A Geometric Algorithm for Ray/Bézier Surfaces Intersection using Quasi-interpolating Control Net

Yohan D. Fougerolle[1], Sandrine Lanquetin[1], Marc Neveu[1], and Thierry Lauthelier[2]
[1] Le2i Lab, UMR CNRS 5158, University of Burgundy
[2] The Bakery

Abstract

In this paper, we present a new geometric algorithm to compute the intersection between a ray and a rectangular Bézier patch. The novelty of our approach resides in the use of bounds of the difference between a Bézier patch and its quasi-interpolating control net. The quasi-interpolating polygon of a Bézier surface of arbitrary degree approximates the limit surface within a precision that is function of the second order difference of the control points, which allows for very simple projections and 2D intersection tests to determine sub-patches containing a potential intersection. Our algorithm is simple, because it only determines a 2D parametric interval containing the solution, and efficient because the quasi-interpolating polygon is directly computed, which avoids both minimum or maximum evaluations of the basis functions or complex envelops construction.

1 Introduction

For the past decades, ray tracing has been intensively studied and has found numerous applications in computer graphics, medical imagery, visualization, and entertainment. In ray tracing systems, the major issues concern space partitions, rays coherency, and ray/primitive intersection. For complex scenes, often modeled by several thousands of surfaces, such as Bézier, NURBS or subdivision surfaces, the efficiency and the memory load of the surface/ray intersection algorithm quickly becomes crucial. We present in this paper a new algorithm for free-form surfaces/ray intersection computation represented by rectangular Bézier surfaces.

Most of the related works can be classified into two categories: free-form surfaces ray tracing, more specifically algorithms for determining patch ray intersection, and envelopes/bounding box construction and basis functions bounds. Several approaches have been proposed to ray trace parametric surfaces, such as Bézier, NURBS, or subdivision surfaces. Campagna *et al.* [3] optimized and extended to NURBS the Bézier clipping algorithm proposed by Nishita *et al.*, where the algorithm identifies and cuts away regions of the patch that do not intersect the ray. Wang *et al.* [10] combined Newton iterations, Bézier-clipping, and ray coherency to speed up Nishita's algorithm. Kobbelt *et al.* [4] suggested a necessary theory for implementing ray tracing of subdivision surfaces using oriented bounding boxes. Unfortunately, this algorithm is slow, has to handle several special cases, and the construction of the oriented bounding boxes may be unsafe, as shown by Peters and Wu in [8]. Aside from direct intersection or collision tests, Benthin *et al.* proposed several strategies to optimize ray tracing algorithms [1, 2]. More specifically, they showed that a careful choice between well known algorithms, appropriate data structures, and implementations may lead to speeds up in an order of a magnitude. Algorithms that directly handle free-form surface, such as the works of Benthin *et al.* to ray trace Bézier surfaces [2], or Müller *et al.* for Loop subdivision surfaces, are not yet wild spread over our community and are not yet available in commercial products.

A fundamentaly different approach consists in constructing tight envelops of the surface patch to accelerate the intersection tests. The core of these methods is to define sharp function bounds to build envelops of the surface that are as tight as possible. Sharp bounds for non linear functions have been proposed in the works of Lutterkort [6] and Reif [9]. From these theoretical bases, Peters *et al.* introduced subdividable linear efficient variety enclosure, known as sleves [8]. In [11], Wu and Peters combine concepts from [4, 6] and proposed a technique to construct envelops for Loop subdivision surfaces.

This paper aims at the key problem of ray/free form surface intersection while keeping the computations as simple as possible and minimizing the required memory. We propose a very simple algorithm to accurately compute the intersection between a ray and a quadrangular Bézier surface patch of arbitrary bi-degree. Our algorithm does not require any heavy geometric construction, such as building envelopes as proposed in [4, 8]. While Campagna's

Bézier clipping algorithm requires two projections and the construction of upper and lower envelopes (or convex hull construction), our algorithm is simpler: it only requires one projection, and avoids min max detections. Furthermore, our algorithm, that is designed for quadrangular Bézier patches, can also be extended to regular Catmull-Clark and Doo-Sabin patches since these surfaces can be converted into Bézier surfaces. Our method has two major advantages: it requires to refine the control mesh only if there exist a potential intersection at a given precision, which allows to reduce the size of the data handled during the intersection test, and avoids geometric constructions and their necessary updates at each iteration.

The structure of the rest of the papers is as follows: section 2 presents the mathematical background on sharp bounds between a Bézier curve and its quasi-interpolating polygon. In section 3, we extend Zhang's results to tensor product surfaces for Bézier patches to build the quasi-interpolating polygon Bézier surfaces of arbitrary bi-degree. Section 4 presents our algorithm in detail, section 5 presents a comparison with Bézier clipping algorithm. We present our conclusions and future work in section 6.

2 Sharp bounds of Bézier curves

Reif and Lutterkort independently proposed bounds of the difference between spline polynomials and their control structures in [6, 9], where bounds on L_P-norm are given in terms of the maximum second order differences of the control points. The notations used in this paper are the notations used by Reif in [9].

Let $\mathbb{P}^d$ be the space of polynomials of degree less than or equal to d on the unit interval $[0, 1]$. We denote by $B^d = \left[B_0^d, \ldots, B_d^d\right]$ the row vector of the Bernstein polynomials of degree d, $H^d = \left[H_0^d, \ldots, H_d^d\right]$ the hat functions, piecewise linear functions, with respect to the points with parametric value $t_j^d = j/d$, $j = 0, \ldots, d$, and $L^d = B^d - H^d$ their difference. Any polynomial $g \in \mathbb{P}^d$ can be written in Bernstein-Bézier form as $g = B^d P$, where $P = [P_0, \ldots, P_d]^T \in \mathbb{R}^{d+1}$ is a column vector of real valued control points. The corresponding control polygon $h = H^d P$ is a piecewise linear function with values P_j at the break points t_j^d.

For all $t \in [0, 1]$, P being the $d + 1$ control points, P^* being the Greville's points, and Δ^2 the second differences of the control points, Reif's theorem 2.2 in [9] states:

$$\left\|L^d(t)P\right\|_\infty \leq \left\|L^d(t)P^*\right\|_\infty \left\|\Delta^2 P\right\|_\infty . \quad (1)$$

In theorem 3.1 in [7], Nairn *et al.* propose a similar bound of the deviation between a polynomial curve and its control polygon defined as:

$$\left\|L^d(t)P\right\|_\infty \leq N_\infty(d) \left\|\Delta^2 P\right\|_\infty . \quad (2)$$

Depending on the degree d of a curve, the Nairn's coefficient $N_\infty(d)$ is defined by:

$$N_\infty(d) = \frac{\lfloor d/2 \rfloor \lceil d/2 \rceil}{2d} . \quad (3)$$

In [12], Zhang and Wang proposed a better bound of the difference between a Bézier curve and its quasi-interpolating polygon, which is a broken line built from linear combinations of the Bézier control points. For a curve $C^d(t)$, with degree d, the points Q_i^d of the quasi-interpolating polygon Q^d are defined by:

$$\begin{aligned} &Q_0^d = P_0, Q_n^d = P_n, \\ &Q_k^d = \frac{P_{k-1} + 2P_k + P_{k+1}}{4}, k = 1, \cdots, d-1. \end{aligned} \quad (4)$$

This can be written using matrices by:

$$Q^d = M_d P . \quad (5)$$

As an example, for cubic Bézier curves, M_3 is defined as:

$$M_3 = \frac{1}{4}\begin{pmatrix} 4 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 4 \end{pmatrix} . \quad (6)$$

Following Reif's notation, the maximum deviation between a Bézier curve of degree d and its quasi-interpolating polygon $Q^d(t)$ can be written as:

$$\left\|C^d(t) - Q^d(t)\right\|_\infty = \left\|\left(B^d(t) - H^d(t)M_d\right)P\right\|_\infty .$$

In [12], Zang and Wang showed that the deviation between a Bézier curve $C(t)$ and its quasi-interpolating polygon $Q(t)$ is bounded by:

$$\left\|\left(B^d(t) - H^d(t)M_d\right)P\right\|_\infty \leq Z_\infty(d) \left\|\Delta^2 P\right\|_\infty . \quad (7)$$

The coefficient $Z_\infty(d)$ is function of the degree d of the curve, can explicitly be expressed as a function of the Nairn's coefficient [12], and is defined as:

$$\begin{aligned} &Z_\infty(2) = \frac{1}{16}, \text{ and} \\ &Z_\infty(d) = N_\infty(d) - \frac{1}{4}, \text{ for } d \geq 3 . \end{aligned} \quad (8)$$

Zhang and Wang recall values for $N_\infty(d)$ and $Z_\infty(d)$ for $d = 2, \cdots, 8$, which are respectively:

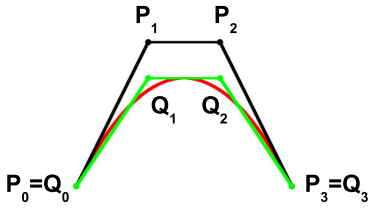


Figure 1. Quasi-interpolating polygon for a cubic Bezier curve

$$N_\infty(2,\cdots,8) = \left(\frac{1}{4},\frac{1}{3},\frac{1}{2},\frac{3}{5},\frac{3}{4},\frac{6}{7},1\right) \text{ and}$$
$$Z_\infty(2,\cdots,8) = \left(\frac{1}{16},\frac{1}{12},\frac{1}{4},\frac{7}{20},\frac{1}{2},\frac{17}{28},\frac{3}{4}\right).$$

Figure 1 shows an example of cubic Bézier curve and its corresponding quasi-interpolating polygon: in black is the control polygon, that corresponds to a broken line, that is the piecewise linear polynomial defined by $h = H^d P$ evaluated at break points points t_j^d, in red is the Bézier curve, and in green the quasi-interpolating polygon introduced by Zand and Wang, that corresponds to the broken line, that is the piecewise linear polynomial $m = H^d M_d P$ evaluated at break points points t_j^d.

3 Bounds for Bézier surfaces

In this section, we extend the quasi-interpolating polygon introduced par Whang and Wang for Bézier curves to Bézier surfaces. In the case of tensor product polynomials of bi-degree $d = (d_1, d_2)$, any polynomial $g \in \mathbb{P}^d$ can be written in Bernstein-Bézier form as $g(u,v) = B^{d_1}(u) P \left(B^{d_2}(v)\right)^T$, where P is a $(d_1+1)\times(d_2+1)$ matrix of control points. The control net corresponding to $g(u,v)$ is a piecewise bilinear function $h(u,v) = H^{d_1}(u) P \left(H^{d_2}(v)\right)^T$. The deviation between g and h is expressed by means of a linear operator $L^d(u,v)$ acting on matrices of control points and defined as:

$$L^d(u,v) = B^{d_1}(u) P \left(B^{d_2}(v)\right)^T - H^{d_1}(u) P \left(H^{d_2}(v)\right)^T.$$

Using Theorems 2.1 and 2.2 in [9], Reif extends his result to tensor product polynomials by:

$$\left\|L^d P\right\|_\infty \le \left\|L^{d_1} P^*\right\|_\infty \left\|\Delta_1^2 P\right\|_\infty + \left\|L^{d_2} P^*\right\|_\infty \left\|\Delta_2^2 P\right\|_\infty. \tag{9}$$

This can be written using Nairn's coefficients as:

$$\left\|L^d P\right\|_\infty \le N_\infty(d_1)\left\|\Delta_1^2 P\right\|_\infty + N_\infty(d_2)\left\|\Delta_2^2 P\right\|_\infty. \tag{10}$$

In other words, in the case of tensor product polynomials, the bound of the difference between a bilinear surface, which corresponds to the control net and the corresponding surface can be decomposed into a sum of bounds along each dimension. We recall the second forward differences are defined as:

$$\left\|\Delta_1^2 P\right\| = \max_{0\le i\le d}\left\|\Delta^2 P_i\right\|_\infty \text{ and}$$
$$\left\|\Delta_2^2 P\right\| = \max_{0\le i\le d}\left\|\Delta^2 (P_i)^T\right\|_\infty,$$

where $P_0,\ldots,P_d$ are the column vectors of P.

For Bézier patches of bi-degree $d = (d_1, d_2)$, the quasi-control polygon Q is defined as:

$$Q = M_{d_1} P (M_{d_2})^T, \tag{11}$$

where $P = (P_{ij})$ is a $(d_1+1)\times(d_2+1)$ matrix of control points, M_{d_1} and M_{d_2} are $(d_1+1)\times(d_1+1)$ and $(d_2+1)\times(d_2+1)$ Zang's square matrices, defined using equation 4, respectively.

The deviation between a bivariate Bézier surface $S(u,v)$ of bi-degree $d = (d_1, d_2)$ and its quasi-interpolating control polygon $Q(u,v)$ is defined as:

$$\|S(u,v) - Q(u,v)\| = \left\|S(u,v) - (MPM^T)(u,v)\right\| =$$
$$= \left\|B^{d_1}(u) P \left(B^{d_2}(v)\right)^T - H^{d_1}(u) M P M^T \left(H^{d_2}(v)\right)^T\right\|$$
$$= \left\|B^{d_1}(u) P \left(B^{d_2}(v)\right)^T - H^{d_1}(u) M P \left(\left(H^{d_2}(v)\right) M\right)^T\right\|$$

Since the maximum deviation $\|(B - HM)P\|_\infty$ is bounded (equation 7), we can apply Reif's extension to tensor product polynomials to determine a bound between a Bézier patch and its quasi-interpolating control net, defined by:

$$\begin{aligned}\|(S-Q)(u,v)\|_\infty &= \left\|(S^{d_1,d_2} - Q^{d_1,d_2})(u,v)\right\|_\infty \\ &\le Z_\infty(d_1)\left\|\Delta_1^2 P\right\|_\infty + Z_\infty(d_2)\left\|\Delta_2^2 P\right\|_\infty.\end{aligned} \tag{12}$$

As a result, we obtain a new bound for bi-degree Bézier surfaces over the interval $[0,1]^2$. For instance, in the case of a bicubic Bézier surface, *i.e.* $d_1 = d_2 = 3$ and $Z_\infty(3) = 1/12$, the bound of the maximum deviation is defined as:

$$\|(S-Q)\| \le \frac{1}{12}\left(\left\|\Delta_1^2 P\right\|_\infty + \left\|\Delta_2^2 P\right\|_\infty\right) \tag{13}$$

Figure 2 shows a bicubic Bézier patch with its control polygon and its quasi-interpolating control polygon.

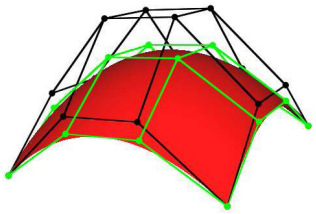


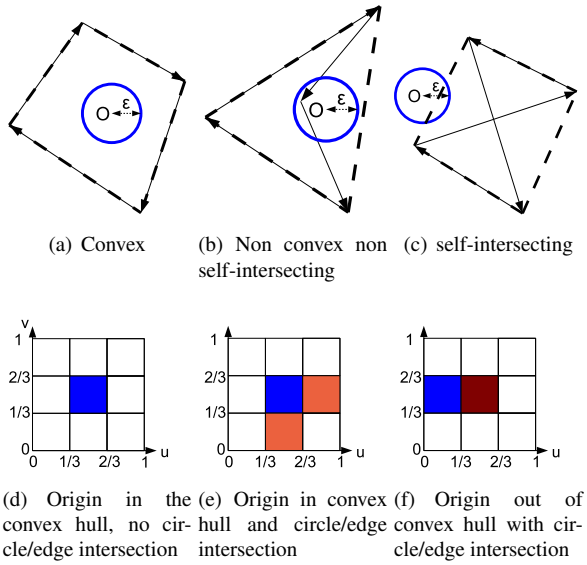
Figure 2. Quasi-interpolating polygon for a bicubic Bezier patch

In black is the control polygon, which corresponds to a piecewise bilinear polynomial defined by $h(u,v) = H^d1(u)P\left(H^d2(v)\right)^T$ evaluated at break points $(u_j^{d_1}, v_k^{d_2})$. In red is the bicubic Bézier surface, and in green is the quasi-interpolating polygon net, which corresponds to the piecewise bilinear polynomial defined by $m(u,v) = H^{d_1}(u)M_{d_1}P\left(H^{d_2}(v)M_{d_2}\right)^T$, evaluated at break points points $(u_j^{d_1}, v_k^{d_2})$.

4 Ray/Bézier surface intersection algorithm

Our algorithm is similar to Nishita's algorithm: it uses projection and Bézier clipping to determine a region around the intersection between a ray and a Bézier patch. Through several iterations, the algorithm refines the regions by cutting away parts of the surface until a user defined precision is reached. It proceeds as follows: the surface patch in input is considered to be a Bézier patch. In the case of B-Splines or regular Catmull-Clark patches, a change of basis has to be applied to build the corresponding Bézier control polygon as mentioned in [5]. At each iteration, the quasi-interpolating polygon of the current Bézier control net is computed using equation 4. The maximum deviation between the surface and the quasi-interpolating polygon, noted ϵ, is computed using equation 12. If ϵ is small enough (up to a user defined threshold), the patch is considered as flat, and direct ray/plane intersection is computed. Else, the quasi-interpolating polygon is projected onto an orthogonal plane to the ray, and several intersection tests are performed in 2D to determine which part may contain an intersection. The parts labeled as potentially containing an intersection are then clipped and stacked, and the others are discarded. The algorithm iterates this process until the patch stack is empty. The complete pipeline can be represented by the flowchart illustrated in figure 4.

To determine if there exists an intersection between the surface and a given ray, one can build an envelop for the quasi interpolating polygon. Unfortunately, this method is computationally expensive and increases the number of faces and vertices to model the envelop, before seeking for



(a) Convex (b) Non convex non self-intersecting (c) self-intersecting

(d) Origin in the convex hull, no circle/edge intersection (e) Origin in convex hull and circle/edge intersection (f) Origin out of convex hull with circle/edge intersection

Figure 3. The three cases of projected parametric squares onto orthogonal plane to a ray and associated detection types. The convex hull is in dashed line.

potential intersections. Our algorithm does not require any envelop construction, and therefore reduces the memory use and the number of computations. Instead of offsetting the quasi-interpolating polygon to construct an envelop, it is far simpler to offset the ray. If the ϵ-envelop of the ray, *i.e.* a cylinder, intersects the quasi-interpolating polygon, there may exist an intersection. The patch is then clipped to reduce the parametric interval that may contain an intersection. This technique has two major advantages: a cylinder is a very simple primitive with an implicit equation, which is very convenient for inside/outside tests and intersection detections. More importantly, the existence of an intersection between a cylinder and a quasi-interpolating control can be determined in 2D. Indeed, the quasi-interpolating control is composed of bilinear parametric squares, *i.e.* ruled surfaces. Therefore, the orthogonal projection of the borders of a bilinear surface onto a plane (in our case, the plane orthogonal to the ray) is a four sided polygon (non necessary convex). Thus, determining if the quasi-interpolating polygon intersects a cylinder is equivalent to determine in an orthogonal plane to the ray if the convex hull of its projection intersects a circle of radius ϵ.

The several cases that may arise are summed up on figure 3. The simplest case is shown on figure 3(a): the ϵ-circle is contained within the convex hull of the projected points and does not intersect the convex hull. Only the current parametric square is labeled. When the circle intersects one of the edges of the projected polygon, the parametric squares

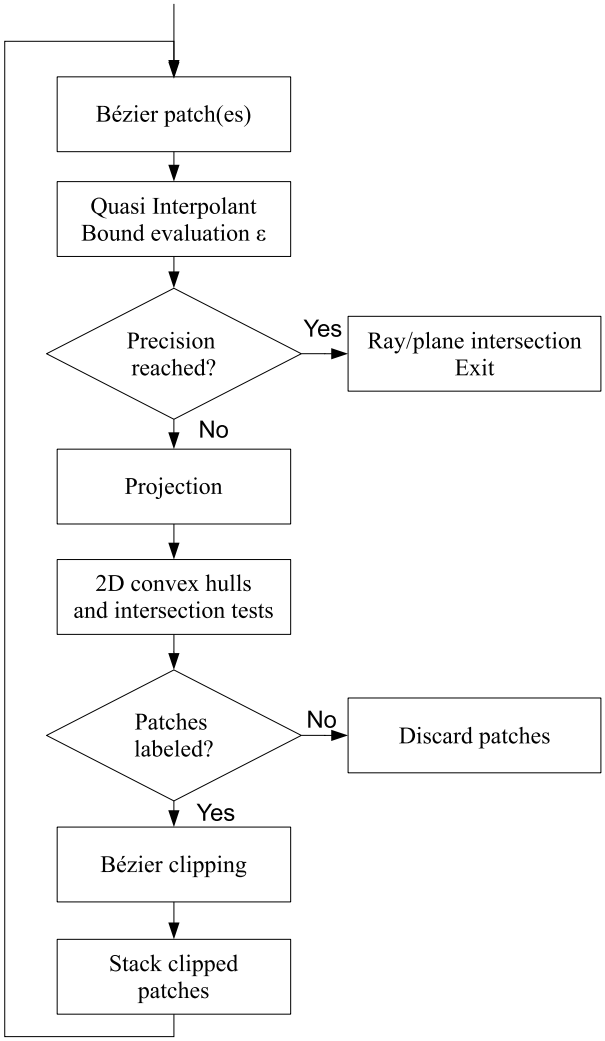


Figure 4. Ray/Bezier patch intersection algorithm

sharing the intersected edges are also labeled as containing an intersection. For instance, on figures 3(b) and 3(e), the origin is within the hull (previous case), but the circle intersect two edges. Therefore, the two parametric neighbors sharing the edges are also labeled as potentially containing an intersection, in orange on figure 3(e). There still exists a case that may appear: the center of the ϵ-circle is not contained in the convex hull but the circle intersects one or several edges of the convex hull: the two parametric squares associated are labeled as containing an intersection.

Figure 5 illustrates the result of our algorithm for the five first iterations and the final result. The control polygon of the Bézier patches is in black for the two first iterations, the quasi-interpolating polygons are in green in and their projections are in blue. On the projected plane, the ϵ-circle is drawn in red. The considered ray and the computed intersections and normals are drawn in red.

5 Comparison

First used by Nishita *et. al.* for ray tracing, Bézier clipping is used in an iterative algorithm that seeks all the solutions of the ray patch intersection problem up to an user definable precision ϵ within the parametric domain. The first step of every ray patch intersection algorithm is the reduction of the problem from 3D into a two-dimensional problem. Usually, the ray is represented as the intersection of two non parallel planes. Then, the parametric patch equation is substituted into the two planes equations, which discards the depth information along the ray. The roots of the two resulting equations are the parameters of the corresponding intersections between the ray and the Bézier patch. Nishita *et al.* determine the roots of these two equations using Newton iterations and by iteratively clipping away the parts of the patches that do not to intersect the ray. The computation is based on the convex hull property of Bézier patches. Campagna et al. [3] improved Nishita's algorithm by simply altering the 2D parameter directions and slightly enlarging the remaining region of interest, which led to fast, robust, and stable algorithm.

The major difference between our approach and Nishita's or Campagna's approach, is that our algorithm only uses elementary geometric operations and matrices products. There is no implicitization of the ray, nor there is any root finding of any polynomial. At each iteration, our algorithm first constructs a quasi-interpolating net. As shown before, the difference between Q.I. and the surface patch is bounded by a value ϵ. Since ϵ if defined as the maximum of second order differences of the control points of a given net, ϵ tends to zero as the considered control net becomes flatter. Therefore, ϵ can be seen as a flatness coefficient. As soon as ϵ as reached a user defined value, the Q.I. can be considered flat, and an intersection point can be computed as the intersection between a ray and a plane. In our case, the dimensional reduction from 3D to 2D is obtained by projecting the Q.I. net onto an orthogonal plane to the ray and by offsetting the ray by ϵ. This leads to a simpler problem where we isolate the corresponding parametric values of the projected quasi-interpolating polygon that intersects a circle of radius ϵ.

6 Conclusions and future work

In this paper, we have presented a new geometric algorithm to compute the intersection between a Bézier surface of arbitrary bi-degree and a ray. Our algorithm takes advantage of the recent developments on functions bounds, and

more specifically the construction of a piecewise bilinear surface, presented as the quasi-interpolating polygon, and the associated maximum deviation between this surface and the Bézier surface. To apply our algorithm to surfaces, we have extended the results of Zang and Wang from curves to tensor product surfaces using Reif's and Lutterkort works. Combined with well known clipping methods, our approach allows the determination of the intersection (and the associated normal to the surface) without any polynomial root finding using Newton iterations nor complex function hull computations. No heavy geometric construction, such as 3D envelops, is needed because the quasi-interpolating polygon iteration can be seen as the result of several matrix applications (Clipping matrix and Zang matrix) that are applied directly to the initial control net. Furthermore, all the computations performed are very simple: matrix multiplications (for computing the quasi-interpolating polygon, clipping, and projections), direct quadratic equations solving (for intersection tests between the projected edges and the ϵ-circle), and inner products of 2D vectors (for the 2D convex hull), which allows for promising improvements related to fast computations on GPU (especially with 4x4 matrices in the case of bicubic surfaces). The research perspectives are also numerous and concern several aspects: the extension to irregular surfaces, the improvement of the parametric reduction during the clipping, the extension to triangular Bézier surfaces, the integration and simulation of this approach in ray tracing, and its optimization using GPU.

References

[1] C. Benthin. *Real Time Ray Tracing on Current GPU Architecture*. PhD thesis, Saarlang University, January 2006.

[2] C. Benthin, I. Wald, and P. Slusallek. Interactive ray tracing of free-form surfaces. In *in In Proc. of Afrigraph*, 2004.

[3] S. Campagna, P. Slusallek, and H. Seidel. Ray tracing of spline surfaces: Bézier clipping, chebyshev clipping, and bounding volume hierarchy - a critical comparison with new results. *The Visual Computer*, 13(6):265–282, August 1997.

[4] L. Kobbelt, K. Daubert, and H. Seidel. Ray tracing of subdivision surfaces. In *in In Proc. of 9th Eurographics Workshop on Rendering*, pages 69–80, 1998.

[5] C. Loop. *Generalized B-spline Surfaces of Arbitrary Topological Type*. PhD thesis, University of Washington, 1992.

[6] D. Lutterkort. *Envelopes of Nonlinear Geometry*. PhD thesis, Purdue University, December 1999.

[7] D. Nairn, J. Peters, and D. Lutterkort. Quantitative bounds on the distance between a polynomial piece and its bézier control polygon. *Computer Aided Geometric Design*, 16(7):613–631, 1999.

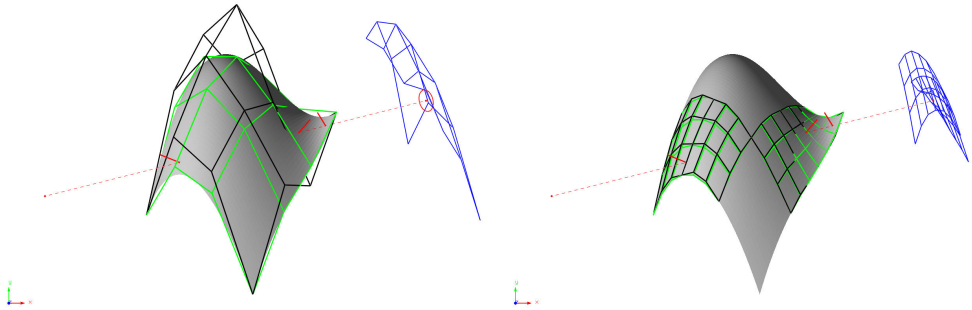
[8] J. Peters and X. Wu. Sleves for planar spline curves. *Computer Aided Geometric Design*, 21(6):615–635, 2004.

[9] U. Reif. Best bounds on the approximation of polynomials and splines by their control structure. *Computer Aided Geometric Design*, 17(6):579–589, 2000.

[10] S. Wang, Z. Shih, and R. Chang. An efficient and stable ray tracing algorithm for parametric surfaces. *Journal of Information Science and Engineering*, (18):541–561, 2001.

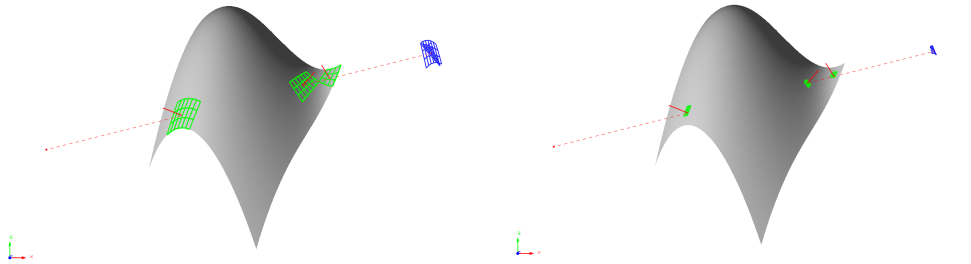
[11] X. Wu and J. Peters. Interference detection for subdivision surfaces. *Computer Graphics Forum*, 23(3):577–584, 2004.

[12] R. Zhang and G. Wang. Sharp bounds on the approximation of a bézier polynomial by its quasi-control polygon. *Computer Aided Geometric Design*, 23:1–16, 2006.



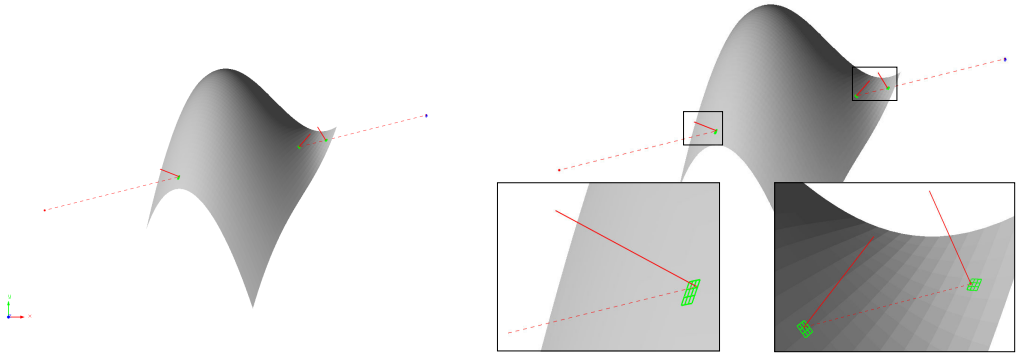
(a) First iteration: 4 subpatches potentially contain an intersection

(b) Second iteration: 5 potential intersections are detected



(c) Third iteration: 3 potential intersections are detected

(d) Fourth iteration: patch reduction around the 3 intersections



(e) Fifth iteration and final result

(f) Zoomed patches containing the 3 intersections, the patches dimensions are inferior to the face dimensions used for rendering the surface

Figure 5. Illustration of the refinement process. In grey is the surface, in red are the ray, the intersections detected and approximated normals, in green is the quasi-interpolating polygons constructed, in black the corresponding Bezier nets on a) and b). The projected quasi-interpolating polygons are in blue on the right of each figure, and the ϵ-circle is drawn in red.