Algorithm for directional projection of parametric curves onto B-spline surfaces

Hai-Chuan Song, Jun-Hai Yong

Algorithm for directional projection of parametric curves onto *B*-spline surfaces

Hai-Chuan Song[1,2,3,4], Jun-Hai Yong[1,3,4]

[1]School of Software, Tsinghua University, Beijing 100084, P. R. China
[2]Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P. R. China
[3]Key Laboratory for Information System Security, Ministry of Education of China, Beijing 100084, P. R. China
[4]Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, P. R. China

ABSTRACT

This paper proposes an algorithm for calculating the directional projection of parametric curves onto ***B***-spline surfaces. It consists of a second order tracing method with which we construct a polyline to approximate the pre-image curve of the directional projection curve in the parametric domain of the base surface. The final 3D approximate curve is obtained by mapping the approximate polyline onto the base surface. The Hausdorff distance between the exact directional projection curve and the approximate curve is controlled less than the user-specified distance tolerance. And the continuity of the approximate curve is $\varepsilon_T - G^1$, where ε_T is the user-specified angle tolerance. Experiments demonstrate that our algorithm is faster than the existing first order algorithm.

Keywords: Directional projection, ***B***-spline, Curves on surfaces, Approximation

1 INTRODUCTION

Directional projection of curves onto surfaces is widely implemented in CAD (computer aided design) systems [3],[12]. It plays fundamental roles in many operations, for example surface cutting, model designing, etc. It can also be utilized as a tool for constructing curves on surfaces. Given parametric curve $P(t)$ and surface $S(u,v)$ in 3D space, the directional projection of $P(t)$ onto S can be defined by the directional projection of the points of $P(t)$ onto S. Letting p denote an arbitrary point on $P(t)$, which is called the test point, the directional projection of p onto S is a set of points q on S such that the vector $(q-p)$ is parallel and in the same direction with a given vector Dir, which can be expressed as follows:

$$(q-p)\times Dir = 0,$$

and $(q-p)\cdot Dir > 0$ holds. As we move the test point p along $P(t)$, the motion of q results in a set of points on S, and that is the directional projection curve (see Fig. 1.).

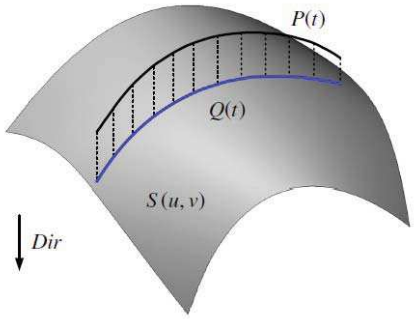

Fig. 1: The definition of the directional projection.

Denote the directional projection curve by $Q(t)$. To get $Q(t)$, first order algorithm were proposed in [16]. They derived the first order differential equation systems. By solving the system with numerical methods, they got a sequence of points along $Q(t)$. Then the approximate projection curve could be obtained using the interpolation method on the point sequence.

There are some drawbacks in the first order algorithm above. First, the step length of the points they got along $Q(t)$ is not well controlled (just by a constant parametric increment), which results in the uneven distribution of the points, and directly impact on the approximation result. Second, the approximate curve does not lie completely on S, which is not acceptable for many CAD applications such as surface blending and surface-surface intersection [19]. The approximation precision of the approximate curve cannot be controlled. Moreover, they cannot deal with the "jumping" projection case (this will be introduced in Subsection 3.4).

In order to overcome the drawbacks of the first order algorithm, we provide an algorithm dealing with the directional projection of parametric curves onto ***B***-spline surfaces, which approximates the exact directional projection curve $Q(t)$ with a piecewise curve on S. And this is similar to our published method [13], in which we deal with the orthogonal projection of parametric curves onto ***B***-spline surfaces.

According to the definition of the directional projection, the projection of a curve onto a surface can be regarded as the projections of every point of the curve onto the surface. And this is actually related to the intersections of lines and surfaces, which is also one of the key techniques that we will apply in our algorithm, and of which we will make brief survey below.

1.1 Line Surface Intersection

The intersection of line and surface is the fundamental problem in curve surface computation, and is of great use in geometric modeling, robotic techniques, collision detections, etc [14]. The subdivision method proposed by Whitted [11],[17] was firstly utilized to solve this problem, which is very easy and of high robustness. However, when elevating the precision requirement, the efficiency of this method will become low. As a transformation of the subdivision method, Nishita et al. [9] proposed the B*é*zier clipping method, which recursively divide the surface into regions that may contain the solution, and regions that do not contain the solution.

Generally speaking, the intersection of line and surface can be reduced into the solving of a nonlinear equation system. As a classic method, Newton-Raphson method was widely used in calculating the intersection of line and surface [1],[2],[5],[8],[10],[14],[15],[18]. The choice of the initial is very important for Newton-Raphson method to converge reliably. As a result, on one hand, some researches on whether the Newton-Raphson will converge with a given initial value were proposed. Toth [5] determined the converging interval of the Newton-Raphson method by using interval analyzing techniques. Lischinski and Gonczarowski [4] made use of the consistency of the surface and

improved Toth's method [5]. With the help of Kantorovich detection, Srijuntongsiri and Vavasis [14] can determine whether the convergence order of Newton-Raphson method is second order or not. On the other hand, some researchers considered how to provide a good initial value. Generally, they subdivide the surface recursively until every surface patches are flat enough, and the intersection points of the line and the bounding boxes of the surface patches are chosen as the initial value [1],[2],[8],[10],[18]. Wang et al. [15] combined the Bézier clipping method and Newton-Raphson method together.

2 OUTLINE OF OUR ALGORITHM

First, some symbols which will be used all through the paper will be introduced. In 3D space, given a parametric test curve $P(t)$ where $t \in [m,n]$, a normalized projection direction vector Dir and a ***B***–spline base surface $S(u,v)$ defined by:

$$S(u,v) = \sum_{i=0}^{n_u} \sum_{j=0}^{n_v} N_i^p(u) N_j^q(v) P_{i,j},$$

where $u \in [a,b]$, $v \in [c,d]$, $P_{i,j}$ are the control points, $N_i^p(u)$ and $N_j^q(v)$ are the *p*th-degree and *q*th-degree ***B***-spline basis functions, respectively (see Fig. 2.).

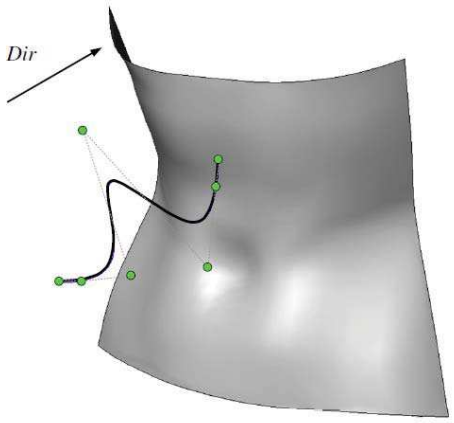

Fig. 2: Surface of the stomach of Venus: the original B-spline surface, the test curve and the projection direction.

We denote the directional projection curve of $P(t)$ onto S by $Q(t)$. Since $Q(t)$ lies on S, there exists a pre-image curve $q(t) = (u(t), v(t))^T$ of $Q(t)$ in the parametric domain of S and $Q(t) = S(u(t), v(t))$ holds. We suppose that $Q(t)$ is at least G^1 continuous and $q(t)$ is in Bézier representation.

We provide an algorithm which generates an approximate curve of $Q(t)$ on S. First, we construct a polyline (a continuous curve composed of one or more line segments) to approximate $q(t)$ in the parametric domain of S, taking the Hausdorff distance and angle tolerances into account. Then the final 3D piecewise approximate curve is obtained by mapping the approximate polyline onto S. The Hausdorff distance between $Q(t)$ and the final approximate curve is less than ε_D. And the continuity of the final approximate curve is $\varepsilon_T - G^1$, which means that for any pair of adjacent curves in a

piecewise curve, the angle between the end derivatives at the mutual point of them is less than ε_T. Here ε_D and ε_T are the user-specified tolerances. The main algorithm flow is described as follows:

1. Compute the directional projection of the starting point of $P(t)$ onto S with the line surface intersection method [15].
2. Based on the projection point, with a second order tracing method, an array of points along $q(t)$ is derived. Set the array of points as the vertices, and the initial approximate polyline is obtained in the parametric domain of S.
3. According to the user-specified Hausdorff distance tolerance ε_D, discrete points are sampled at specified positions on $q(t)$ and added into the approximate polyline so as to guarantee that the Hausdorff distance between the final approximate curve and $Q(t)$ is less than ε_D.
4. According to the user-specified angle tolerance ε_T, split $q(t)$ at specified position to make sure that the final approximate curve is at least $\varepsilon_T - G^1$ continuous.
5. After the steps above, the final approximate polyline in the parametric domain of S is obtained. Map the polyline onto S using blossoming techniques [6],[7] and the 3D approximate curve of $Q(t)$ is obtained.

The rest of the paper is organized as follows. Section 3 introduces the second order tracing method, and constructs the initial approximate polyline in the parametric domain of S. In Section 4, the approximation precision and the continuity of the final approximate projection curve are controlled, using our published method [13]. Section 5 shows the experimental results and Section 6 concludes the paper.

3 CONSTRUCTING THE INITIAL APPROXIMATE POLYLINE OF THE DIRECTIONAL PROJECTION IN THE PARAMETRIC DOMAIN OF S

First, the initial approximate polyline is to be constructed in the parametric domain of S with our second order tracing method, which can be summarized in the following steps:

1. Compute the first directional projection of the points on $P(t)$ onto S, and take the projection point as the current tracing point.
2. Compute the first and second order derivatives of $q(t)$ at the current tracing point in the parametric domain of S and locally approximate $q(t)$ with the second order Taylor polynomial.
3. Get the parametric increment using the constant arc length increment strategy. Then the new test point on $P(t)$ and the estimated position of the next tracing point are obtained.
4. Refine the accuracy of the estimation with the Newton-Raphson method. Connect the result point of the refinement and the current tracing point with a line segment, and add it at the end of the approximate polyline. Take this refinement result point as the new tracing point.
5. Repeat steps 2 ~ 4 until the test point we get reaches the end of $P(t)$ and the initial approximate polyline is obtained.

3.1 Preparation for Tracing

Suppose P_0 to be the starting point of $P(t)$. To start the tracing we need to compute the directional projection points of P_0 onto $S(u,v)$. For these points lies on a line L, which passes P_0 and is parallel with Dir, so we derive these points utilizing line surface intersection method. We first divide the ***B***-spline surface $S(u,v)$ into several B*é*zier surface patches (see Fig.3.). Then we can just take advantage of the division result to compute the directional projection of P_0. And it can be figured out in the following three steps:

1. First we take the surface patches, whose control polygons intersect with L, as the candidate surface patches. According to the convex hull ability of Bézier surfaces, other surfaces can be pruned. So we can reduce the range of the computation.
2. Then for every candidate surface patch, use the line surface intersection method [15] to compute the intersection of L with it. And each intersection point is taken as one of the candidate intersection points of L with the base surface.
3. Eliminate the intersection points, which do not satisfy the condition $(p - P_0) \cdot Dir > 0$, from the candidate intersection points.

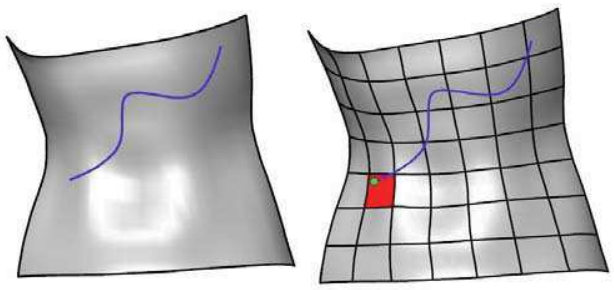

Fig. 3: Surface of the stomach of Venus: (a) the original B-spline surface; (b) the division result and the candidate surface patch.

So we can get the intersection points of L and the base surface. Assume that one of the projection points of P_0 onto S is Q_0, meanwhile in the parametric domain of S, we can get the corresponding pre-image point q_0 on $q(t)$.

To control the tracing process, we preserve two points $cur\_P \in \mathbf{R}^3$ and $cur\_Q \in \mathbf{R}^2$ to store the values of the current test point on $P(t)$ and the corresponding current tracing point on $q(t)$. cur_P and cur_Q have the same parameter value on $P(t)$ and $q(t)$, respectively.

After the directional projection of P_0, the point q_0 is taken as the current tracing point cur_Q and P_0 is taken as the current test point cur_P. To keep on tracing, we need to get the next tracing point according to cur_Q. Note that there may exist more than one intersection point of L and the base surface, and for each intersection point, we take it as a seed point, and trace the corresponding projection curve, respectively. Taking one projection curve for example, now we will introduce how to construct the approximate curve of the projection curve.

3.2 The First and Second Order Derivatives of the Directional Projection Curve

We estimate the next tracing point with the second order Taylor expansion of $q(t)$ at cur_Q, so we need to compute the first and second order derivatives of $q(t)$ at cur_Q. Now we will deduce the first and second order differential equation systems satisfied by the directional projection. Assume that S is at least C^2 continuous locally.

First, recall the definition of the directional projection:

$$(q - p) \times Dir = 0, \tag{3.1}$$

where p is the point on the test curve $P(t)$, q is the corresponding point on the projection curve $Q(t)$, Dir is the projection direction vector, $S(u,v)$ is the base surface, and $Q(t) = S(u(t), v(t))$ holds.

Take the derivative of Eqn. (3.1) on both sides with respect to t, and it follows that:

$$\left(S_u \frac{du}{dt} + S_v \frac{dv}{dt} - \frac{dp}{dt}\right) \times Dir = 0.$$

According to the distributive law of cross product, we have:

$$(S_u \times Dir)\frac{du}{dt} + (S_v \times Dir)\frac{dv}{dt} = \frac{dp}{dt} \times Dir,$$

multiply S_u and S_v on both sides of the equation above, respectively, and it follows that:

$$\begin{cases} S_v \cdot \left(S_u \times Dir\right)\dfrac{du}{dt} = S_v \cdot \left(\dfrac{dp}{dt} \times Dir\right) \\ S_u \cdot \left(S_v \times Dir\right)\dfrac{dv}{dt} = S_u \cdot \left(\dfrac{dp}{dt} \times Dir\right) \end{cases}.$$

Solve the system above, we have:

$$\begin{cases} \dfrac{du}{dt} = \dfrac{S_v \cdot \left(\dfrac{dp}{dt} \times Dir\right)}{S_v \cdot \left(S_u \times Dir\right)} \\ \dfrac{dv}{dt} = \dfrac{S_u \cdot \left(\dfrac{dp}{dt} \times Dir\right)}{S_u \cdot \left(S_v \times Dir\right)} \end{cases}. \tag{3.2}$$

Eqn. (3.2) is just the first order differential equation system of the directional projection, and this was first derived by [16]. Moreover, take the derivative of Eqn. (3.2) on both sides with respect to t, we have:

$$\begin{cases} \dfrac{d^2u}{dt^2} = \dfrac{S_v \cdot \left(\dfrac{d^2p}{dt^2} \times Dir\right) + \left(S_{vu}\dfrac{du}{dt} + S_{vv}\dfrac{dv}{dt}\right) \cdot \left(\dfrac{dp}{dt} \times Dir\right)}{S_v \cdot \left(\left(S_{uu}\dfrac{du}{dt} + S_{uv}\dfrac{dv}{dt}\right) \times Dir\right) + \left(S_{vu}\dfrac{du}{dt} + S_{vv}\dfrac{dv}{dt}\right) \cdot \left(S_u \times Dir\right)} \\ \dfrac{d^2v}{dt^2} = \dfrac{S_u \cdot \left(\dfrac{d^2p}{dt^2} \times Dir\right) + \left(S_{uu}\dfrac{du}{dt} + S_{uv}\dfrac{dv}{dt}\right) \cdot \left(\dfrac{dp}{dt} \times Dir\right)}{S_u \cdot \left(\left(S_{vu}\dfrac{du}{dt} + S_{vv}\dfrac{dv}{dt}\right) \times Dir\right) + \left(S_{uu}\dfrac{du}{dt} + S_{uv}\dfrac{dv}{dt}\right) \cdot \left(S_v \times Dir\right)} \end{cases}. \tag{3.3}$$

And Eqn. (3.3) is just the second order differential equation system of the direction projection.

3.3 Searching For the Next Tracing Point

We estimate the position of the next tracing point with the second order Taylor expansion of $q(t)$ at cur_Q, whose expression is as follows:

$$q(t_0 + \Delta t) = q(t_0) + q'(t_0)\Delta t + \frac{q''(t_0)\Delta t^2}{2} + o(\Delta t^2), \tag{3.4}$$

where $q(t_0) = cur\_Q,$, Δt is the parametric increment and $o(\Delta t^2)$ is the second order Taylor remainder, which is a vector-valued function $f(\Delta t)$ such that $\lim_{\Delta t \to 0} f(\Delta t) / \Delta t^2 = 0$. By omitting the remainder of the expansion, the polynomial follows:

$$\overline{q}(t_0+\Delta t)=q(t_0)+q^{'}(t_0)\Delta t+\frac{q^{''}(t_0)\Delta t^2}{2}, \tag{3.5}$$

with which we approximate $q(t)$ locally. Given the value of t_0, we can get the values of $q^{'}(t_0)$ and $q^{''}(t_0)$ with Eqn. (3.2) and (3.3), respectively. So the only unknown factor in Eqn. (3.5) is Δt, which directly impacts on the distribution of the sampling points and the approximation result.

In order to obtain an evenly distributed sampling during the generation process of the initial approximate polyline, we use the step length controlling method based on constant arc length increment used in [13] as follows.

Let ds and dt denote the arc differential and parameter differential of $Q(t)$, respectively. It follows that:

$$ds=\left|Q^{'}(t)\right|dt.$$

Further, according to the chain rule, we can get the expression of $Q^{'}(t)$:

$$Q^{'}(t)=S_u\frac{du}{dt}+S_v\frac{dv}{dt}, \tag{3.6}$$

where the values of S_u, S_v, $\frac{du}{dt}$, $\frac{dv}{dt}$ can all be calculated with the given value of t_0, respectively. Hence, with the user-specified arc length increment Δs on $Q(t)$, the corresponding approximate parametric increment Δt is determined by:

$$\Delta t=\frac{\Delta s}{\left|Q^{'}(t)\right|}. \tag{3.7}$$

Substitute Δt into Eqn. (3.5), the estimated position of the next tracing point on $q(t)$ can be obtained as $\overline{q}(t_0+\Delta t)=(\overline{u},\overline{v})$.

This step length controlling method is incomplete in [13], because it is hard to estimate the total arc length of the projection curve before we can generate it. And improper arc length increments may leads to the wrong results. In the case of directional projection, the arc length of the projection curve $Q(t)$ can be estimated like this. We can first sample on the test curve $P(t)$, link the sample points with a polyline, with which we approximate $P(t)$. Then we directionally project this polyline onto the control polygons, which are only composed of planar triangles, of the Bézier surface patches we derived in Subsection 3.1. The directional projections of polyline onto triangles can be computed in linear time and are much easier to realize, whose projection result curve is also a polyline. Then the length of the projection result polyline is taken as the approximate arc length of the projection curve $Q(t)$, and we can choose a proper arc length increment according to the estimated total arc length.

Usually, the estimated point deviates from the precise point we desire, so an error exists. According to the definition of the directional projection in Eqn. (3.1), we define the error of the estimation as follows:

$$\varepsilon=\frac{\left\|Dir\times\left(S(\overline{u},\overline{v})-cur\_P\right)\right\|}{\left\|S(\overline{u},\overline{v})-cur\_P\right\|}, \tag{3.8}$$

where $(\overline{u},\overline{v})=\overline{q}(t_0+\Delta t)$ is the estimated point generated by our tracing method in the parametric domain, $S(\overline{u},\overline{v})$ is the 3D image point of $(\overline{u},\overline{v})$ on S, Dir is the normalized projection direction. The geometric significance of ε is the absolute value of the sine of the angle between Dir and $(S(\overline{u},\overline{v})-cur\_P)$. Specially, when $S(\overline{u},\overline{v})$ coincides with cur_P, Eqn. (3.8) becomes invalid for the

denominator equals to 0. In this situation, we let the error $\varepsilon = 0$, because cur_P lies on S now and $S(\overline{u},\overline{v})$ is just the projection point of it.

As a result, to get the precise point, a refinement is needed. Take the point $P(t_0 + \Delta t)$ as the new test point cur_P. We introduce the following Newton-Raphson method based refinement method.

Multiply S_u and S_v on both side of Eqn. (3.1), respectively, and set $r = ((S(u,v) - p) \times Dir)$. We have the following equation system:

$$\begin{cases} f(u,v) = ((S(u,v) - p) \times Dir) \cdot S_u(u,v) = r \cdot S_u(u,v) = 0 \\ g(u,v) = ((S(u,v) - p) \times Dir) \cdot S_v(u,v) = r \cdot S_v(u,v) = 0 \end{cases}, \tag{3.9}$$

Solve the equation system above with Newton-Raphson method, where we set:

$$\delta_i = \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = \begin{bmatrix} u_{i+1} - u_i \\ v_{i+1} - v_i \end{bmatrix},$$

$$J_i = \begin{bmatrix} f_u & f_v \\ g_u & g_v \end{bmatrix} = \begin{bmatrix} S_{uu} \cdot r & S_{uv} \cdot r + S_u \cdot (Dir \times S_v) \\ S_{uv} \cdot r + S_v \cdot (Dir \times S_u) & S_{vv} \cdot r \end{bmatrix},$$

$$k_i = -\begin{bmatrix} f(u_i, v_i) \\ g(u_i, v_i) \end{bmatrix}.$$

Then the parameter increment of the iteration can be obtained by solving:

$$J_i \delta_i = k_i.$$

Then the convergence criteria of the iteration are as follows:

1. $\left\| (u_{i+1} - u_i) S_u(u_i, v_i) + (v_{i+1} - v_i) S_v(u_i, v_i) \right\| \le \varepsilon_1$,
2. $\left\| S(u_i, v_i) - cur\_P \right\| \le \varepsilon_1$,
3. $\dfrac{\left| Dir \times \left(S(u_i, v_i) - p \right) \right|}{\left\| Dir \right\| \cdot \left\| S(u_i, v_i) - p \right\|} \le \varepsilon_2$,

where (u_i, v_i) is the parameter of the *i*th iteration, ε_1 and ε_2 two zero tolerances of Euclidean distance and sine. The iteration is halted if any of the three conditions above is satisfied. In the iterations above, we set the initial parameter value equals to $(\overline{u},\overline{v})$, and set $p = cur\_P$ in the above Newton-Raphson method. The convergence of the iteration depends on the errors of the estimated initial values. As shown in the experimental results in Section 5, the average estimation error of our algorithm is comparable with that of the first order algorithms [16] implemented with Runge-Kutta method, which have been widely used.

Let $next\_Q \in \mathbf{R}^2$ denote the result point of the refinement in the parametric domain of S. Connect cur_Q and $next\_Q$ with a line segment L, and add L into the approximate polyline. Take $next\_Q$ as the new tracing point and we continue seeking for the next tracing point.

The tracing will not stop until the test point we obtain reaches the end of $P(t)$. Then we can get the initial approximate polyline of $q(t)$ in the parametric domain of S.

3.4 Projection Jumping Checking

In the generating process of the approximate projection curve, the directional projection points of the test point on $P(t)$ onto S may "jump", with the increment of the parameter (see Fig. 4.). The

projection line L_1 of the current tracing point $P(t_0)$ only intersects with surface in the back part (the point Q_1 in Fig. 4.). However the projection line L_2 of the next tracing point $P(t_0 + \Delta t)$ intersects both in front and back parts of the surface (the points Q_2 and $Q_2^{'}$ in Fig. 4.). And this is called the "jumping" of the intersection points. If we still only trace the projection curve in the back part of the surface, the should-exist projection curve in the front part of the surface will be missed. We deal with this case as follows.

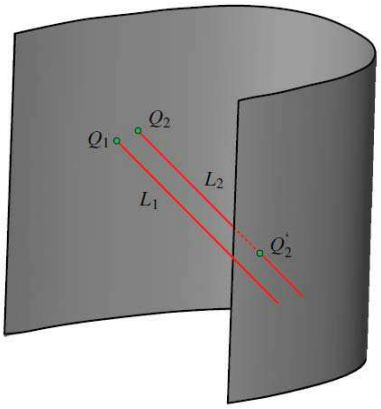

Fig. 4: The jumping of the intersection points.

For each projection curve $Q_i(t)$, we record the tag $Active\_Patch_i$ representing the surface patch the $cur\_Q_i(t)$ of $Q_i(t)$ lies in. When we compute the next projection point, we construct the projection lines L_2 which start from the point $P(t_0 + \Delta t)$ and is parallel to Dir, and L_1 which start from the point $P(t_0)$ and is parallel to Dir. Compute the intersection of L_2 and the control polygons of all surface patches. For every surface patch $Inter\_Patch$, whose control polygon intersects with L_2, we check whether the "jumping" happens with the following criteria:

1. $Inter\_Patch$ is not the neighbour of any of $Active\_Patch_i$.
2. $Inter\_Patch$ is the neighbour of some of $Active\_Patch_i$, but the distance from the point $S(cur\_Q_i)$ to the common edge of $Inter\_Patch$ and $Active\_Patch_i$ is bigger than Δs, the user specified arc length increment.
3. $Inter\_Patch$ is just one of $Active\_Patch_i$, but the number of intersection points of L_2 and the control polygon of $Inter\_Patch$ is more than that of L_1 (In this case, according to the variation diminishing property of the Bézier surface, the number of intersection points of L_2 and $Inter\_Patch$ is may be more than that of L_1).
4. $Inter\_Patch$ is just one of $Active\_Patch_i$, but the distance from any of intersection points of L_2 and the control polygon of $Inter\_Patch$ to the corresponding intersection point of L_1 is bigger than Δs.

The “jumping” may happen if any of the four conditions above is satisfied. Under this condition, we need to compute the intersection points of L_2 with the base surface $S(u,v)$ using line surface intersection method [15] as we did in Subsection 3.1. For each intersection point derived, match it with the estimated point (we got in Subsection 3.3) of current tracing points cur_Q_i (each estimated point can only match to one nearest intersection point within the distance Δs). Each matched intersection point is taken as the new tracing point of the corresponding projection curve. And each unmatched intersection point is taken as a new seed point which is the start point of a new segment of projection curve.

Another special case of “jumping” is the projection curve crosses the boundary of the surface, where the number of intersection points of L_2 and $S(u,v)$ is less than that of L_1. This can be detected like this.

1. The estimated point (we got in Subsection 3.3) of current tracing points cur_Q_i exists the boundary of the surface.
2. The number of intersection points of L_2 and the control polygons of the surface patches is less than that of L_1.

The “jumping” may happen if any of the two conditions above is satisfied. Under this condition, we need to compute the intersection points of L_2 with the base surface $S(u,v)$ using line surface intersection method [15] as we did in Subsection 3.1. If L_2 does not intersect the surface at any projection curve, the tracing of the corresponding projection curve should be stopped. Fig. 5. shows our processing result of this “jumping” case.

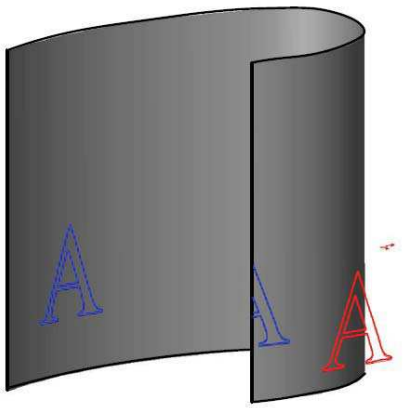

Fig. 5: The processing result of jumping.

Recall that the first order algorithm [16] only solve the first differential equation systems with numerical method, so it cannot deal with this “jumping” case, which frequently appears in practical.

4 CONTROLLING THE APPROXIMATION PRECISION AND THE CONTINUITY OF THE FINAL APPROXIMATE PROJECTION CURVE

After the construction of the initial approximate polyline in the parametric domain of $S(u,v)$, we control the approximation precision and the continuity of the final approximate curve with the same method we used in [13], which adds more sampling points of $Q(t)$ into the approximate polyline and

guarantees that the Hausdorff distance between the final 3D approximate curve and the exact projection curve $Q(t)$ is less than the user-specified Hausdorff distance tolerance ε_D, and the continuity of the final 3D approximate curve is $\varepsilon_T - G^1$, where ε_T is the user-specified angle tolerance. More details can be found in [13].

With the final approximate polyline in the parametric domain of S, we map the polyline to the Bézier surfaces we get in Subsection 3.1 using blossoming techniques [6],[7], and the corresponding 3D approximate curve of the exact directional projection curve $Q(t)$ on S is obtained, which is in Bézier form and lies completely on S. The degree of the 3D approximate curve is $p+q$, where p and q are the *u*-direction degree and *v*-direction degree of the ***B***-spline surface S, respectively.

5 EXPERIMENTAL RESULTS AND COMPARISONS

We present several examples for the directional projection. All the experiments are implemented with Microsoft Visual Studio 2005 with windows XP on the same PC with Intel Core2 Duo CPU 2.53 GHz, 1GB Memory. In all of our experiments below $\varepsilon_1 = \varepsilon_2 = 10^{-10}$, where ε_1, ε_2 are two convergence tolerances for Newton-Raphson method introduced in Subsection 3.3.

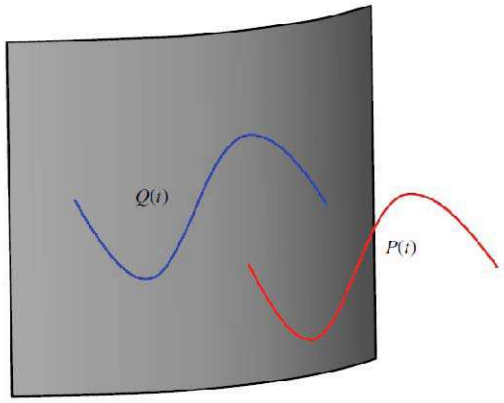

Fig. 6: **Example 1**, Directional projection of a cubic ***B***-spline curve onto the bicubic ***B***-spline free form surface where the projection direction is opposite to z axis.

We implement the first order algorithms in [16] with fourth order Runge-Kutta method (R-K4), which is also the recommended method in [16]. Comparisons between the first order algorithms [16] and our second order tracing method on efficiency and accuracy are performed during the process of generating the initial approximate polyline in **Example 1**, where a cubic ***B***-spline curve $P(t)$ is directionally projected onto a 5×1 order Bézier surface, as illustrated in Fig. 6.. Specifically, the 6 control points of $P(t)$ are:

$$\begin{cases} P_0 = (0.00, -43.75, 3.86)^T \\ P_1 = (0.00, -40.13, -3.86)^T \\ P_2 = (0.00, -23.80, -31.58)^T \\ P_3 = (0.00, -11.87, 33.86)^T \\ P_4 = (0.00, 10.60, 6.40)^T \\ P_5 = (0.00, 16.70, -2.27)^T \end{cases}.$$

And the knot vector is: (0:00; 0:00; 0:00; 0:00; 0:26; 0:68; 1:00; 1:00; 1:00; 1:00): The 6×1 control points of the surface are:

$$\begin{cases} P_{00}=(-98.75,32.50,-40)^T \\ P_{10}=(-91.77,22.86,-40)^T \\ P_{20}=(-81.48,1.23,-40)^T \\ P_{30}=(-86.57,-32.46,-40)^T \\ P_{40}=(-93.98,-50.36,-40)^T \\ P_{50}=(-97.84,-57.27,-40)^T \end{cases}, \begin{cases} P_{01}=(-98.75,32.50,40)^T \\ P_{11}=(-91.77,22.86,40)^T \\ P_{21}=(-81.48,1.23,40)^T \\ P_{31}=(-86.57,-32.46,40)^T \\ P_{41}=(-93.98,-50.36,40)^T \\ P_{51}=(-97.84,-57.27,40)^T \end{cases}.$$

For the purpose of equity, constant parametric increment strategy, as Runge-Kutta method does, are utilized in both of the two kinds of algorithms. Further, according to this paper, both of the two kinds of algorithms generate the initial approximate polyline by estimating the position of the tracing point. In order to measure the accuracy of the two kinds of algorithms, we record the estimation errors, defined in Eqn. (3.8), for each estimated point of the two kinds of algorithms, and compute the average estimation errors for each parametric increment of the two kinds of algorithms. And the refinement based on the Newton-Raphson method introduced in Subsection 3.3 is performed in both of the two kinds of algorithms. The comparison results of the two kinds of algorithms on **Example 1** are recorded in Tab. 1..

Increment	*Time (ms)*		*Errors*		*Number of discrete points*
Δt	*R- K4 [16]*	*ours*	*R- K4 [16]*	*ours*	
1×10^{-2}	1.927	1.486	2.14749e-010	2.1971e-010	101
1×10^{-3}	14.443	9.372	1.02919e-012	2.27651e-010	1001
1×10^{-4}	151.814	96.347	1.71291e-014	9.19197e-014	10001
1×10^{-5}	1698.425	1076.167	1.9516e-014	1.67022e-014	100001
1×10^{-6}	17359.394	7194.017	1.49422e-014	1.42424e-014	1000001

Tab. 1: Time cost and error comparison for **Example 1**.

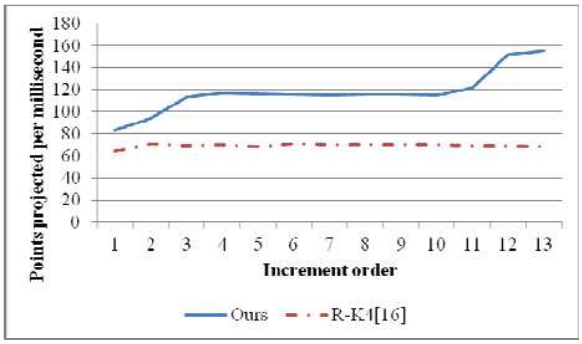

Fig. 7: Efficiency comparison between our second order tracing method and the first order algorithms for **Example 1**.

As shown in Tab. 1., as we decrease the parametric increment, the average errors decrease, the time costs and the numbers of discrete points increase. We first consider the errors. For each parametric increment, the average errors of our algorithm are comparable with those of the first order

algorithm with R-K4, and are even smaller than the first order algorithm when the parametric increment decreases to 1×10^{-5}. As for the time costs, the difference of the two kinds of algorithms can be clearly illustrated in Fig. 7. Our algorithm is about 2 times faster than the first order algorithms implemented in R-K4 method. And the gaps are getting larger as the parametric increment decreases.

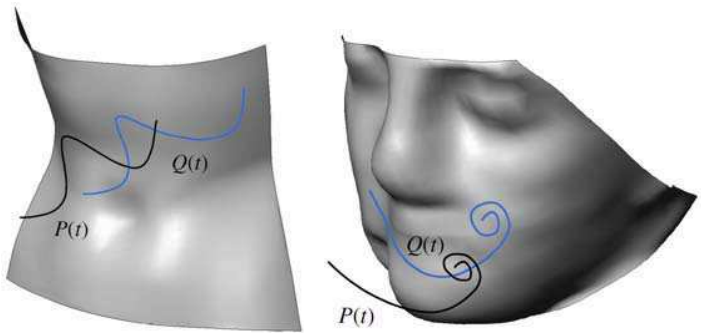

Fig. 8: Directional projection of a cubic ***B***-spline curve onto the bicubic ***B***-spline free form surface where the projection direction is opposite to z axis, $\varepsilon_D = 1\times10^{-3}$, $\varepsilon_T = 10^{\circ}$: (a) **Example 2**, surface of the stomach of Venus; (b) **Example 3**, surface of woman face.

Besides the comparisons with the first order algorithm during the process of generating the initial approximate polyline, we also test the performance of our algorithm for the whole projection process, which uses the constant arc length increment strategy, performs "jumping" checking and controls the approximation precision and the continuity of the final approximate projection curve. Fig. 8. shows the directional projection of cubic ***B***-spline curves onto the bicubic ***B***-spline free form surfaces. The results of our algorithm on **Example 2** and **Example 3** are shown in Tab. 2..

	Example 2		***Example 3***	
Arc length increment Δs	0.05	0.05	0.05	0.05
Tolerance ($\varepsilon_D / \varepsilon_T$)	$10^{-3}/10^{\circ}$	$10^{-3}/1^{\circ}$	$10^{-3}/10^{\circ}$	$10^{-3}/1^{\circ}$
Degree	6	6	6	6
Number of segments	90	121	97	126
Number of control points	541	723	583	757
Continuity	$10^{\circ}-G^1$	$1^{\circ}-G^1$	$10^{\circ}-G^1$	$1^{\circ}-G^1$
Time (ms)	187.036	215.652	240.406	313.298

Tab. 2: Results of our algorithm for the Examples.

As we can see from the results, the efficiency of the algorithm meets the real-time requirement. Moreover, the approximation precision and the continuity of the final approximate projection curve can be controlled.

Then we present two examples for "jumping" case projection, which we introduced in Subsection 3.4. Fig. 9(a). and 9(b). plot the "jumping" case projections, where two cubic ***B***-spline curves are directionally projected onto a tour (transformed into NURBS form) and a free form ***B***-spline surface, respectively. The solid blue curves are the test curve, while the curve with green break points is the 3D approximate projection curve. As we can see from the results, the "jumping" is correctly detected, and new segment of projection curves are added into the final projection result, and the projection curve stops at the boundary of the surface.

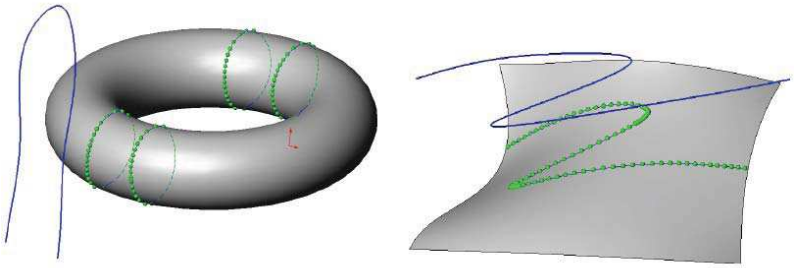

Fig. 9: “jumping” case projection.

Then, more examples of our algorithm are shown in Fig. 10. and Fig. 11..

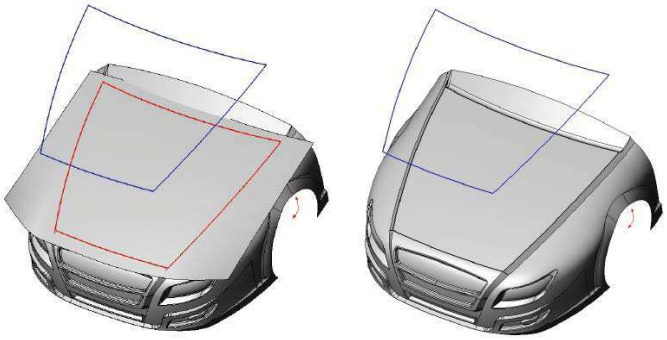

Fig. 10: The design of the front cover of a car.

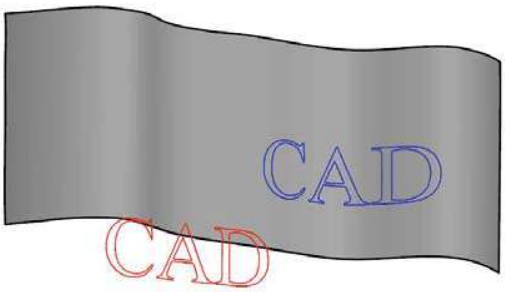

Fig. 11: Construction of curves on surface.

6 CONCLUSION

This paper presents an approximation algorithm for directional projection of parametric curves onto ***B***-spline surfaces. The second order tracing method is applied to construct the initial approximate polyline in the parametric domain of the surface. With the initial approximate polyline, further

sampling on the exact directional projection curve is performed to control the approximation precision and the continuity of the final 3D approximate curve.

The main contribution of this paper is the derivation of the second order differential equation system of the directional projection, with which we can compute the farthest points and the Hausdorff distances between the curve segments and their corresponding line segments in the parametric domain using iteration methods, which enables us to control the approximation precision and the continuity of the approximate curve [13].

Experimental results indicate that the accuracy of our algorithm is comparable with that of the first order algorithm [16], and at the same time our algorithm is faster than it. Moreover, our algorithm can deal with the "jumping" projection case, and control the approximation precision and the continuity of the approximate curve, while the first order algorithm [16] cannot do this.

ACKNOWLEDGEMENTS

The research was supported by Chinese 973 Program (2010CB328001) and Chinese 863 Program (2012AA040902). The first author was supported by the NSFC (61035002, 61272235). The second author was supported by the NSFC (61063029, 61173077).

REFERENCES

[1] Alain, F.; John, B.: Chebyshev polynomials for boxing and intersections of parametric curves and surfaces, Computer Graphics Forum 13 (3), 1994, 127–142.
[2] Barth, W.; W. Sturzlinger,: Efficient ray tracing for Bézier and B-spline surfaces, Computer and Graphics 17 (4), 1993, 423–430.
[3] CATIA, http://www.catia.com/, Dassault Systems.
[4] Daniel, L.; Jakob, G.: Improved techniques for ray tracing parametric surfaces, The Visual Computer 6 (3), 1990, 134–152.
[5] Daniel, T.: On Ray Tracing Parametric Surfaces, Computer Graphics (SIGGRAPH'85 Proceedings) 18 (4), 1985, 171–179.
[6] DeRose, T. D.: Composing Bézier simplexes, ACM Transactions on Graphics 7 (3), 1988, 198–221.
[7] DeRose, T. D.; Goldman, R. N.; Hagen, H.; Mann, S.: Functional composition algorithm via blossoming, ACM Transactions on Graphics 12 (2), 1993, 113–135.
[8] Michael, S.; Richard, B.: Ray Tracing Free-Form B-spline Surfaces, IEEE Computer Graphics and Application 6 (2), 1986, 41–49.
[9] Nishita, T.; Sederberg, T. W.; Kakimoto, M.: Ray tracing trimmed rational surface patches, Computer Graphics (SIGGRAPH'90 Proceedings) 24 (4), 1990, 337–345.
[10] Qin, K.-H.; Gong, M.-L.; Guan, Y.-J.; Wang, W.-P.: New method for speeding up ray tracing NURBS surfaces, Computer Graphics 21 (5), 1997, 577–586.
[11] Rubin, S. M.; Whitted, T.: 3-Dimensional Representation For Fast Rendering Of Complex Scenes, Computer Graphics (SIGGRAPH'80 Proceedings) 14 (3), 1980, 110–116.
[12] SolidWorks, http://www.solidworks.com/, Dassault Systems.
[13] Song, H.-C.; Yong, J.-H.; Yang, Y.-J.; Liu, X.-M.: Algorithm for orthogonal projection of parametric curves onto B-spline surfaces, Computer Aided Design 43 (4), 2011, 381–393.
[14] Srijuntongsiri, G.; Vavasis, S. A.: A condition number analysis of a line-surface intersection algorithm, SIAM Journal on Scientific Computing 30 (2), 2007, 1064–1081.
[15] Wang, S.-W.; Shih, Z.-C.; Chang, R.-C.: An efficient and stable ray tracing algorithm for parametric surfaces, Journal of Information Science and Engineering 18 (4), 2002, 541–561.
[16] Wang, X.-P.; Wei, W.; Zhang, W.-Z.: Projecting curves onto free-form surfaces, International Journal of Computer Applications in Technology, 37 (2), 2010, 153–159.
[17] Whitted, T.: Improved Illumination Model For Shaded Display, Communications of the ACM 23 (6), 1980, 343–349.
[18] Yang, C.-G.: On Speeding Up Ray Tracing Of B-spline Surfaces, Computer Aided Design 19 (3), 1987, 122–130.
[19] Yang, Y.-J.; Cao, S.; Yong, J.-H.; Zhang, H.; Paul, J.-C.; Sun, J.-G.; Gu, H.-j.: Approximate computation of curves on B-spline surfaces, Computer Aided Design 40 (2), 2008, 223–234.