,1]^2 \subset \mathbb{R}^2 & \to & \mathbb{R}^3 \\ (u,v) & \mapsto & \dfrac{\sum_{i=0}^{d_1}\sum_{j=0}^{d_2} w_{i,j}\mathbf{b}_{i,j}B_i^{d_1}(u)B_j^{d_2}(v)}{\sum_{i=0}^{d_1}\sum_{j=0}^{d_2} w_{i,j}B_i^{d_1}(u)B_j^{d_2}(v)} = \left(\dfrac{f_1(u,v)}{f_0(u,v)}, \dfrac{f_2(u,v)}{f_0(u,v)}, \dfrac{f_3(u,v)}{f_0(u,v)}\right) \end{array} \tag{1}$$

where $b_{i,j}$, $w_{i,j}$ and $B_i^{d_j}$ denote the control points, the weights and the Bernstein polynomials respectively, and f_0, f_1, f_2, f_3 are polynomials in the parameter (u, v) of degree $\leqslant (d_1, d_2)$, inequalities between bi-degrees being understood component-wise. To build an MRep of ϕ, first pick a degree $\nu = (\nu_1, \nu_2)$ and consider the set of 4-tuples of polynomials $(g_0(u,v), g_1(u,v), g_2(u,v), g_3(u,v))$ such that

$$\text{degree}(g_k(u,v)) \leqslant \nu \text{ and } \sum_{k=0}^{3} g_k(u,v) f_k(u,v) \equiv 0.$$

It is a finite dimensional vector space and the computation of one of its bases, say $L_1, \ldots, L_{r_\nu}$, is achieved by solving a linear system. From here, an MRep of degree ν for ϕ is defined as follows. Each L_p being a 4-tuple of polynomials, it can be identified with a polynomial

$$g_{p,0}(u,v) + Xg_{p,1}(u,v) + Yg_{p,2}(u,v) + Zg_{p,3}(u,v),$$

where $g_{p,k}(u,v)$ denotes a polynomial expressed in the tensor product Bernstein basis:

$$g_{p,k}(u,v) = \sum_{i=0}^{\nu_1} \sum_{j=0}^{\nu_2} \alpha_{i,j}^{p,k} B_i^{\nu_1}(u) B_j^{\nu_2}(v),\ \alpha_{i,j}^{p,k} \in \mathbb{R}. \tag{2}$$

Then, setting $m_\nu := (\nu_1 + 1)(\nu_2 + 1)$, the $m_\nu \times r_\nu$-matrix

$$\mathbb{M}_\nu := M_{\nu,0} + XM_{\nu,1} + YM_{\nu,2} + ZM_{\nu,3}, \tag{3}$$

where

$$M_{\nu,k} := \begin{pmatrix} \alpha_{0,0}^{1,k} & \alpha_{0,0}^{2,k} & \cdots & \alpha_{0,0}^{r_\nu,k} \\ \alpha_{1,0}^{1,k} & \alpha_{1,0}^{2,k} & \cdots & \alpha_{1,0}^{r_\nu,k} \\ \vdots & \vdots & & \vdots \\ \alpha_{\nu_1,\nu_2}^{1,k} & \alpha_{\nu_1,\nu_2}^{2,k} & \cdots & \alpha_{\nu_1,\nu_2}^{r_\nu,k} \end{pmatrix}$$

is an MRep of degree $\nu = (\nu_1, \nu_2)$ for ϕ. It is an implicit representation of the supporting surface of the Bézier patch parameterized by ϕ because of the following *drop-of-rank property* [9]:

Assume that $\nu \geqslant (2d_1 - 1, d_2 - 1)$ or $\nu \geqslant (d_1 - 1, 2d_2 - 1)$, then for any point $P \in \mathbb{R}^3$, the rank of $\mathbb{M}_\nu(P)$ is equal to m_ν if and only if the point P does not belong to supporting surface of the Bézier patch parameterized by ϕ. The notation $\mathbb{M}_\nu(P)$ stands for the evaluation of $\mathbb{M}_\nu$ at P which is computed as a simple linear combination of 4 matrices (Eq. (3)).

Matrix representations also hold for both 2D and 3D rational Bézier curves [9]. They rely on classical linear algebra algorithms, such as computing a null space and determining the rank of a matrix. Therefore, in the context of numerical computations with NURBS models, we rely heavily on singular value decomposition (SVD), which is known to be one of the most reliable numerical tools for these operations. For the sake of clarity, we briefly recall its definition (see [19] for more details).

Let $A \in \mathbb{R}^{m \times n}$ be a real matrix and $m \leqslant n$ (if $m > n$ then simply apply what follows to A^T). The singular value decomposition (SVD) of A is a decomposition of the form $A = U\Sigma V^T$ where $U = [u_1, \ldots, u_m] \in \mathbb{R}^{m \times m}$ and $V = [v_1, \ldots, v_n] \in \mathbb{R}^{n \times n}$ are real orthogonal matrices and the matrix

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 & 0 & \ldots & 0 \\ 0 & \sigma_2 & \ddots & \vdots & \vdots & & \vdots \\ \vdots & \ddots & \ddots & 0 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & \sigma_m & 0 & \ldots & 0 \end{bmatrix} \in \mathbb{R}^{m \times n}$$

has non-negative diagonal elements appearing in the conventional decreasing order $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_m \geqslant 0$. The numbers σ_i, also denoted $\sigma_i(A)$, are referred to as the singular values of A while the vectors u_i and v_i are referred to as the left and right singular vectors of A, respectively. The *numerical rank* of A, within a given tolerance ϵ, is determined by analyzing the singular values of A. In our context we use the following definition

$$\operatorname{rank}(A, \epsilon) := \min \{k \ :\ \sigma_{k+1}(A)/\sigma_k(A) < \epsilon\}.$$

Having found the numerical rank of A, its *numerical null space* is obtained from the right singular vectors $v_{r+1}, \ldots, v_n$. For our experiments, $\epsilon = 10^{-6}$.

2.2. Intersection Algorithm

Hereafter, we describe the main steps of our algorithm for computing the intersection points between a line and a Bézier patch. We will use it in Section 4.2 to devise a meshing algorithm.
Input:

- A line R parameterized by $R(t) = O + t\mathbf{D}$ where O denotes a point on R, $\mathbf{D}$ a direction vector of R and t its parameter;
- A Bézier patch S with bi-degree (d_1, d_2), given by its control points and weights (Eq. (1)).

Output: All intersection points between R and S.

Step 1 (implicitization): Compute an MRep $\mathbb{M}_{\nu_0}$ of S as described in Section 2.1 with $\nu_0 = (2d_1 - 1, d_2 - 1)$ if $d_1 \leqslant d_2$ and $\nu_0 = (d_1 - 1, 2d_2 - 1)$ otherwise. This computation is done only once at the first occurrence and stored.

Step 2 (evaluation): Generate a (singular) pencil of matrices by evaluating $\mathbb{M}_{\nu_0}$ at all the points on R :

$$\mathbb{M}_{\nu_0}(R(t)) = A - tB$$

where A and B denote real matrices of size $m_{\nu_0} \times r_{\nu_0}$. By the drop-of-rank property, the intersection points between R and the supporting surface of S correspond to the values of the parameter t such that the rank of the pencil $A - tB$ is not equal to m_{ν_0}.

Step 3 (pencil reduction): Extract the regular part of the (singular) pencil $A - tB$ to get a non-degenerate square pencil of matrices $A' - tB'$ (see Section 2.3).

Step 4 (eigencomputation): Compute the generalized eigenvalues $t_1, \ldots, t_r$ of this pencil $A' - tB$ to get all the intersection points $P_i = R(t_i)$, $i = 1, \ldots, r$, between R and the supporting surface of S. Then, keep those points that lie within the convex hull of the control polygon of S.

Step 5 (inversion): For each intersection point P obtained in Step 4, compute all of its pre-images in the parameter space of the supporting surface of S (see Section 2.4).

Step 6 (filtering): For each intersection point P obtained in Step 4, check whether one of its pre-images (computed in Step 5) is inside the patch domain $[0, 1] \times [0, 1]$.

This algorithm is based on Busé's method [9]. However, to apply this method for meshing NURBS surfaces by means of Delaunay refinement (see Section 4), we must deal with two main classes of *degenerate cases* (see Table 3) that are not addressed in [9]. The first class of degenerate cases corresponds to the numerical analysis of some (nearly) tangential geometric configurations, where the pencil reduction step returns an empty pencil (Example 1). The second class of degenerate cases corresponds to intersection points with several pre-images in the parameter space of the Bézier patch S, in which case the original inversion method [9] does not apply (Example 2 and Figure 3). In the following we detail Step 3 and Step 5, that deal with these two classes of degenerate cases.

2.3. Pencil Reduction

The existence of the regular part of a singular pencil of matrices is a consequence of the Kronecker's form [4, §3.2]. Ba et al. [4, §3.3] propose a method, based on LU-decomposition, for extracting this regular part in the context of general curve/surface intersection problems by means of MRep. In our context, this extraction algorithm via SVD is simplified as Step 1-4 in the following algorithm; Step 5 is our treatment for degenerate cases where Step 1-4 fails to return a non-empty regular pencil.
Input: a singular pencil of $m \times n$-matrices $A - tB$, $m \leqslant n$.

Output: a regular pencil of $r \times r$-matrices $A' - tB'$, with B' invertible, which is equivalent to the regular part of the input pencil.

Step 1: Compute an SVD of B, $B = U_1 \Sigma_1 V_1^T$, and transform B into column echelon form: $BV_1 = [\, * \mid 0 \,]$. Then, apply the same transformation to A: $AV_1 = [\, * \mid A_1 \,]$.

Step 2: Compute an SVD of A_1, $A_1 = U_2\Sigma_2 V_2^T$, and transform A_1 into row echelon form:

$$U_2^T A_1 = \left[\frac{A_1'}{0}\right].$$

It follows that we obtain a new pencil $A' - tB'$ which is a reduction of the pencil $A - tB$:

$$U_2^T(A - tB)V1 = \left[\begin{array}{c|c} * & * \\ \hline A' - tB' & 0 \end{array}\right].$$

Step 3: Repeat steps 1 and 2 until B' is full column rank.

Step 4: If B' is not a square matrix, apply the above process to the transposed pencil ${A'}^T - t{B'}^T$.

In theory, the above process should always return a non-empty regular pencil whose eigenvalues correspond to the intersection points, unless the line is not intersecting with the supporting surface of the Bézier patch at finite distance, or is contained in it without crossing its singular locus. However, during our experiments we found that it may return an empty pencil for the first class of degenerate cases, that is to say for cases where the line is nearly tangential to, or nearly contained in the supporting surface of the Bézier patch (see Figure 1).

Step 5: To deal with these degenerate cases, we check whether the rank of the pencil $A - tB$ equals m; this can be done probabilistically at a random value of t, or deterministically at $m + 1$ distinct values of t. We then proceed as follows.

(a) If the rank of $A - tB$ is not m, then the line is entirely contained in the surface. In such a case, it is often useful to return the intersection points between the line and the Bézier patch boundary. These line/curve intersection problems are also solved by means of MRep. Observe that this approach also applies even if the patch boundary is trimmed: In this case the line is intersected with the trimmed curve whose control points are obtained by composition of the 2D-trimming curve and the patch parameterization.

(b) If the rank of $A-tB$ is m, choose a rank m square sub-pencil and compute its generalized eigenvalues. Then, by the drop-of-rank property these eigenvalues yield a superset of the intersection points we seek for. The correct ones can be extracted via the inversion step (Section 2.4). We mention that there are several methods to choose a rank m square sub-pencil of $A - tB$, one of them being to compute a column-reduction of the pencil and then keep the pivot columns. Note that these degenerate cases are more compute-intensive, as the size of the eigencomputation is larger than the size of the reduced pencil $A' - tB'$ (from Step 1-4) in non-degenerate cases.

Example 1. A line intersecting properly a bi-cubic Bézier patch is illustrated by Figure 1. The MRep of this patch is a 18×27-matrix. The pencil reduction step with this line fails because this line is asymptotically nearly contained in the surface. However, the generalized eigenvalues of a full rank 18×18 sub-pencil provide the correct intersection point.

2.4. Inversion

After obtaining the intersection points between the line and the Bézier patch at the end of Step 4, we must decide whether these points belong to the Bézier patch domain and if this patch is split from a trimmed NURBS surface, whether they are trimmed by the *pcurves* in the parameter space of the NURBS surface (see Section 4.2). Therefore, we need to compute the pre-images of the intersection points through the Bézier patch parameterization (1). This inversion problem is solved by Busé [9, §3.3] for cases where the intersection point has a unique pre-image on the surface. However, during our experiments we found that cases where the point has multiple pre-images are frequent (see Example 2, Figure 3 and also Table 3), including on canonical surfaces.

We propose a general inversion algorithm for extracting all the pre-images of an intersection point on the Bézier patch. It is based on the following property of MRep: Let $\mathbb{M}_\nu$ be an MRep of the Bézier patch and $P \in \mathbb{R}^3$ be a point such that $P = \phi(u_0, v_0)$ for some parameter value (u_0, v_0), then by construction the vector

$$\left[B_0^{\nu_1}(u_0)B_0^{\nu_2}(v_0), B_1^{\nu_1}(u_0)B_0^{\nu_2}(v_0), \ldots, B_{\nu_1}^{\nu_1}(u_0)B_{\nu_2}^{\nu_2}(v_0)\right]^T$$

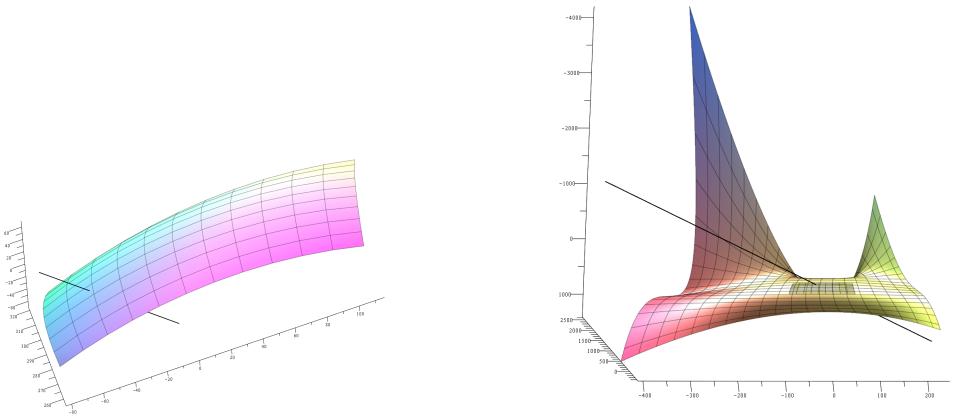


Figure 1: Degenerate case in pencil reduction. A bi-cubic Bézier patch defined on a unit domain (left) and its supporting surface plotted with domain $[-2,2]\times[-3.5,3]$ (right, zoomed out). The left Bézier patch is marked in gray in the supporting surface shown right. The line intersects the patch properly but is asymptotically nearly contained in the surface.

belongs to the null space of $\mathbb{M}_\nu(P)^T$. This property and the fact that the dimension of the null space of $\mathbb{M}_\nu(P)^T$ is controlled by the pre-images of P [9, Fact 4], lead to the following algorithm.
Input:

- A Bézier patch of degree (d_1,d_2) with parameters $(u,v)\in[0,1]\times[0,1]$ and an MRep $\mathbb{M}_\nu$ computed in the implicitization step (Section 2.2).
- A point P belonging to the supporting surface of the Bézier patch.

Output: All the pre-images of the point P.

Assumption: The Bézier patch has no singularity that is local to the parameters so that any point with multiple pre-images corresponds to a self-intersection of the surface with distinct parameters. This is not a restrictive hypothesis in the context of modeling with NURBS surfaces.

Step 1: Compute an SVD of $\mathbb{M}_\nu$ evaluated at P, $\mathbb{M}_\nu(P)=USV^T$, and deduce its (numerical) rank (Section 2.1) and the nullity of $\mathbb{M}_\nu^T(P)$ that we will denote by r. Then, let K be the submatrix of U corresponding to its last r columns. It is a $(\nu_1+1)(\nu_2+1)\times r$-matrix whose columns form a basis of an approximate kernel of $\mathbb{M}_\nu^T(P)$. By construction, its rows are indexed by the tensor product Bernstein basis (Eq. (2)):

$$\left\{B_0^{\nu_1}(u)B_0^{\nu_2}(v),B_1^{\nu_1}(u)B_0^{\nu_2}(v),\dots,B_{\nu_1}^{\nu_1}(u)B_0^{\nu_2}(v),B_0^{\nu_1}(u)B_1^{\nu_2}(v),\dots,B_{\nu_1}^{\nu_1}(u)B_{\nu_2}^{\nu_2}(v)\right\}. \tag{4}$$

Later in the algorithm, we require $\nu_2+1>r$ for eigencomputations. If this is not the case, we exchange the role of u and v. If this property is still not satisfied, then the MRep is rebuilt with $\nu_2=r$ (or $\nu_1=r$).

Step 2: Decompose matrix K into ν_2+1 submatrices of size $(\nu_1+1)\times r$ as follows. For all $i=0,\dots,\nu_2$ define the matrix K_i as the submatrix of K consisting of the rows indexed by the basis

$$\left\{B_0^{\nu_1}(u)B_i^{\nu_2}(v),B_1^{\nu_1}(u)B_i^{\nu_2}(v),\dots,B_{\nu_1}^{\nu_1}(u)B_i^{\nu_2}(v)\right\}=B_i^{\nu_2}(v)\times\left\{B_0^{\nu_1}(u),B_1^{\nu_1}(u),\dots,B_{\nu_1}^{\nu_1}(u)\right\}.$$

Then, build the two matrices

$$A:=\sum_{i=1}^{\nu_2}\frac{i}{d_2}K_i,\qquad B:=\sum_{i=0}^{\nu_2}K_i.$$

Observe that the rows of A and B are indexed by the basis

$$\left(B_0^{\nu_1}(u),B_1^{\nu_1}(u),\dots,B_{\nu_1}^{\nu_1}(u)\right). \tag{5}$$

Step 3: Let C be the change of basis matrix (changing to power basis) such that

$$\begin{pmatrix} 1 \\ u \\ \vdots \\ u^{\nu_1} \end{pmatrix} = C \times \begin{pmatrix} B_0^{\nu_1}(u) \\ B_1^{\nu_1}(u) \\ \vdots \\ B_{\nu_1}^{\nu_1}(u) \end{pmatrix}.$$

Build matrices $A' := CA$ and $B' := CB$. Then, define A'_r, resp. B'_r, to be the top $r \times r$ block of A', resp. B'.

Step 4: Check that the rank of B'_r equals r. If not, this means that at least two pre-images of P have the same u-coordinate value. The patch is then said to be *degenerate* in the u direction (Example 2). Return to Step 2 and exchange the role of u and v.

Step 5: Compute the eigenvalues and eigenvectors (via a simultaneous computation) of the pencil $A'_r - vB'_r$. This provides the v-coordinates of all the pre-images of P. Keep those values that are inside $[0, 1]$. If there is no such point, it means that P does not belong to the Bézier patch.

Step 6: Let $v_0 \in [0, 1]$ be a v-coordinate obtained at Step 5. Denote by V_0 a matrix whose columns form a basis of the eigenspace associated to v_0; its rank equates the number of pre-images of P with v-coordinate v_0. Now, compute the matrix $M_u = B'V_0$ and denote by T the top square block of M_u, and by S the top square block of M_u without its first row. Then, finally, the eigenvalues of the pencil $S - uT$ yield the u-coordinates of the pre-images of P with v-coordinate v_0.

This algorithm gets stuck at Step 4 if there exists axis aligned pre-images in both u and v directions. Although we did not encounter such cases in our experiments (notice that at least four pre-images are required), we explain below how to deal with these *very degenerate cases*.

At the end of Step 2, multiply K by a change of basis matrix that converts from the tensor product Bernstein basis (4) to the tensor product power basis

$$\left\{1, u, u^2, \ldots, u^{\nu_1}, v, vu, vu^2, \ldots, vu^{\nu_1}, \ldots, v^{\nu_2}, v^{\nu_2}u, \ldots, v^{\nu_2}u^{\nu_1}\right\}. \tag{6}$$

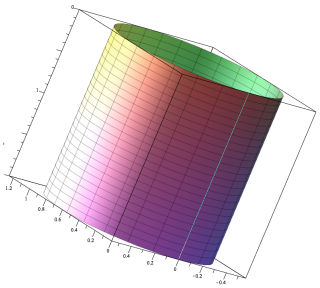
Thus, we get a new matrix $\overline{K}$ whose rows are indexed by the power basis (6). Since $\overline{K}$ has rank r, for K has rank r, there exists a rank r square submatrix of $\overline{K}$. Choose one and denote it by M_1; its rows are indexed by an ordered subset, denoted S, of (6).

Now, define the matrix M_u, resp. M_v, as the submatrix of $\overline{K}$ corresponding to the rows that are indexed by the subset $u.S$, resp. $v.S$ (multiplication is component wise). Notice that M_1 must be chosen so that $u.S$ and $v.S$ are included in (6). If this is not possible, then the MRep $\mathbb{M}_\nu$ is rebuilt by increasing the degree (ν_1, ν_2) to one of the following: $(\nu_1 + 1, \nu_2)$ if $u.S$ is not contained, or $(\nu_1, \nu_2 + 1)$ if $v.S$ is not contained, or $(\nu_1 + 1, \nu_2 + 1)$ if both are not contained. From this construction, we deduce that the eigenvalues of the pencil $M_u - tM_1$, resp. $M_v - tM_1$, are the u-coordinates, resp. the v-coordinates of all the pre-images of the point P. More generally, observe that for any pair $(\alpha, \beta) \in \mathbb{R}^2$, the eigenvalues of the pencil $\alpha M_u + \beta M_v - tM_1$ are the evaluation of the linear form $L(x, y) := \alpha x + \beta v$ at the pre-images of the point P (the previous cases correspond to $(\alpha, \beta) = (1, 0)$ and $(\alpha, \beta) = (0, 1)$). Note that, this treatment proposed for *potential very degenerate cases* involves more eigencomputations than the algorithm described in Step 1-6 for other cases, and thus is more time-consuming.

Example 2. Figure 2 illustrates a cylinder over a degree 5 Bézier plane curve with a seam edge corresponding to the lines $u = 0$ and $u = 1$ in the parameter space. The MRep $\mathbb{M}_{4,1}$ of this Bézier patch is a 10×15-matrix. When evaluated at the point $P = (0, 0, 1)$ on the seam, its rank drops to $r = 2$. Applying the above algorithm, A' and B' are 5×2-matrices (Step 3) and the computation of the generalized eigenvalues of the pencil $A'_2 - vB'_2$ (step 5) returns $v_0 = 0.5$ with multiplicity 2 (and V_0 is a 2×2 invertible matrix). Finally, T is the 2×2-matrix built from the two first rows of B', S is built from the second and third rows of B' and the computation of the generalized eigenvalues of the pencil $S - uT$ (step 6) returns

$$u_0 = 1.00000000000012, u_1 = -1.22196368144728 \times 10^{-13}.$$

Observe that these two pre-images of P are degenerate in the v direction.



$$\begin{bmatrix} 0.0 & 0.0 & 0.0 & 1.0 \\ 1.0 & 0.0 & 0.0 & 1.0 \\ 1.0 & 2.0 & 0.0 & 1.0 \\ -1.0 & 2.0 & 0.0 & 1.0 \\ -1.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 2.0 & 1.0 \\ 1.0 & 0.0 & 2.0 & 1.0 \\ 1.0 & 2.0 & 2.0 & 1.0 \\ -1.0 & 2.0 & 2.0 & 1.0 \\ -1.0 & 0.0 & 2.0 & 1.0 \\ 0.0 & 0.0 & 2.0 & 1.0 \end{bmatrix}$$

Figure 2: Degenerate case in inversion. A Bézier patch of degree $(5, 1)$ with a seam edge and its control point matrix. The points on the seam have two pre-images. More cases shown in Figure 3.

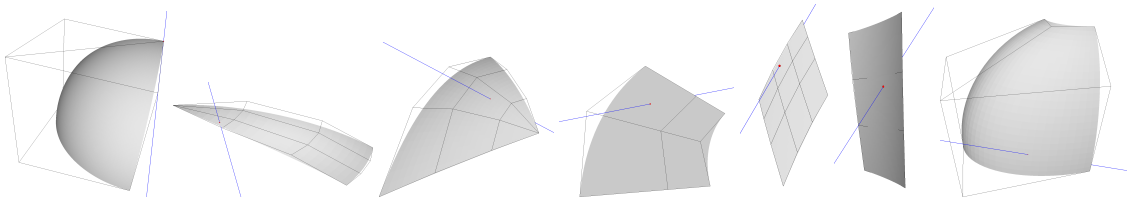


Figure 3: *Multiple Pre-images.* More examples of degenerate cases in inversion, where the intersection has multiple pre-images. The first case shows an intersection with infinite number of pre-images. Others show intersections that have multiple pre-images (degenerate in u or v direction) but only one valid, i.e. belonging to $[0, 1] \times [0, 1]$.

Another class of degenerate cases corresponds to configurations where a point P on the Bézier surface has an infinite number of pre-images (i.e. a plane curve with possibly some isolated points). In such cases, the above inversion algorithm does not apply, but these cases can be detected by comparison of the drops of rank of at most three MReps evaluated at P and built in consecutive degrees [9, §3.3, Fact 4]. Nevertheless, in the context of modeling with NURBS surfaces, in particular with Bézier patches, such a situation only appears when P is a point where a row of control points coalesce (Figure 3 (a)), because we are focusing on points P on the Bézier patch (not the supporting surface). Therefore, in practice these degenerate cases can be detected.

3. Evaluation and Comparison

Compared to intersection algorithms based on Newton iteration or subdivision techniques, our MRep-based algorithm appears to have a better accuracy, especially in difficult cases. This property follows from the use of standard tools of numerical linear algebra, but also from a careful treatment of degenerate cases in the algorithm itself (some examples of such difficult geometric cases are illustrated in Figure 4). However, the counterpart of the reliability of our algorithm is that it is compute-intensive on high degree patches, most of the overall time (95%) being spent in SVD and generalized eigenvalue computations. In the following, we first give some theoretical references to explain why our new intersection algorithm appears to be more robust than existing methods. Then, we give some experiments for comparing timings and accuracy with two other methods based on Newton iteration and subdivision techniques respectively.

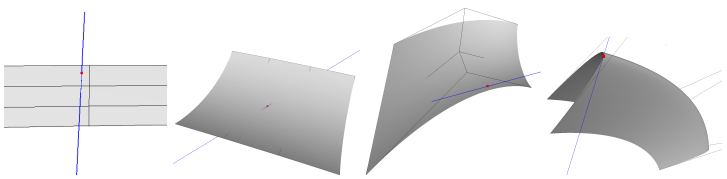


Figure 4: *Subdivision-based Method.* Examples where the subdivision-based method returns inaccurate intersection (accuracy less than 10^{-10}). In most cases, the ray is near tangent with the patch.

3.1. Numerical Aspects

In practice, line/Bézier patch intersection algorithms are applied with numerical computations. In addition, it is also applied with approximated input data, meaning that the control points of the input Bezier patches are usually exact up to a certain numerical precision. Therefore, it is important to ensure that all the steps of our MRep-based intersection algorithm are numerically stable. This is indeed the case because this algorithm has been designed to rely on standard tools of numerical linear algebra that have been widely studied and experienced in many various application domains. Without going into the details, we provide some references where theoretical results on the numerical analysis of the main steps of our algorithm, as described in Section 2.2, can be found. These results strengthen the very good numerical accuracy and robustness of our approach that we have observed in practice so far (see Table 2).

The numerical computation of MRep involves the computation of numerical null spaces of matrices: this is discussed and illustrated in [9, Section 5.2]. In addition, it is shown in [9, Section 5.3] that the pencil of matrices obtained in Step 2 is numerically stable with respect to the expected drop-of-rank property which is fundamental in our method. The extraction of the regular part of a pencil of matrices as done in Step 3 is based on a method that has been introduced by Van Dooren et al. in [15]. As a consequence of the results proved in [15, Section 4], Step 3 is numerically stable, more precisely is numerically backward stable (meaning that the computed regular pencil is the exact regular pencil of a perturbation of the input pencil of matrices). A detailed numerical analysis of the eigencomputation that is performed in Step 4 can be found in [12, Section 3]. Actually, [12, Theorem 3.4] yields a precise statement for bounding the numerical error which is made when computing the eigenvalues of the regular part of a pencil of matrices, that is to say the combination of our Step 3 and Step 4. With Step 5, we are back to the numerical analysis of the computation of the kernel of a matrix (see [9, Section 5.2 and Section 5.4] and [19, Section 8.6.1]).

3.2. Quantitative Comparisons

We now compare quantitatively our MRep-based line/Bézier intersection method with the two other types of intersection methods, namely the Newton iteration method [24] and the subdivision based intersection method provided in the SISL library [34]. SISL is a popular library, and is the only publicly available subdivision method we could find. Our algorithms have been implemented in C++. It uses LAPACK for SVD and generalized eigenvalue computations, EIGEN [20] for matrix manipulations. All timings are measured on a desktop PC equipped with Intel® Core™ i5-4570 CPU @ 3.20GHz, 15.6 GB memory.

Table 1 records the cost to initialize an MRep for a Bézier patch. The time increases with the patch degree as a consequence of the increasing size of the matrix built for the patch. Note however that the MRep is initialized only once. As shown in the last column of Table 2, our line/Bézier patch intersection cost highly depends on the matrix size of MRep, which itself depends on the patch degree (Section 2.1). Our method returns intersections with geometric accuracy above 10^{-10} in general.

For comparison we sample the Bézier patch, shoot random rays passing through the samples and compare the computed intersections with the samples: both the parameter value and the 3D point. The two alternate approaches (Newton iteration and subdivision) proceed in parameter space while ours compute in geometric space. For rigorous comparison we thus consider that an intersection fails to return an intersection within the desired accuracy when both the parameter value and the 3D point location are not within the desired accuracy. Note that the flattening step of the Newton iteration approach splits on average a patch into 100 sub-patches.

Table 2 compares the average computation time per ray and the rate of inaccurate intersections with desired accuracy 10^{-10}. As depicted by Figure 4, the subdivision based method [34] is likely to return inaccurate intersections in near tangent situations.

Degree	Time (ms)	Degree	Time (ms)	Degree	Time (ms)
(2,2)	0.13	(2,3)	0.35	(3,3)	0.83
(2,5)	1.5	(3,5)	3.9		

Table 1: *Timings.* Time to compute a MRep of a Bézier patch in the implicitization step. The MRep matrix is only initialized once.

Degree	Newton Time (ms) / Inaccurate rate	Subdivision	Ours
(2,2)	0.025 / 7.6e-3	0.0677 / 3.46e-4	0.097 / 0
(2,3)	0.032 / 0.022	0.2136 / 2.1e-3	0.223 / 0
(3,3)	0.041 / 0.012	0.338 / 2.37e-3	0.42 / 0
(2,5)	0.038 / 0.019	0.228 / 5.37e-4	0.518 / 0
(3,5)	0.047 / 0.019	0.389 / 2.12e-3	1.12 / 0

Table 2: *Comparisons.* Comparison of the computational time and accuracy (10^{-10}) against the Newton iteration [24] and the subdivision-based approach from the SISL library [34].

4. Application to Meshing

We now apply our intersection algorithm to the meshing of NURBS models using the Delaunay-based mesh generation framework from the CGAL library [23]. We first define some notations.

Notations. A *NURBS model* is composed of a group of NURBS surfaces. A *NURBS surface* is defined over a rectangle parameter domain [27, §4.4] with trimming curves (also defined in the parameter space, denoted *pcurves*, see Figure 5c) that are parametric and oriented. The orientation of a trimming curve is used to include or exclude the corresponding enclosing area. A *NURBS surface* can be further split into a collection of *Bézier patches* (Figure 5b). An *edge* refers to the boundary of a NURBS surface. It can be either an untrimmed boundary or a *trimmed edge*. When two NURBS patches are adjacent via an edge we refer to a *common edge*. When an edge corresponds to an open boundary of the whole input surface we refer to a *boundary edge*. Where a surface is self-adjacent we refer to a *seam edge*. Finally, a common edge may be either sharp or smooth depending on the continuity of normals across that edge.

The most common approach for meshing NURBS surfaces consists in adaptive sampling and triangulation in 2D parameter space [22, 26, 28, 33, 2, 21, 17, 3, 7], and then mapping it in 3D. While parametric surfaces enable efficient and accurate evaluation of geometric properties within each individual patch, meshing the complete union of NURBS patches remains a scientific challenge for the following reasons:

- *Common edges*. Assume a perfect NURBS model where all patches meet so that the common edges are watertight. Seamless meshing requires that all sub-meshes (one per patch) are conforming at the common edges [3]. However, conforming itself adds unnecessary constraints along these common edges which may lower the final mesh quality and approximation accuracy. In addition, the trimmed NURBS representation is often not perfectly watertight [30, 32].

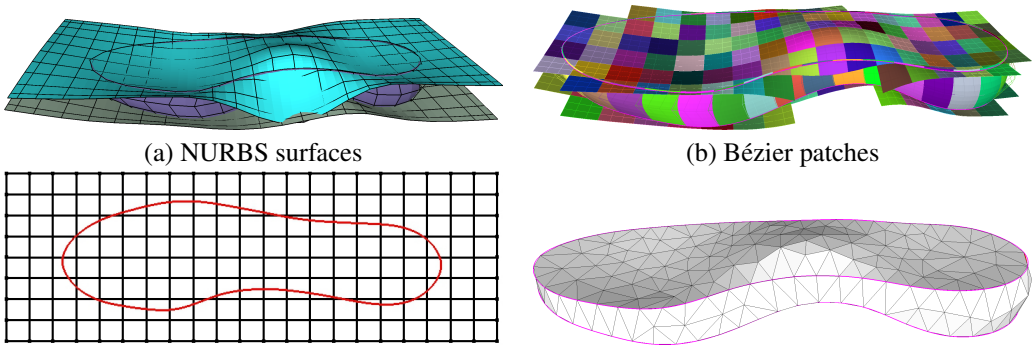
- *Distortion*. Meshing in parameter space is highly dependent on the properties of the parameterization. Although the geometric properties of the mesh can be measured in embedding space [36, 28, 2], they are controlled in parameter space via local evaluations.

- *Sizing*. Meshing in parameter space is constrained to generate at least one or two triangles per patch, which relates the maximum size of the triangles to the patch decomposition. We seek an approach where mesh sizing is less constrained.

4.1. Delaunay Mesh Generation

We adopt a greedy mesh generation approach based on Delaunay refinement and filtering [23, 8]. We first generate an initial 3D Delaunay triangulation based on few points sampled on the input surface and its sharp features such as crease edges and corners if any. We then refine the triangulation by inserting points which are computed as 3D intersections between line segments (the duals of Voronoi faces) and the input surface. As the mesh is generated in 3D, such meshing process does not depend on the decomposition into patches and their parameterizations. This approach requires devising a reliable line/NURBS intersection method, referred to as an *oracle*. However, and as acknowledged in previous work [8], the intersection is the major bottleneck.

Three user-specified parameters are required to control the properties of the final mesh:



(a) NURBS surfaces (b) Bézier patches

(c) *pcurve* (in red) (d) $\epsilon = 0.05$

Figure 5: *Sandal sole*. (a) Control meshes depicted with black lines. (b) 214 Bézier patches after splitting and filtering (via *pcurve*) (c) Knot lines (in black) and trimming curve (in red) in parameter space of the NURBS surface on top. (d) Feature-preserving meshing result.

- Facet size: upper bound (R) on the radii of the surface Delaunay ball;
- Facet shape: lower bound (θ) on the surface facet angle;
- Approximation: upper bound (ϵ) on the approximation error of the surface facets.

In all experiments we use R to control the minimum distance between random sample points and the density of sample points along the boundary and crease edges.

4.2. Intersection Oracle

The Delaunay refinement approach requires the process described in Section 2 in order to compute the *farthest* intersection point between a line query (the dual Voronoi edge of a facet, i.e., a line segment or a ray) and the input trimmed NURBS surfaces. More specifically, we take as input a collection of trimmed NURBS surfaces $(N_i)_{i=1,\dots,n}$, together with their parameter spaces $(\Omega_i)_{i=1,\dots,n}$. The *pcurves* in Ω_i are NURBS curves. We generate as output the farthest intersection point between the query and the surfaces. To extend the line/Bézier intersection algorithm described in Section 2.2, we detail next the additional steps required to handle the intersection with trimmed NURBS surfaces.

Pre-processing step. As MRep only applies to Bézier patches, each NURBS surface N_i is first split into a set of Bézier patches by knot insertions [27, §5.3] (Figure 5b). These Bézier patches are then filtered with respect to the *pcurves* in Ω_i: Bézier patches entirely culled are filtered out, the others being marked as either partially culled or untrimmed. To improve efficiency, we then build a hierarchical data structure (an axis-aligned bounding box tree referred to as AABB tree) where each leaf of the tree is the bounding box of a Bézier patch. For each intersection query we first perform an intersection test between the query and the boxes of the tree. For each intersected box we further test whether the query intersects the convex hull of the Bézier patch contained in that box. The full line/Bézier intersection test is then performed only for the patches whose convex hull intersects with the query, by means of the algorithm detailed in Section 2.

Culling by pcurves. For each line/Bézier intersection point P with its parameter value ω (computed in Step 5, Section 2.2) we proceed as follows: if this Bézier patch is untrimmed, P is returned as intersection point. Otherwise, the following culling step is required. Mapping ω back to the original NURBS surface domain Ω yields its NURBS parameter value ω'. We then perform in Ω an inside/outside test against the *pcurves*. More specifically, the test is performed by shooting a random ray from ω' and finding its first intersection with the *pcurves*. To this end, the same methodology as for intersecting surfaces by means of MRep but in 2D is also used, applying similar accelerations via bounding boxes and convex hulls. The intersection test is finalized by checking whether the scalar product of the *pcurve* normal at this first intersection and the direction of the random ray is negative. Finally, we sort the intersection points according to their parameter values along the supporting line of the query, and return the farthest point.

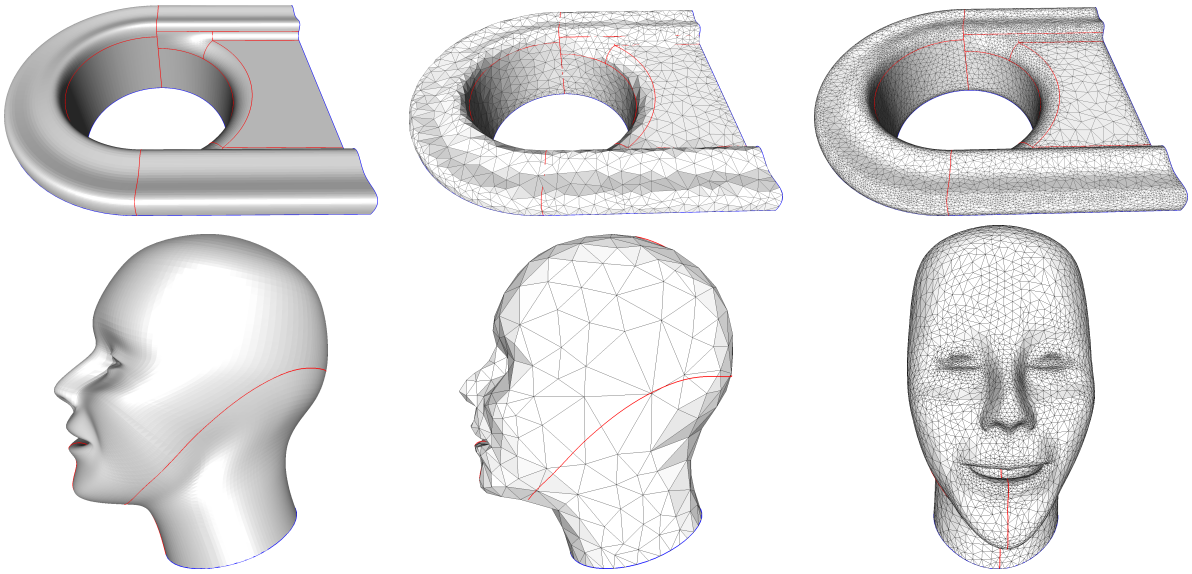


Figure 6: Seamless meshing. Top: meshing with $\epsilon = 0.005$ and 0.0005. The initial point set is generated by sampling along the open boundary. Bottom: meshing across smooth edges (red). The initial point set is generated by sampling along the boundary edge (blue). Meshes generated with $\epsilon = 0.01, 0.001$ (side and front view).

4.3. Experiments

We can generate isotropic triangle meshes ranging from uniform to adaptive, and from coarse to fine, through adjusting the parameters ϵ and R (scaled by the longest diagonal of the bounding box). The approximation accuracy is controlled through ϵ, while the maximum local mesh sizing is controlled through R. For a fixed R, decreasing ϵ generates adaptive meshes with small triangles on high curvature areas. For isotropic meshing the shape of the triangles is controlled by the lower bound on the triangle angles θ, set by default to 25°. Figure 6 depicts meshes that are oblivious to the input patch layout, with more degrees of freedom to generate coarse meshes.

Figure 7 illustrates seamless and adaptive meshing, despite the presence of degenerate patches with multiple pre-images. For this closed smooth surface the initial point set is generated by random ray shooting until finding ten points sufficiently far apart in 3D.

Our framework provides control on the behavior of the mesh generator on the edges of the NURBS surface. On the smooth *Cup* surface we can generate a mesh either seamlessly or while preserving the smooth common edges between NURBS patches (Figure 8).

Sharp creases can be preserved when properly detected and passed to the mesh generator in the form of parametric curves (Figures 9, 10 and 11). The sharpness of a common edge between two adjacent NURBS patches is detected by walking simultaneously on both sides of the common edge, evaluating the normals and measuring their angle deviation. A common edge is considered sharp when the deviation is larger than a user-specified angle tolerance, set to 5° in all feature-preserving experiments. Meshing is initialized by sampling a handful of points along the sharp creases, then the mesh generator first refines only along the sharp creases via the ball-protection paradigm [23] before resuming the refinement of smooth parts via Delaunay refinement.

As we view the set of NURBS patches as a single surface in embedding space, we avoid the issues of dealing with each patch independently in parameter space. We compare our meshes with the ones generated by Rhino [29] and GMSH [18]. Figure 11 illustrates the fact that our method is less constrained by the initial path layout and parameterizations. As depicted in Figure 12(d), meshing per patch in parameter space could imply non-conforming elements along common edges. Note that as the *Rhino* software is not aimed at isotropic meshing we do not compare this criterion.

Figure 13 illustrates our intersections at work for generating an isotropic tetrahedron mesh on the *mech part* model. For volume meshing the oracle queried by the mesh generator computes all intersections between a random ray shot from a query point, in order to deduce the inside/outside location.

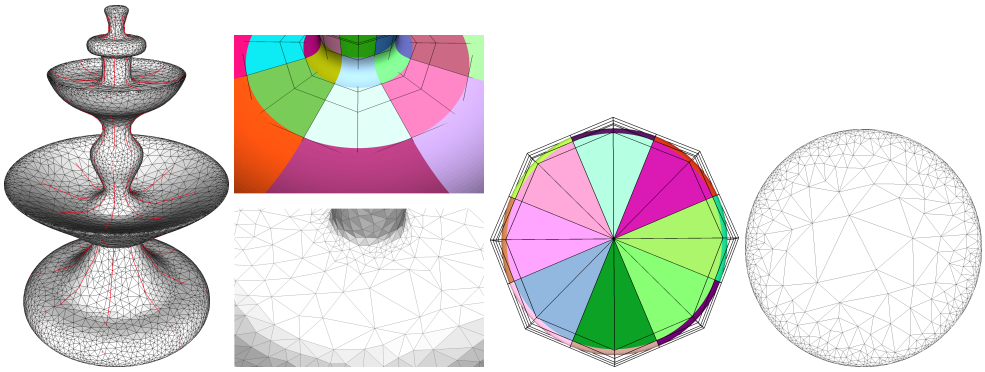


Figure 7: *Fountain*. Seamless meshing with $\epsilon = 0.001$. On the fan-shaped Bézier patches (degree (2,2), see closeup on the right), the points have four pre-images and are degenerate in one direction (Section 2.4).

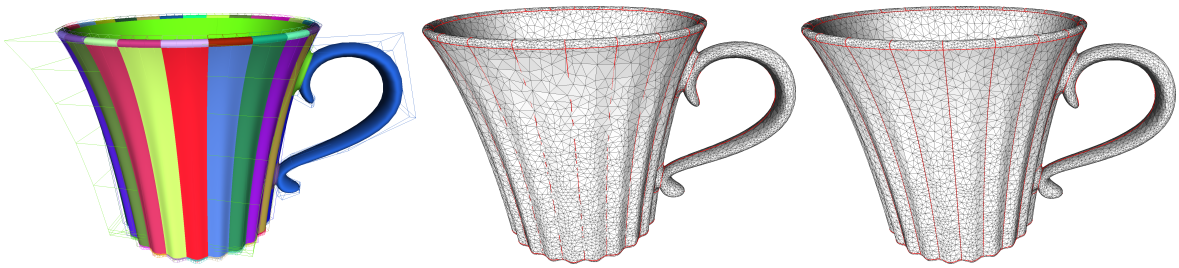


Figure 8: *Cup*. Left: 102 NURBS surfaces. Middle: Seamless meshing with $\epsilon = 0.001$. The initial point set (10 points) is randomly generated. Right: Edge-preserving meshing with $\epsilon = 0.001$. The initial point set is generated by sampling the common edges (except closed seams).

Table 3 records, for each input model, the number of NURBS patches, the total number of Bézier patches after splitting, the input parameter ϵ, the number of facets in the result mesh and the total meshing time. The time highly depends on the patch degree and ϵ. All experimented models have degrees $\leq$ (3,5). The last 4 columns of Table 3 record the statistics on the number of degenerate cases. The first type of degenerate cases is about geometry: it happens where the line is nearly tangential to or contained in the supporting surface of the Bézier patch. The second type of degenerate cases happens during inversion, when the intersection point has multiple pre-images on the supporting surface.

- Type I(a): The pencil reduction fails to return a regular pencil;
- Type I(b): Type I(a) that misses a true intersection with the Bézier patch, while our method via computing a superset detects it;
- Type II(a): The intersection has multiple pre-images;
- Type II(b): Type II(a) with multiple pre-images degenerated in u, or in v, or in both directions.

Note that *Mech* and *Hole* have nonzeros in Type I(a) but zeros in Type I(b) because they have surfaces such as planes or cylinders where the line is almost parallel to the supporting surface without intersecting the patch.

4.4. Limitations

Our approach is reliable but compute-intensive on high degree patches. The Delaunay mesh generation framework comes with theoretical guarantees over the topology and geometric approximation error, when several conditions are met [6]. These conditions are not always met by imperfect input models. Another requirement is to specify a proper sizing function. More specifically, the mesh sizing must be set locally to a fraction of the local feature size (LFS). For inputs with close surface sheets such sizing may be very small and hence lead to overly dense meshes. For these cases the parametric approaches relying solely on a local curvature estimate may yield coarser meshes [7], but are constrained by the local NURBS patch layout.

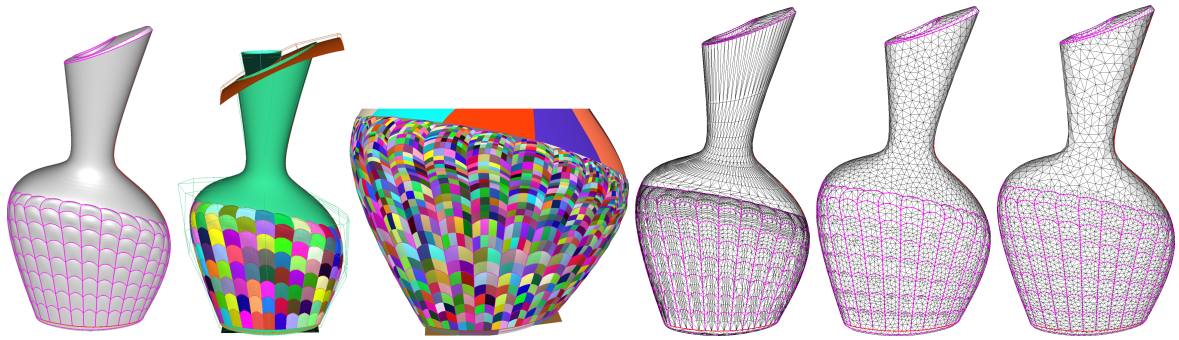


Figure 9: *Perfume bottle*. Top: input model, 255 NURBS surfaces, closeup on the Bézier patches (3786 overall). Bottom: mesh generated by Rhino (left), GMSH (middle), and our feature-preserving mesh with $\epsilon = 0.005$ (right). 12k triangles in each mesh.

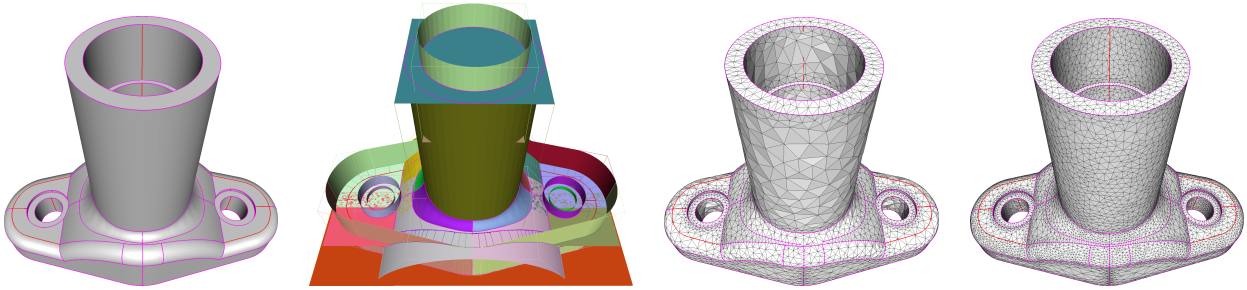


Figure 10: *Mech part*. From left to right: input NURBS model, 41 NURBS patches (122 Bézier patches, not shown), Feature preserving mesh with $\epsilon = 0.01$ and $\epsilon = 0.0005$. Sharp edges are depicted in pink, smooth edges in red.

Model	#N	#B	ϵ	#F	Time	Case I(a)	Case I(b)	Case II(a)	Case II(b)
Fountain	8	232	0.001	19k	25s	0	0	15.5%	6.64%
Sandal Sole	3	214	0.05	0.6k	5s	0	0	0	0
Hole	11	177	0.005	2k	4s				
			0.0005	19k	51s	0.13%	0	0	0
Mech Part	41	122	0.01	8k	33s				
			0.0005	22k	1m21s	2.86%	0	0	0
Head	2	472 (28%)	0.01	0.9k	5s				
			0.001	9k	43s	0	0	0	0
Cup	102	350 (65%)	0.001	30k	5m	0	0	0.54%	0.42%
			0.001	28k	4m51s				
Scoop	15	1411	0.001	7k	1m4s	2.3%	0.62%	0.1%	0.07%
Penguin	18	394	0.005	9k	1m9s	0	0	1.13%	1.13%
Perfume Bottle	255	3786 (2%)	0.005	12k	6m45s	2.5%	0.16%	0	0

Table 3: *Parameters, timings and ratios of degenerate case.* #N denotes the number of NURBS patches, #B the number of Bézier patches after splitting and filtering (also the proportion of degree (3,5) patches if any), ϵ the surface approximation parameter, and #F the number of triangle facets in the output mesh. R is set by default to 0.2. Both R and ϵ are expressed in ratio of the length of bounding box diagonal. Most patches have degree (3,3), (2,3) and (2,5). The ratios of degenerate cases during meshing are recorded in the last 4 columns. The ratio is over the total number of ray/surface intersection computations during meshing.

5. Conclusions

We introduced a reliable line/surface intersection method for trimmed NURBS surfaces and applied it to isotropic triangle meshing. We leverage a recent matrix-based implicit representation of NURBS surfaces to compute the intersections between a line query and a set of Bézier patches derived from the input NURBS surface. Our intersection approach also applies in 2D, for culling complex trimming curves in 2D parameter space. Such reliable intersection is

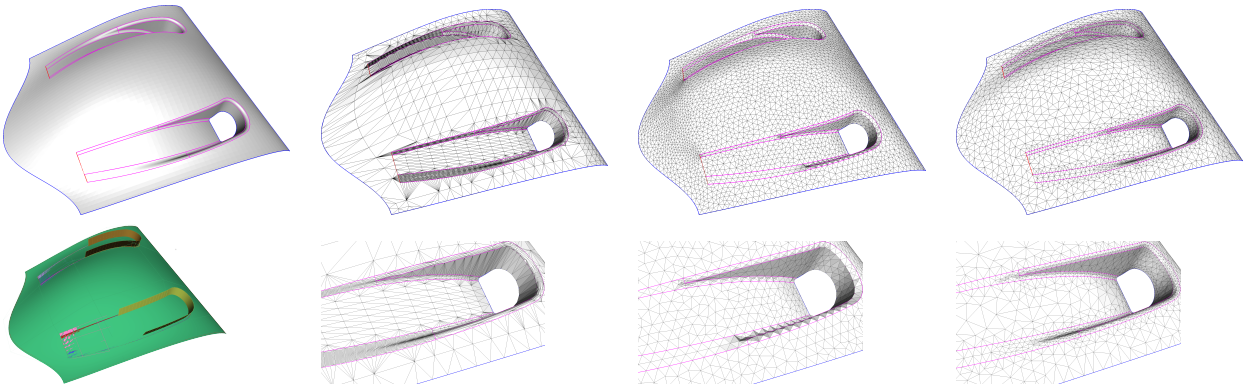


Figure 11: *Scoop*. Top: NURBS model with sharp edges depicted in pink, smooth edges in red, open boundary in blue; mesh (7k triangles) generated by Rhino, GMSH and our approach ($\epsilon = 0.001$). Bottom: NURBS surfaces, closeups on above meshes.

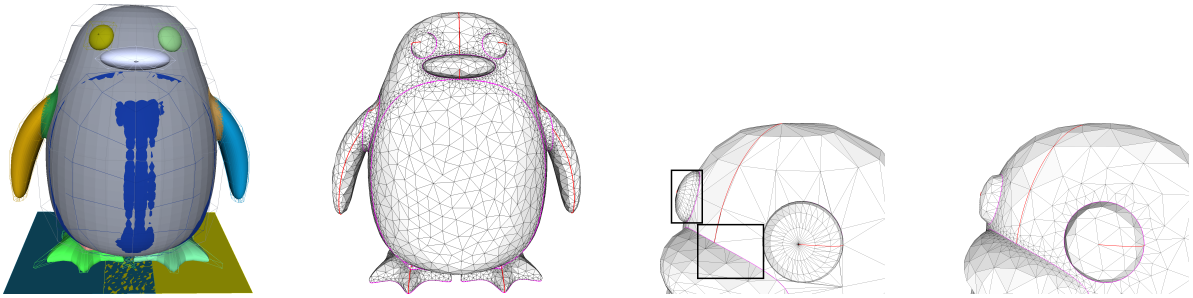


Figure 12: *Penguin*. Top: input model with 17 NURBS surface and our result with $\epsilon = 0.005$. Bottom: closeup of mesh generated by Rhino [29] with mismatched triangles along the sharp edges (left), and our conforming mesh (right). The blue spot on the belly is due to two overlapping surfaces, and the front of the grey surface is trimmed off.

particularly well suited to Delaunay-based meshing of NURBS models. As special care is taken to handle degenerate cases such as intersection points with multiple pre-images, we can mesh challenging input surfaces such as patches with seams.

As future work, we will investigate a principled way to estimate the local feature sizing of NURBS surfaces in order to compute suitable mesh sizing functions. A challenging problem is to devise a mesh generation algorithm with high robustness to defect-laden inputs such as gaps and intersecting surfaces along trimming curves. Ideally we seek for an algorithm with high stability properties, i.e., in which the output meshes degrade gracefully against the amount of input defects. Finally, we also plan to investigate the applicability of our MRep-based approach to the problem of finding orthogonal projections of query points on curves and surfaces.

References

[1] O. Abert, M. Geimer, and S. Muller. Direct and fast ray tracing of NURBS surfaces. In *IEEE Symposium on Interactive Ray Tracing 2006*, pages 161–168, Sept 2006.

[2] Marco Attene, Bianca Falcidieno, Michela Spagnuolo, and Geoff Wyvill. A mapping-independent primitive for the triangulation of parametric surfaces. *Graphical Models*, 65(5):260–273, 2003.

[3] R. Aubry, S. Dey, E.L. Mestreau, B.K. Karamete, and D. Gayman. A robust conforming NURBS tessellation for industrial applications based on a mesh generation approach. *Computer-Aided Design*, 63(0):26 – 38, 2015.

[4] Thang Luu Ba, Laurent Busé, and Bernard Mourrain. Curve/surface intersection problem by means of matrix representations. In *Proc. Conf. Symbolic Numeric Computation*, SNC '09, pages 71–78. ACM, 2009.

[5] Davide Bianchi, Manuel Sánchez del Río, and Claudio Ferrero. Ray tracing of optical systems using NURBS surfaces. *Physica Scripta*, 82(1):015403, 2010.

[6] Jean-Daniel Boissonnat and Steve Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67(5):405–451, September 2005.

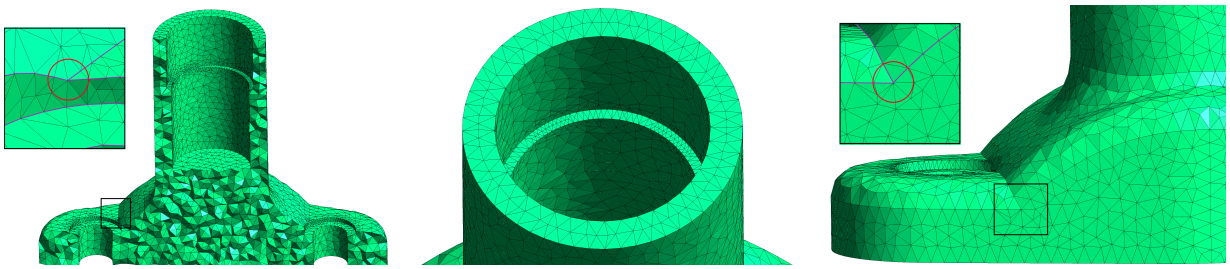


Figure 13: *Volume meshing.* An isotropic tetrahedron mesh generated via Delaunay refinement with top and front views and closeups on two cusps (marked in red circles with feature lines highlighted in pink).

[7] Jonathan R. Bronson, Joshua A. Levine, and Ross T. Whitaker. Particle systems for adaptive, isotropic meshing of cad models. *Engineering with Computers*, 28(4):331–344, 2012.

[8] Oleksiy Busaryev, Tamal K. Dey, and Joshua A. Levine. Repairing and meshing imperfect shapes with Delaunay refinement. In *SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 25–33, 2009.

[9] Laurent Busé. Implicit matrix representations of rational Bézier curves and surfaces. *Computer-Aided Design*, 46:14–24, 2014.

[10] Swen Campagna, Philipp Slusallek, and Hans-Peter Seidel. Ray tracing of spline surfaces: Bézier clipping, Chebyshev boxing, and bounding volume hierarchy – a critical comparison with new results. *The Visual Computer*, 13(6):265–282, 1997.

[11] Siu-Wing Cheng, Tamal K. Dey, and Jonathan Shewchuk. *Delaunay Mesh Generation*. Chapman & Hall/CRC computer and information science series, 1st edition, 2012.

[12] James Demmel and Bo Kagstrom. The generalized schur decomposition of an arbitrary pencil a–&lgr;b—robust software with error bounds and applications. part i: Theory and algorithms. *ACM Trans. Math. Softw.*, 19(2):160–174, June 1993.

[13] Tor Dokken. Finding intersections of b-spline represented geometries using recursive subdivision techniques. *Comput. Aided Geom. Des.*, 2(1-3):189–195, 1985.

[14] Tor Dokken, Vibeke Skytt, and Anne-Marie Ytrehus. Recursive subdivision and iteration in intersections and related problems. In *Mathematical Methods in Computer Aided Geometric Design*, pages 207–214. Academic Press, 1989.

[15] Paul Van Dooren and Patrick Dewilde. The eigenstructure of an arbitrary polynomial matrix: computational aspects. *Linear Algebra and its Applications*, 50(0):545 – 579, 1983.

[16] Alexander Efremov, Vlastimil Havran, and Hans-Peter Seidel. Robust and numerically stable Bézier clipping method for ray tracing NURBS surfaces. In *Proc. of SCCG '05*, pages 127–135. ACM, 2005.

[17] Pascal J. Frey and Paul L. George. *Mesh Generation: Application to Finite Elements*. ISTE, 2007.

[18] Christophe Geuzaine and Jean-Franois Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. http://geuz.org/gmsh/, 2014.

[19] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins University Press, third edition, 1996.

[20] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[21] Michael Guthe, Aákos Balázs, and Reinhard Klein. GPU-based trimming and tessellation of nurbs and t-spline surfaces. *ACM Trans. Graph.*, 24(3):1016–1023, July 2005.

[22] Bernd Hamann and Po-Yu Tsai. A tessellation algorithm for the representation of trimmed NURBS surfaces with arbitrary trimming curves. *Computer-Aided Design*, 28(6-7):461–472, June 1996.

[23] Clément Jamin, Pierre Alliez, Mariette Yvinec, and Jean-Daniel Boissonnat. CGALmesh: a generic framework for delaunay mesh generation. *ACM Trans. Math. Software*, October 2014.

[24] William Martin, Elaine Cohen, Russell Fish, and Peter Shirley. Practical ray tracing of trimmed NURBS surfaces. *J. Graph. Tools*, 5(1):27–52, January 2000.

[25] Rohit Nigam and P. J. Narayanan. Hybrid ray tracing and path tracing of Bézier surfaces using a mixed hierarchy. In *Proc. of ICVGIP '12*, pages 1–8. ACM, 2012.

[26] Leslie A. Piegl and Arnaud M. Richard. Tessellating trimmed NURBS surfaces. *Computer-Aided Design*, 27(1):16–26, 1995.

[27] Leslie A. Piegl and Wayne Tiller. *The NURBS Book (2nd Ed.)*. Springer-Verlag, New York, 1997.

[28] Leslie A. Piegl and Wayne Tiller. Geometry-based triangulation of trimmed NURBS surfaces. *Computer-Aided Design*, 30(1):11–18, 1998.

[29] Rhino. http://www.rhino3d.com/tutorials, 2014.

[30] Thomas W. Sederberg, Thomas G. Finnigan, Xin Li, Hongwei Lin, and Heather Ipson. Watertight trimmed NURBS. *ACM Trans. Graphics*, 27(3):79, August 2008.

[31] Thomas W. Sederberg and T. Nishita. Curve intersection using Bézier clipping. *Computer-Aided Design*, 22(9):538–549, 1990.

[32] Jingjing Shen, Jiří Kosinka, Malcolm A. Sabin, and Neil A. Dodgson. Conversion of trimmed NURBS surfaces to Catmull-Clark subdivision surfaces. *Computer Aided Geometric Design*, 31(7-8):486–498, 2014.

[33] Kenji Shimada, Atsushi Yamada, and Takayuki Itoh. Anisotropic triangular meshing of parametric surfaces via close packing of ellipsoidal bubbles. In *Proc. 6th International Meshing Roundtable*, pages 375–390, 1996.

[34] The sintef spline library. https://www.sintef.no/projectweb/geometry-toolkits/sisl/.

[35] Jane Tournois, Camille Wormser, Pierre Alliez, and Mathieu Desbrun. Interleaving delaunay refinement and optimization for practical isotropic tetrahedron mesh generation. *ACM Trans. Graph.*, 28(3):75:1–75:9, July 2009.

[36] Joseph R. Tristano, Steven J. Owen, and Scott A. Canann. Advancing front surface mesh generation in parametric space using a Riemannian surface definition. In *Proc. 7th International Meshing Roundtable*, pages 429–445, 1998.